ROBUST SECURE DATA AUDITING IN CLOUD COMPUTING

K. Nanthini*

T. Saravanan**

Abstract

Cloud Storage allows users to store their data and use the cloud applications without the need of local hardware and software resources. Cloud Storage service possess many security risks against storage exactness. This paper presents a flexible distributed storage integrity auditing mechanism to achieve fast localization of data error with low complexity. The proposed design allows users to audit their data and it achieves dynamic data support to ensure the correctness and availability of user's data in cloud. i.e., it efficiently supports block modification, deletion, append.

Index Terms – Cloud computing, Storage correctness, Data integrity, Dynamic data, Error Localization.

* PG Student Department of Information and Technology, PSN College of Engineering and Technology, Tirunelveli

** Assistant Professor Department of Information and Technology, PSN College of Engineering and Technology, Tirunelveli

I. INTRODUCTION

Cloud computing has evolved through a number of phases which include grid and utility computing, application service provision (ASP), and Software as a Service. But the overarching concept of delivering

Computing resources through a global network is rooted in the sixties. Since the sixties, cloud computing has developed along a number of lines, with Web 2.0 being the most recent evolution. However, since the internet only started to offer significant bandwidth in the nineties, cloud computing for the masses has been something of a late developer.

One of the first milestones for cloud computing was the arrival of Salesforce.com in 1999, which pioneered the concept of delivering enterprise applications via a simple website. The services firm paved the way for both specialist and mainstream software firms to deliver applications over the internet.

The next development was Amazon Web Services in 2002, which provided a suite of cloud-based services including storage, computation and even human intelligence through the Amazon Mechanical Turk.

Then in 2006, Amazon launched its Elastic Compute cloud (EC2) as a commercial web service that allows small companies and individuals to rent computers on which to run their own computer applications.

Another big milestone came in 2009, as Web 2.0 hit its stride, and Google and others started to offer browser-based enterprise applications, though services such as Google Apps.

Cloud storage is one of the possible services that can be provided to individuals and organizations through the cloud computing model. As the user relinquishes the control over their data to the cloud service provider, problems concerning data integrity and data availability arises.

The rest of the paper deals with the following sections: section II discuss about the literature survey, section III discuss about the proposed design, and section IV describes the conclusion of the paper.

II. LITERATURE SURVEY

In cloud storage systems, the server that stores the client's data is not necessarily trusted. Therefore, users would like to check if their data has been tampered with or deleted. However, outsourcing the storage of very large files (or whole file systems) to remote servers presents an additional constraint: the client should not download all stored data in order to validate it because this may be prohibitive in terms of bandwidth and time, especially if the client performs this check frequently. Several approaches have been proposed to assist the client verification of file availability and integrity.

They are,

- Provable Data Possession (PDP).
- Proof of Retrievability (POR).
- High Availability and Integrity Layer (HAIL).
- Algebraic Signature based Remote Data Possession Checking (RDPC).

2.1 Provable Data Possession (PDP)

Provable Data Possession (PDP) is a cryptographic technique that allows users to store their data at an untrusted server and have probabilistic guarantees that the server possesses the original data.

Ateniese et al. [2] have formalized a model called provable data possession (PDP). Their scheme utilized public key-based homomorphic tags for auditing the data file. In this model, data (often represented as a file F) is preprocessed by the client, and metadata used for verification purposes is produced. The file is then sent to an untrusted server for storage, and the client may delete the local copy of the file. The client keeps some (possibly secret) information to check server's responses later. The server proves the data has not been tampered with by responding to challenges sent by the client.

Ateniese et al. [3] present a scheme with somewhat limited dynamism. They have developed a dynamic PDP solution called Scalable PDP. This scheme is based entirely on symmetric-key cryptography. Furthermore, each update requires re-creating all the remaining challenges, which is problematic for large files.

Chris Erway et al. [4] provide a definitional framework and efficient constructions for dynamic provable data possession (DPDP), which extends the PDP model to support provable updates on the stored data.

Reza Curtmola et al. [5] extend PDP to apply to multiple replicas so that a client that initially stores t replicas can later receive a guarantee that the storage system can produce t replicas, each of which can be used to reconstruct the original file data. This scheme is called as Multiple-Replica Provable Data Possession (MR-PDP).

The major drawback of PDP is that it checks only the possession of data and it does not recover data in case of a failure (does not support data availability).

2.2 Proof of Retrievability (POR)

A POR is a challenge-response protocol that enables a prover (cloud-storage provider) to demonstrate to a verifier (client) that a file *F* is retrievable, i.e., recoverable without any loss or corruption.

The POR scheme [6] uses special blocks (called sentinels) hidden among other blocks in the data. During the verification phase, the client asks for randomly picked sentinels and checks whether they are intact. If the server modifies or deletes parts of the data, then sentinels would also be affected with a certain probability.

