International Journal of Engineering Innovation & Research
Volume 2, Issue 1, ISSN: 2277 – 5668

# Dynamic Storage Assurance on Cloud Computing

**P. Dhanalakshmi**
PG Scholar
dhanamjai55@rediffmail.com

**V. Ramesh**
Asst.Professor
ramesh_8607@yahoo.co.in

***Abstract* - Cloud computing investigate the problem of data security in cloud data storage, which is essentially a distributed storage system. Distributed storage integrity auditing mechanism, utilizes the homomorphic token and distributed erasure-coded data. The proposed design allows users to audit the cloud storage with very lightweight communication and computation cost. The auditing result not only ensures strong cloud storage correctness guarantee, but also simultaneously achieves fast data error localization, i.e., the identification of misbehaving server and recover the corrupted data. Considering the cloud data are dynamic in nature, the proposed design further supports secure and efficient dynamic operations on outsourced data, including block modification, deletion, and append. Analysis shows the proposed scheme is highly efficient and resilient against Byzantine failure, malicious data modification attack, and even server colluding attacks. With the advent of data storage and new technology trends that result in new failure modes for storage, interesting challenges arise in ensuring data integrity and security. In this paper, we discuss the cause of integrity violations and implementation issues to perform efficient integrity assurance. The main aim of this paper is to prevent the file from integrity violations and recovering the corrupted file with low cost overhead.**

***Keywords* – Cloud computing, Third Party Auditor, Error Localization, Distributed Storage, Token Computation.**

## I. INTRODUCTION

In cloud data storage system, users store their data remotely on clouds, so that the correctness and availability of data files must be guaranteed to be identical. Enabling public auditability for cloud storage is of critical importance so that owners can resort to a specialized third party auditor (TPA) to audit cloud storage services and maintain strong storage correctness guarantee, while saving their own precious computing resources. Third Party auditing, in case the user does not have the time, feasibility or resources to perform the storage correctness verification, he can optionally delegate this task to an independent third party auditor, making the cloud storage publicly verifiable. TPA should not learn user's data content through the delegated data auditing. The data blocks are computed as tokens and assign signature for each tokens. The tokens are passed through multiple servers. Compare the tokens at the client machine with the tokens precomputed in the server. If anyone of the token gets missed, it shows that some of the server is corrupting the data blocks. Retransmit the corrupted data to recover it by Reed Solomon Algorithm. In this paper, we explore the technique used to detect the corrupted blocks easily using homomorphic token precomputation and then Reed Solomon Erasure Coded Technique to acquire the desired blocks from different servers.

## II. CLOUD STORAGE SYSTEM

Cloud storage architecture has three entities:
User: User performs data storage and retrieval operations
Cloud server (CS): provides data storage service
Third Party Auditor (TPA): (optional) trusted to access and expose risk of cloud storage services

User can upload and stores the data into cloud. It can also use the services provided by Cloud Service Provider from the cloud. Users store data into set of distributed cloud servers.

TPA is the optional entity, used to perform some computations instead of users. If the user has not having enough resources to compute tokens or required hardware support then he can easily delegate the work to a third party auditor. Now, TPA is responsible to generate homomorphic token and stores the token securely for further verification [1],[2].

As in Fig. 1, the user will access the file from the cloud server and TPA will perform the public auditing instead of the user to reduce storage overhead.

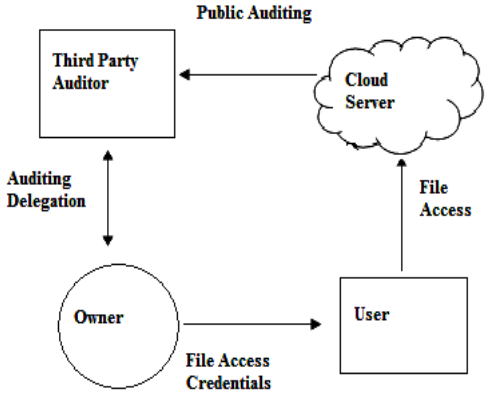



Fig.1. Cloud Storage System

Table I: Token Computation Parameters

| | |
|---|---|
| F | File to be stored |
| D | Secret Vector Matrix |
| PRF | Pseudo Random Function |
| PRP | Pseudo Random Permutation |
| C | Encoded File Matrix |
| $K_{chal}$ | Challenge Key |

## III. SECURED DATA STORAGE

With cloud computing, users need not know about network and server maintenance. They can access different servers around the world without necessarily knowing where the servers are located [8].

