 ISBN: 978-81-927230-0-6

Public Verifiability of Stored Data in Cloud using Disassembly of Data Segment

Adhikrao.Y.Jadhav
Department of Computer Science & Engg.

Sachin P. Patil
Department of Information Technology ,
Annasaheb Dange College of Engg. & Technology, Ashta
India

Dr. Shivaji D. Mundhe,
Director - MCA - SIMCA
Sinhgad Institute of Management and Computer Application

EMail : director_mca_simca@sinhgad.edu,
drshivaji.mundhe@gmail.com

***Abstract*— A distributed cryptographic system is introduced that allows a set of servers to prove to a client that a stored file is intact and retrievable. The system strengthens, formally unifies, and streamlines distinct approaches from the cryptographic and distributed-systems communities. It cryptographically verifies and reactively reallocates file shares. It is robust against an active, mobile adversary (one that may progressively corrupt the full set of servers). We are trying to propose a strong, formal adversarial model for Public Verifiability, and rigorous analysis.**

Index Terms*— Component, formatting, style, styling, insert. *(key words)

I. INTRODUCTION

Cloud storage represents a family of on-line services for archiving, backup and even primary storage of files. Example: Amazon S3, Google. Cloud-storage providers offer clean and simple file-system interfaces to the user by hiding the complexities of hardware management. The cryptographic community has proposed tools called proofs of retrievability (PORs)[4] and proofs of data possession (PDPs)[5] for security assurance. A POR is a challenge response protocol that enables client to verify that a file is retrievable or not from cloud service provider. The benefit of a POR over simple transmission of file is efficiency. The response can be highly compact, and the client can complete the proof using a small fraction of file. As a standalone tool for testing file retrievability against a single server, though, a POR is of limited value. Detecting that a file is corrupted is not helpful if the file is irretrievable and the client has no recourse.

Public verifiability is useful in environments where file is distributed across multiple systems. In this, file is stored in redundant form across multiple servers. A client can test the availability of file on individual servers via a POR. If it detects corruption within a given server, it can appeal to the other servers for file recovery.

In a distributed file system, a file *is* spread across servers with redundancy through an erasure code. This supports file recovery in server errors or failures. It can help a client to check the integrity of file by retrieving fragments of file from individual servers and cross-checking their consistency. The system manages file integrity and availability across a collection of servers. It allows use of PORs as tool by which storage resources can be tested and reallocated when failures are detected.

II. PROPOSED SYSTEM

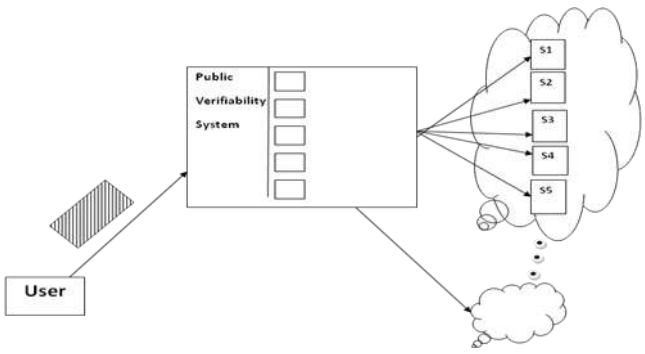


Fig. 1. Architecture of the System

Whenever user wants to store a file on cloud, our system will disperse the entire file in n fragments and stored on n servers instead of replicating entire file on n servers as shown in fig.1.. We can distribute it using an error-correcting code referred as dispersal code. To overcome the problem of corruption attacks we will use **Message Authentication Code (MAC),** computed with secret key known to the client. To make it highly available, we will replicate these data and codebase on different cloud environments using appropriate mechanism. The system will periodically checks the intactness

and retrievability of file by aggregating responses from servers and cross checking it with codeword present in the codebase.