The difference between PDP and POR is that POR checks the possession of data and it can recover data in case of a failure. Usually, a PDP can be transformed to a POR by adding erasure or error correcting codes [9].The major drawback is that it is focussing only on single server scenario.

2.3 High Availability and Integrity Layer (HAIL)

Bowers et al. [9] introduce HAIL (High-Availability and Integrity Layer), a distributed cryptographic system that permits a set of servers to prove to a client that a stored file is intact and retrievable. This scheme is focusing on static or archival data. As a result, their capability of handling dynamic data remains unclear.

2.4 Algebraic Signature based Remote Data Possession Checking (RDPC)

Lanxiang Chen [10] proposes an algebraic signature based RDPC scheme. Algebraic signature can improve efficiency and the running of algebraic signature can achieve tens to

hundreds of megabytes per second. It allows verification without the need for the challenger to compare against the original data. Algebraic signature is a type of hash function that has algebraic properties: taking the signature of the sum of some file blocks gives the same result as taking the sum of the signatures of the corresponding blocks. The drawback of the scheme lies in its probabilistic security.

2.5 Third Party Auditing

Mehul et al. [17] argue that third party auditing is important in creating an online service oriented economy, because it allows customers to evaluate risks, and it increases the efficiency of insurance based risk mitigation.

Wang et al. [18] consider the task of allowing a third party auditor (TPA), on behalf of the cloud client, to verify the integrity of the dynamic data stored in the cloud. To support efficient handling of multiple auditing tasks, the technique of bilinear aggregate signature to extend is explored as well, where TPA can perform multiple auditing tasks simultaneously.

Wang et al. [19] propose to uniquely integrate the homomorphic linear authenticator with random masking technique to achieve privacy-preserving public auditing. The homomorphic linear authenticator and random masking guarantee that the TPA would not learn any knowledge about the data content stored on the cloud server during the efficient auditing process. Also it allows TPA to perform multiple auditing tasks. Yet it does not support distributed server model.

III. PROPOSED DESIGN

In cloud data storage system, users store their data in the cloud and no longer possess the data locally. Thus, the correctness and availability of the data files being stored on the distributed cloud servers must be guaranteed. One of the key issues is to effectively detect any unauthorized data modification and corruption, possibly due to server compromise and/or random Byzantine failures. To address these problems, Raptor Erasure Correcting Code and Hash based Homomorphic authendicator tokens are used.