*File Dispersion*

In cloud data storage, disperse the data file F across set of n=m+k distributed servers [4]. R (m+k, k) is the Reed-Solomon erasure-correcting code used to create k redundancy parity vectors from m data vectors. By placing each m+k vectors on different servers, the data file can survive the failure of any k without any loss and with a space overhead k/m.

Each file F is represented as column vector.

All blocks are elements of Galois Field GF (2p).

Information dispersal matrix A can be written as,

A=(I|P)

Where I is the m*m identity matrix

P is the secret parity generation matrix.

Table II: Notations

| | |
|---|---|
| l | Data Vector size in blocks |
| GF | Galois Field |
| n,m | Parity matrix with row and column vectors |
| P | Parity Generation Matrix |
| w | Parity Vector |

*Token Generation*

The token based approach is reasonable because both, cryptographic coprocessors and standardized interfaces exist that can be used for such tokens [1], [4].

Each token has a random subset of blocks α (i). Each server computes the signature for each individual block of tokens. The user verifies the cloud server t times to ensure the data storage correctness. Precompute t verification tokens for each individual vector G (i) using a PRF f (.), PRP φ (.), a challenge key $K_{chal}$ and a master permutation key $K_{PRP}$.

*Algorithm: Token Generation*

1. Choose file F and get length of the file
2. Generate Secret Vector Matrix D on file F.
3. Generate Checksum Matrix C=D*P, where P is the Parity Vector.
4. Compute Number of Tokens to be taken.

T=F+FL+V

where

T- Number of Tokens to be computed

F- File block size

FL- Length of the File

V- Secret Vector per verification.

5. Compute the tokens by pseudorandom function & pseudorandom permutation function.
6. Compute key signature for each token

Token=Token+Signature

7. Store the precomputed tokens on the cloud server for further verification on client.
8. Disperse the file over the Cloud.

*Fault Tolerance Verification*

TPA will send the file by dividing the file into equal sized blocks and generate a small token signature for each block along with a secret key. The tokens were calculated based on the mathematical calculations with HAIL based techniques [6]. Store the pre-computed tokens at the server, each block is sent with a short signature. After transmission, at the client side, the TPA checks whether the tokens are same or not by comparing the token signature. If the token blocks are same, it stores it in the client or it means it may effect to data corruption.

*Algorithm: Byzantine Fault Tolerance*

1. Client sends the request to server
2. Assign sequence number by using the PREPARE message
3. The server replies for the request with COMMIT message.
4. Request from the client are signed with a private key
5. The client waits for replies from multiple servers
6. Request with invalid signature gets discarded
7. Client ensures that whether signatures from all the servers are received correctly or not.
8. If the client doesn't receive enough replies during a time interval, represents that one of the server is misbehaving.
9. Resends the corrupted token.
10. Simply acts like Challenge-Response Protocol.

*Fault Recovery*

Reed-Solomon codes are block-based error correcting codes with a wide range of applications in digital storage data [5]. Reed-Solomon codes are used to correct errors. The Reed-Solomon encoder takes a block of digital data and adds extra redundant bits. The Reed-Solomon decoder processes each block and attempts to correct errors and recover the original data. It helps to recover the corrupted by retransmitting the corrupted data.

*Algorithm: Fault Recovery*

1. Encoder takes n storage servers holding data (D1..Dn). Each data holds k bytes.
2. Each server holds m parity bits.
3. Calculate parity bits for each block of data using function F (D1.Dn) as shown in Fig. 2
4. Compute Parity Matrix using Vander monde matrix
5. Perform arithmetic operation of Galois Field to find whether all the tokens are integrated at the client side
6. If any one of the m server fails, they can be reconstructed from the surviving devices
7. Recover the failure using Gaussian Elimination

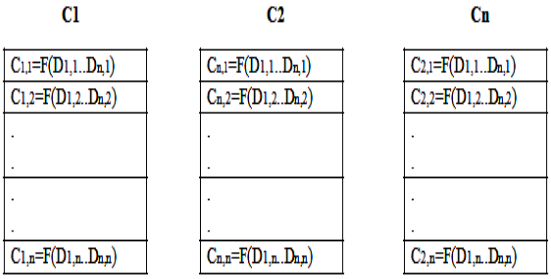



Fig.2. Reed Solomon Parity Checksum

## IV. Storage Correctness Realization

In cloud data storage, there are many potential scenarios where data stored in the cloud is dynamic, like electronic documents, photos, or log files etc. Therefore, it is crucial to consider the dynamic case, where a user may wish to perform various operations of update, delete and append to modify the data file while maintaining the storage

correctness assurance. The straightforward and trivial way to support these operations is for user to download all the data from the cloud servers and re-compute the whole parity blocks as well as verification tokens. This would clearly be highly inefficient.