Following Modules are developed in the system,

1. Devising vector using dispersal code
2. Vector replication at each point in cloud
3. Assembly and verification of data at a point in cloud
4. Public verifiability

III WORK

A file is spread across multiple servers with redundancy through an dispersal code as shown in Fig.2. This supports file recovery in server errors or failures. It can help a client to check the integrity of file by retrieving fragments of file from individual servers and cross-checking their consistency. The system manages file integrity and availability across a collection of servers. It allows use of PORs as tool by which storage resources can be tested and reallocated when failures are detected.

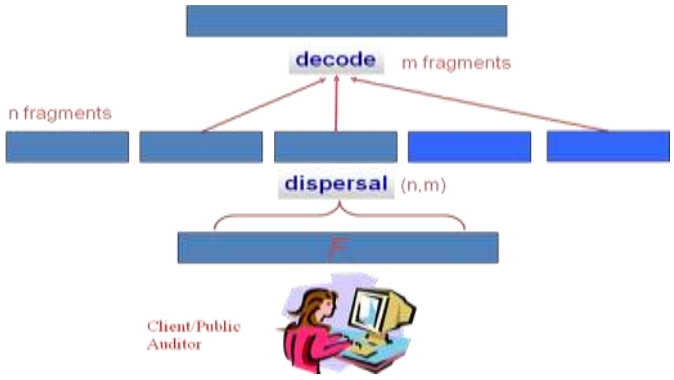


Fig. 2. File Distribution using dispersal code

Module 1: Devising vector using Dispersal code

A file is distributed using the **dispersal code**. Each file block is individually distributed across the n servers under the dispersal code as shown in Fig.3.

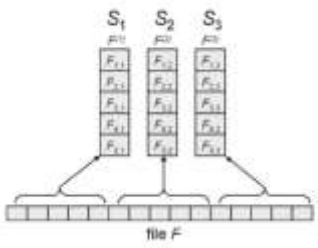


Fig. 3a Dispersal of File

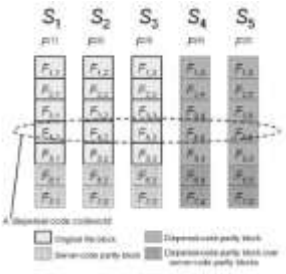


Fig.3b Encoding of file and encoded file with parity blocks

added for both the server and dispersal codes.

To increase the lifetime of a file the **Message Authentication Code(MAC)** is embedded into the dispersal code. MAC may be constructed as the straightforward composition of a **Universal Hash Function (UHF)** with a **pseudorandom function (PRF).**

Module 2: Vector replication at each point in the cloud

The vector generated in the Module 1 is replicated in the codebase present the different cloud data servers.

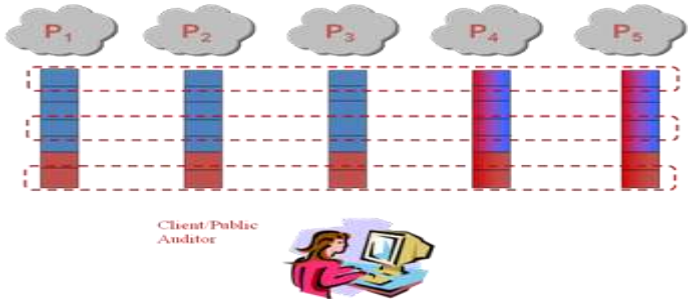


Fig. 4 Replication of vector stripes on different storage server

Module 3: Assembly and verification of data at a point in the cloud

As in the module 2, the file is replicated on different servers. Whenever the request comes from client to check intactness and retrievability of file. We will assemble a stripe from servers and crosscheck with the codeword present in the codebase. Verify data integrity by throwing challenges to some or all of the servers. The stripe generated during response is crosschecked with codeword present in codebase.