*Updation*

In cloud data storage, sometimes the user may need to modify some data stored in the cloud, from its current value to a new one. Refer this operation as data update.

To perform update operation on particular data block client need to recalculate the verification token on updated data. Also client need to update this value of newly calculated token to all the replicas of file in storage cloud.

When user want to perform update operation, the splitted file from all storage servers is merged and given to the user to perform data updates.

Once user has finished with the updating the data, new tokens are calculated on whole file and they are stored on main cloud server. After this, updated file is splitted back and dispersed onto corresponding cloud storage servers. Update operations include modifying file, inserting data, as well as deleting data from file.

*Deletion*

Sometimes, after being stored in the cloud, certain data may need to be deleted. The delete operation we are considering is a general one.

When user wants to delete some file, he can simply delete it. In delete operation, file blocks that are distributed among cloud storage servers are all deleted. Once file is deleted, we cannot perform any recovery of deleted files as there won't be any backup available in main cloud server.

*Append*

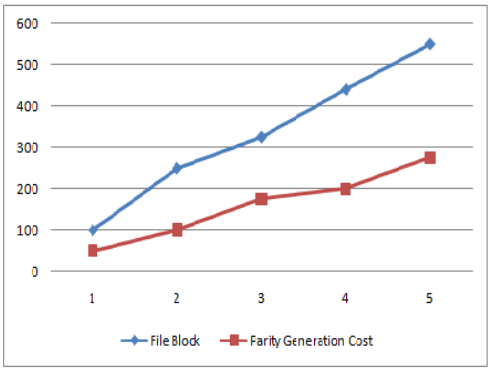
In some cases, the user may want to increase the size of his stored data in file by adding data at the end of the data file, which we refer as data append operation. So in case of append operation whenever user append data to his file, new verification tokens are calculated & stored on main cloud server & file is splitted as before and dispersed among the cloud storage servers.

In case of insert operation, we are treating as a part of update operation and we are relaying on update operation for insert operation.

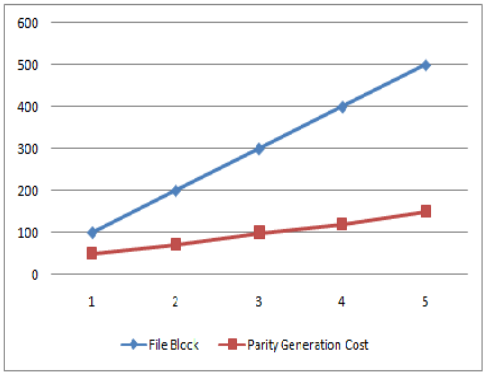
## V. EVALUATION PERFORMANCE

Redundancy parity vectors are calculated via multiplying the file matrix F by secret parity generation matrix P [4], [5].

Pick blocks from among the data and parity vectors to establish a set of m.k linear equations. Once having the knowledge of P, server can modify any part of the data blocks and calculate corresponding parity blocks.




(a)




(b)

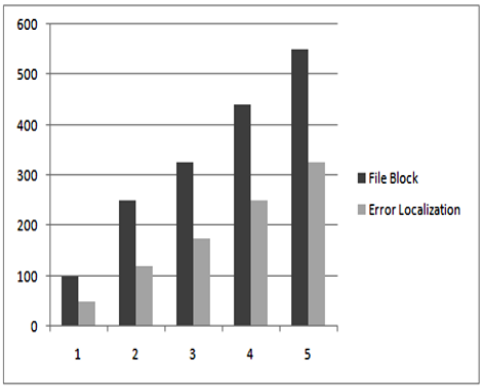
Fig.3. Performance Comparison between two Different Parameters.
(a) Variable File Block with Parity Generation Cost
(b) Fixed File Block with Parity Generation Cost

K determines how many parity vectors are required before outsourcing. Growth of k means large number of parity blocks required to be blinded.