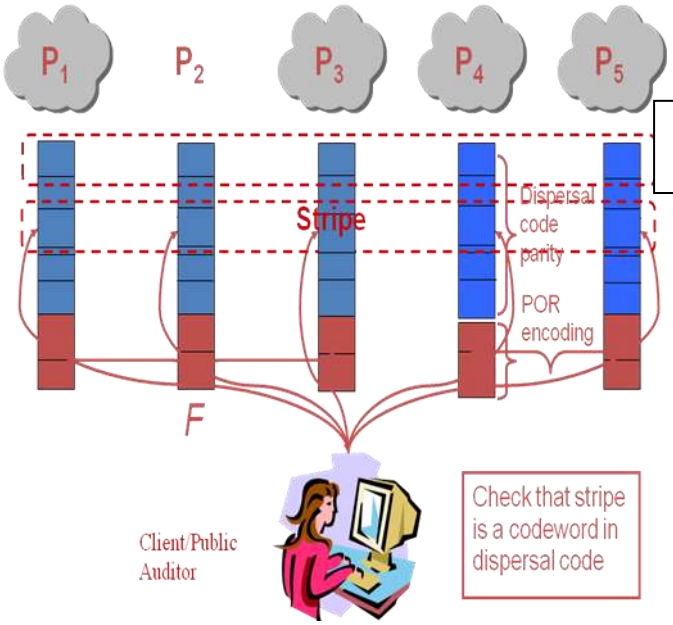


Fig.5 Checking the integrity of the vector at a point in cloud

The System implements a distributed file system in the cloud environment using dispersal code. During fragmentation, vector information is generated and replicated on all servers. The client will request for checking integrity of data. It throws challenge using secrete key to some or all servers. In a response a server will generate a stripe. The stripe is crosschecked with codeword present in the codebase. If it matches, then the data is intact and retrievable.

Module 4: Public Verifiability

Public verifiability allows anyone, not just the client (data owner), to challenge the cloud server for correctness of data storage while keeping no private information. Then, clients are able to delegate the evaluation of the service performance to an independent third party auditor, without devotion of their computation resources. In the cloud, the clients themselves are unreliable or cannot afford the overhead of performing frequent integrity checks. Thus, for practical use, it seems more rational to equip the verification protocol with public verifiability.

Whenever client wants to check integrity of data he need to forward its request to primary server where we can combine all replicated vectors and check whether data is intact or not. Every time it is not necessary that client should check data integrity. Primary server can also do the same. For that, we have scheduled tasks which are running on primary server. If any error occurs and any file fragment has been lost or modified then primary server can collect message digest for that fragment from anyone available secondary server.

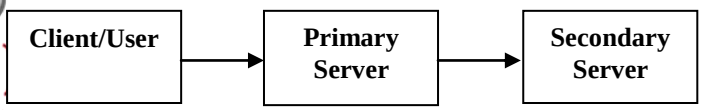


Fig.6.

Implementation of public verifiability using Merkle Hash Tree(MHT) algorithm.

MHT has Authentication structure – To Prove set of elements are undamaged and unaltered.
MHT is Binary Tree – Leaves in the MHT are the hashes of authentic data values.

MHT has following functions -

- (pk, sk) ← KeyGen(1k). This function is run by the client. It takes as input security parameter 1k, and returns public key pk and private key sk.
- (Φ, sigsk(H(R))) ← SigGen(sk, F). This function is run by the client. It takes as input private key sk and a file F which is an ordered collection of blocks {mi}, and outputs the signature set Φ, which is an ordered collection of signatures {σi} on {mi}. It also outputs metadata-the signature sigsk(H(R)) of the root R of a Merkle hash tree. In our construction, the leaf nodes of the Merkle hash tree are hashes of H(mi).
- (P) ← GenProof(F, Φ, chal). This function is run on server. It takes as input a file F, its signatures Φ, and a challenge chal. It outputs a data integrity proof P for the blocks specified by chal.
- {TRUE, FALSE} ← V erifyProof(pk, chal, P). This algorithm can be run by
  either the client or the third party auditor upon receipt of the proof P. It takes
  as input the public key pk, the challenge chal, and the proof P returned from
  the server, and outputs TRUE if the integrity of the file is verified as correct,
  or FALSE otherwise.

Verification Protocol is shown as below,

1. Generate a random
   set *{(i, vi)} i∈I ;*

{(i, vi)} i∈I
Challenge request chal →

2. Compute $\mu = \Sigma i\ vimi;$

3. Compute $\sigma = \Sigma i\ \sigma ivi$

← *Integrity proof P*
{μ,σ,{H(mi),Ωi
igsk(H(R))}{i∈I
,sigsk(H(R))}

4. Compute *R using {H(mi),Ωi}i∈I ;*
5. Verify *sigsk(H(R))* and output *FALSE if fail;*
6. Verify *{mi}i∈I .*

IV RESULTS

"Public Verifiability of stored data in a cloud using Disassembly and assembly of data segments" is implemented using MD5. MD5 is used for encryption of file fragments and finding hash values.

The system offers enhanced security in following way:

A. For public verification, MHT(Meakly Hash Tree) is used by applying random key generation mechanism to send and receive data. For that we are considering three scenarios, LNT (Low Network Traffic), ANT(Average Network Traffic), HNT(High Network Traffic).

TABLE I
Average fragmentation & Decryption Time

		Avg. Fragmentation time (microseconds)	Avg. Decryption time (microseconds)
LNT	**F1**	0.001	0.001
	F2	50	50
	F3	500	500
	F4	500	500
	F5	0.001	0.001
ANT	**F1**	20	20
	F2	86	86
	F3	782	782
	F4	780	780
	F5	60	60
HNT	**F1**	23	23
	F2	92	92
	F3	790	790
	F4	770	770
	F5	40	40

TABLE II
File & Fragmentation size

	File size	**Fragment size**
F1	50KB	5KB
F2	50MB	5MB
F3	500MB	50MB
F4	5GB	500MB
F5	50GB	5GB