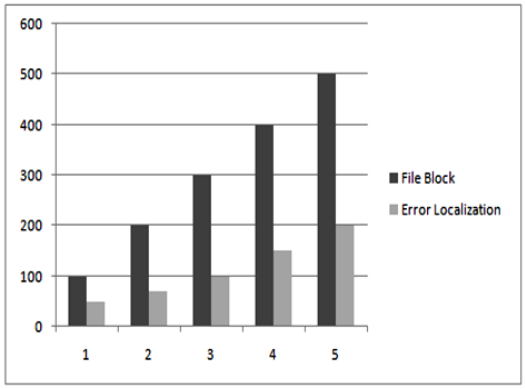
In Fig. 3.a, the files with variable file size blocks will increase the parity vector size according to the file token size. Hence, the parity generation cost will also be increased with increase in file size.

In Fig. 3.b, the fixed file blocks have fixed parity vector size. Hence, the cost is limited according to the token precomputed by the owner.

Take the Tokens dynamically to find the corruptions in the cloud servers. If we choose random data for token precomputation it will not check the entire data hence it cannot detect the corruptions systematically.




(a)




(b)

Fig.4. Detection of data distribution when tokens computed dynamically
(a) Variable File Block (b) Fixed File Block

Fig. 4, the file blocks are taken dynamically. If the file blocks are variable size (Fig. 4.a), it is difficult to find the error localization since the file blocks are integrated with variable sizes i.e., some blocks of smaller size and some blocks of larger size. If the file blocks are fixed (Fig. 4.b), the integration of file is simple and hence it is easy to find the error occurrence in the distribution of files.

In this paper, we are evaluating the parity blocks with low cost overhead and increasing the performance speed. This performance can be achieved by large tokens on pre computation.

Data are continuously distributed through multiple servers in cloud. The token is computed dynamically. If data lost, then it must find out that which server gets corrupted. It is identified by the byzantine fault tolerance algorithm.

## VI. CONCLUSION

The user challenges the cloud server with a set of fixed block of files and assign signature for the tokens. The tokens are distributed through multiple servers to the owner. The signature at the user should match with the tokens pre computed by the user. When the tokens are computed for the fixed size tokens, this will reduce the computational overhead. Growth of parity block size k increases with variable m size file blocks will increases the parity generation cost. Using fixed file blocks will randomly reduce the increase in parity block size; hence the overall parity generation cost can be reduced. From this paper, we uniquely combine the efficient algorithms to achieve secure, scalable and fine-grained access control on outsourced data in the cloud.

## REFERENCES

[1] A. Juels and J. Burton S. Kaliski, "Pors: Proofs of retrievability for large files," in Proc. of CCS'07, Alexandria, VA, October 2007, pp. 584–597.

[2] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for storage security in cloud computing," in Proc. of IEEE INFOCOM'10, San Diego, CA, USA, March 2010.

[3] C. Wang, K. Ren, W. Lou, and J. Li, "Towards publicly auditable secure cloud data storage services," IEEE Network Magazine, vol. 24, no. 4, pp. 19–24, 2010.

[4] C. Wang, Q. Wang, K. Ren, and W. Lou, "Towards Secure and Dependable Storage Services in Cloud Computing", in IEEE Transactions On Services Computing, vol.5, no.2, pp. 220-232, 2012.

[5] C.Wang, Q.Wang, K.Ren and W.Lou, "Ensuring data storage security in cloud computing", proc.17th Intl Workshop Quality of Service (IWQoS'09),pp.1-9,July 2009.

[6] K. D. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in Proc. of CCS'09, 2009, pp. 187–198.

[7] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in Proc. of ESORICS'09, volume 5789 of LNCS. Springer-Verlag, Sep. 2009, pp. 355–370.

[8] C. Cahin, R.Guerraoui, L.Radrigies, Introduction to reliable and secure distributed programming, 2nd edition, Published by Springer, 2011.

[9] http://msdn.microsoft.com/en-us/library/windowsazure/ee706734.aspx

[10] http://www.sit.fraunhofer.de/content/dam/sit/en/studies/Cloud-Storage-Security_a4.pdf

[11] https://cloudsecurityalliance.org/

[12] http://www.opensciencegrid.org/bin/view/storage/

[13] http://www.geni-orca.org/trac/wiki/CloudStorage

## AUTHOR'S PROFILE

**Ms. P. Dhanalakshmi**
The author is currently a ME student in Computer Science and Engineering Department at Kalasalingam Institute of Technology. She had completed BE from PSR Engineering College; affiliated to Anna University.

**Mr. V. Ramesh**
The author is an Assistant Professor in Computer Science and Engineering Department at Kalasalingam Institute of Technology. He received his BE from Syed Ammal Engineering College; affiliated to Anna University, and M. Tech., Degree from Kalasalingam University. His research interests are in the areas of network security and cloud computing security.