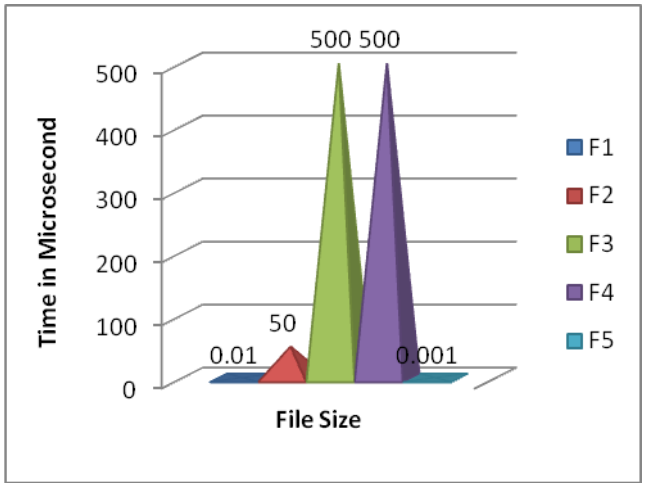


Fig.7: File size and fragment size in Low Network Traffic

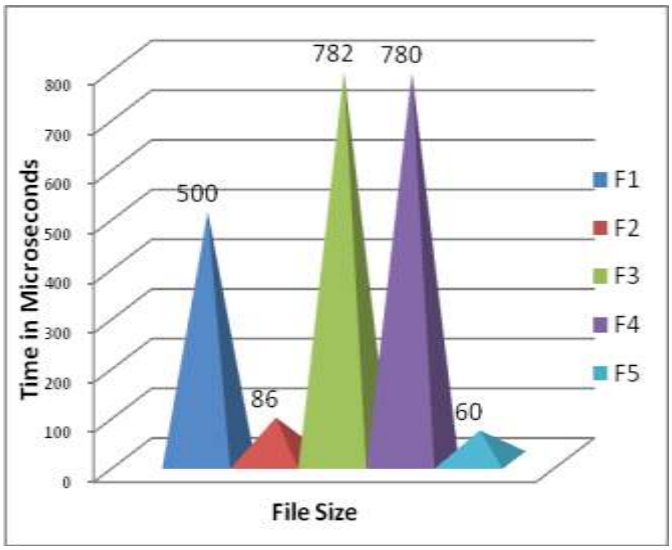


Fig 8: File size and fragment size in Average Network Traffic

ISBN: 978-81-927230-0-6

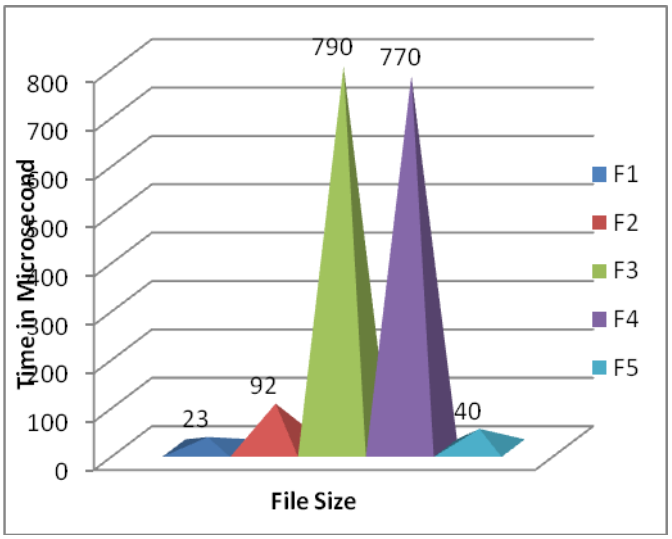


Fig.9 : File size and fragment size in High Network Traffic

B. One secret key is randomly generated for authentication process is MHT and given to the user. If any data loss occurs, this data will be recovered from the various secondary servers.

TABLE III
Average Recovery Time

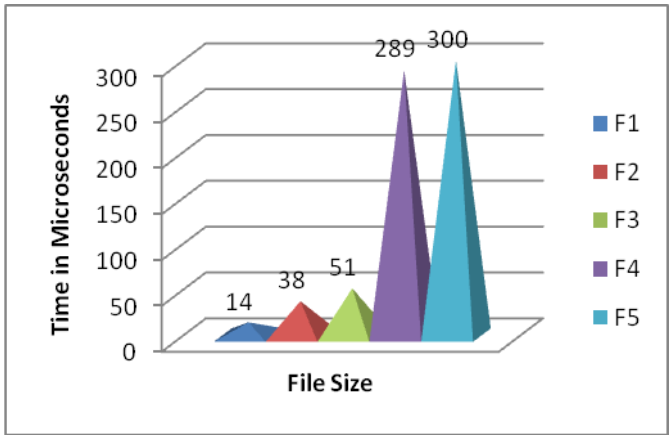


Fig.10.: Fragment loss from Server 1

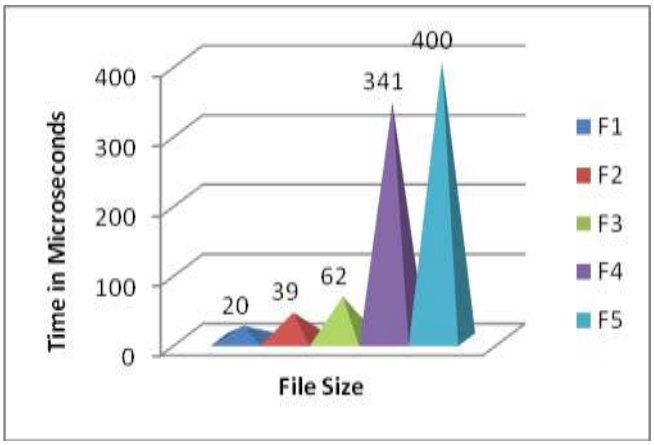
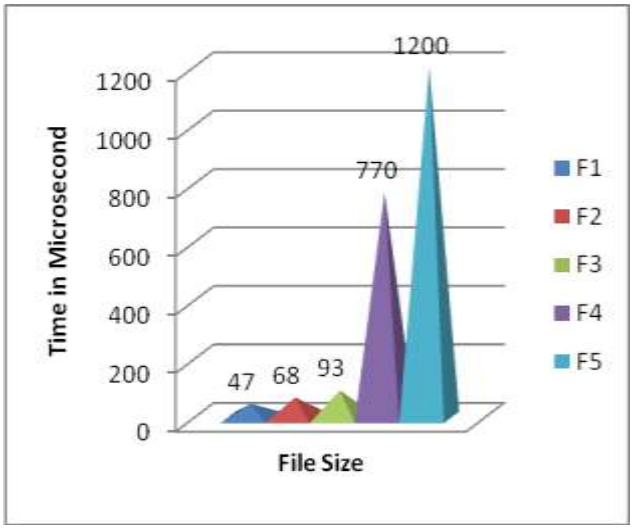


Fig. 11: Fragment loss from Server 1 and Server 2



	Average Recovery Time (Microseconds)				
	F1	**F2**	**F3**	**F4**	**F5**
Fragment loss from Server 1	14	38	51	289	300
Fragment loss from Server 1 and Server 2	20	39	62	341	400
Fragment loss from Server 1,2 &3	47	68	93	770	1200

Fig 12: Fragment loss from Server 1, Server 2 and Server 3

C. If the fragment has been lost then it can be recovered from various secondary servers. Graph 7 is plotted for showing data recovery.

TABLE IV
Fragment Recovery

Product Code	Fragment Recovery (%)
Main Server	100.00
Server1	95.00
Server 2	88.00
Server 3	78.00
Average	90.25

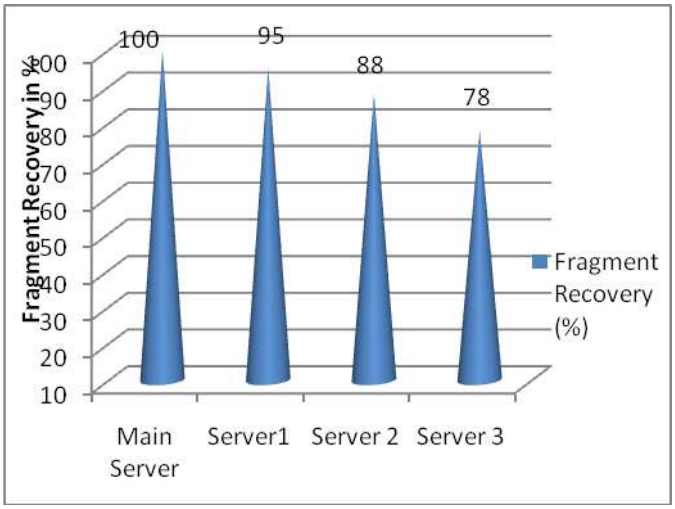


Fig.13: Fragment loss from Server1, 2 and 3

V CONCLUSION

1. The implemented system will check public verifiability of stored data on cloud and if there is any data loss it will be recovered.
2. It is identified that as the loss of number of fragments from different server increase the possibility of fragment recovery decrease.

REFERENCES

[1] Cong Wang, Qian Wang, Kui Ren and Wenjing Lou "Ensuring Data Storage Security in Cloud Computing" ,IEEE 2009

[2] Kun-Yi Cheng and Chun-Hsin Wu,"Peeraid: A Resilent Path-aware Stroage System for Open Clouds" 2009 IEEE.

[3] Wassim Itani, Ayman Kayssi and Ali Chehab, "Privacy-Aware Data Storage and Processing in Cloud Computing Architectres", 2009 8th IEEE International Conference on Dependable Automic and Secure Computing

[4] Hovav Shacham and Brent Waters, "Compacts Proofs of Retrievability for Large Files, Proc. of Asiacrypt 2008, Springer-Verlag, 2008.

[5] Abhishek Verma, Shivram Venkatraman,Matthew Caesar and Roy Campbell," Efficient Metadata Management for cloud computing applications"

[6] Wassim Itani, Ayman Kayssi, Ali Chehab, "Privacy as a Service: Privacy- Aware data Storage and processing in Cloud Computing Architectures", IEEE international conference on Dependable, \ Automatic and secure computing, 2009.