3.1 Architectural Design

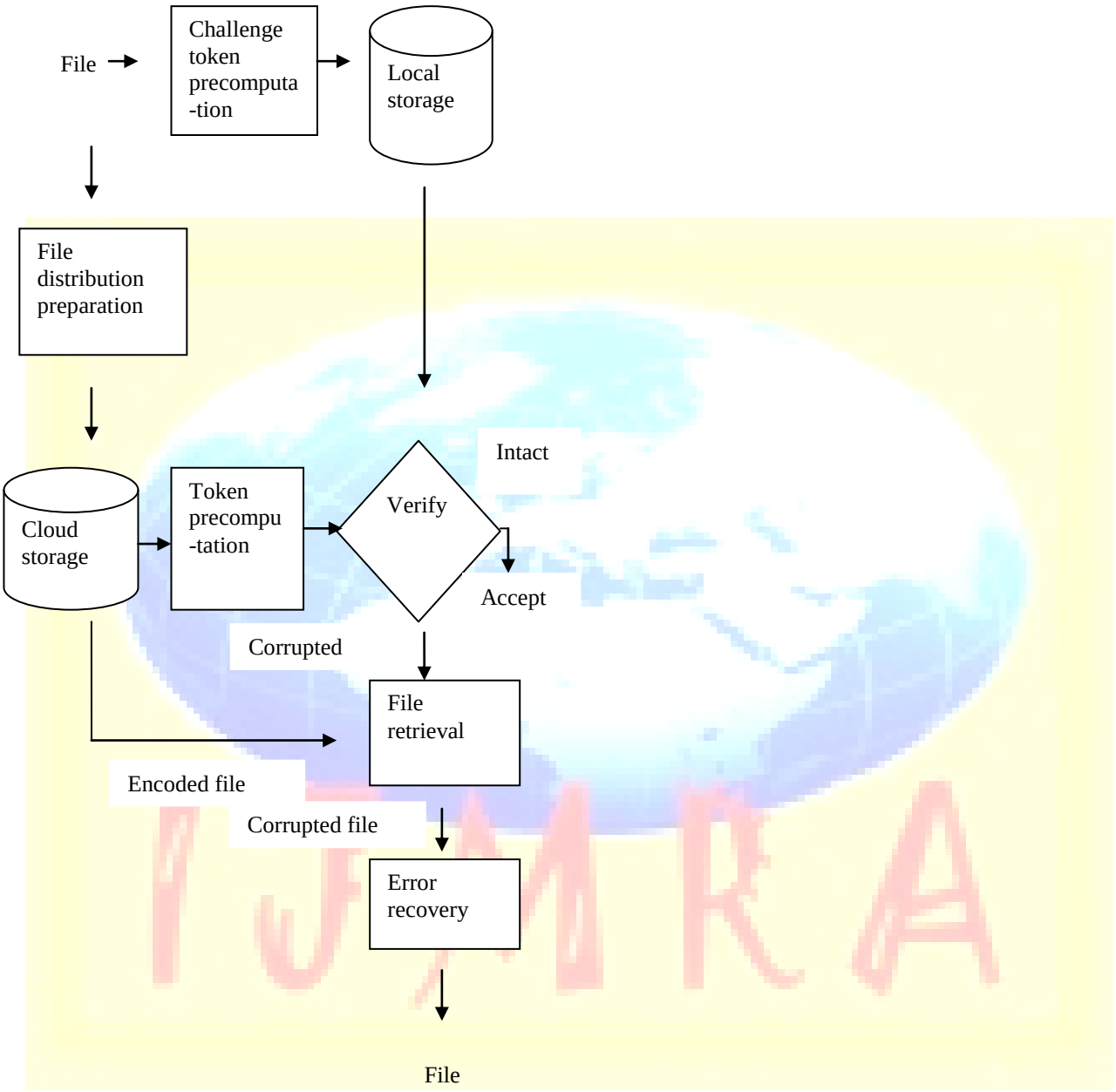

Fig. 3.1: Overall Arichitectural Design

3.2 File Distribution Preparation (Raptor coding)

Raptor erasure-correcting codes are used to tolerate multiple failures in distributed storage systems. In cloud data storage, this technique is used to disperse the data file F redundantly across a set of n = m+ k distributed servers.

Let C be a linear code of block length n and dimension k, and let Ω(x) be a degree distribution. A Raptor code with parameters (k, C, Ω(x)) is an LT-code with distribution Ω(x) on n symbols which are the coordinates of codewords in C. The code C is called the pre-code of the Raptor code. The input symbols of a Raptor code are the k symbols used to construct the codeword in C consisting of n intermediate symbols. The output symbols are the symbols generated by the LT-code from the n intermediate symbols.

3.2.1 Reed Solomon Codes

The pre-code used here is Reed Solomon erasure correcting code. An (m, k) Reed-Solomon erasure-correcting code is used to create k redundancy parity vectors from m data vectors in such a way that the original m data vectors can be reconstructed from any m out of the m + k data and parity vectors. By placing each of the m + k vectors on a different server, the original data file can survive the failure of any k of the m + k servers without any data loss, with a space overhead of k/m.

Let there be m storage devices, D_1, D2 ... Dm. each of which holds l bytes. These are called the "Data Devices". Let there be k more storage devices C_1, C2 ... Ck, each of which also holds l bytes. These are called the "Checksum Devices". The contents of each checksum device will be calculated from the contents of the data devices. The goal is to define the calculation of each C_i such that if any k of D_1, D_2 ... D_m, C_1, C2... C_k fail, then the contents of the failed devices can be reconstructed from the non-failed devices.

3.2.2 LT Codes

In LT Codes, each output symbol is generated by randomly choosing a degree *d* from some suitable degree distribution, choosing *d* distinct input symbols uniformly at random, and taking their sum.

- In encoding phase, each parity bit is XOR of neighboring message or parity bits within its bipartite graph.
- In decoding phase, only one XOR per edge is needed to decode.The decoder repeats the following simplistic decoding operation until all missing message bits are recovered: Given the value of a check bit and all but one of the message bits on which it depends, set the missing message bit to be the XOR of the check bit and its known message bits.

3.3 Challenge Token Precomputation

In order to achieve assurance of data storage correctness and data error localization simultaneously, this scheme entirely relies on the precomputed verification tokens. The main idea is as follows: before file distribution the user precomputes a certain number of short verification tokens on individual vector,

$G^{(j)}$ ($j \in \{1, 2 \ldots n\}$), each token covering a random subset of data blocks. Later, when the user wants to make sure the storage correctness for the data in the cloud, he challenges the cloud servers with a set of randomly generated block indices. Upon receiving challenge, each cloud server computes a short "signature" over the specified blocks and returns them to the user. The values of these signatures should match the corresponding tokens precomputed by the user.

Suppose the user wants to challenge the cloud server's t times to ensure the correctness of data storage. Then, the user must precompute t verification tokens for each G (j) ($j \in \{1, 2... n\}$), using a PRF f (.), a PRP Φ, a challenge key k_{chal}, and a master permutation key K_{PRP}.

```
Procedure
  Choose parameters l, n and function f, Φ;
  Choose the number t of tokens;
  Choose the number r of indices per
  Verification;
  Generate master key KPRP and challenge
   Key kchal;
  For vector G (j), j ←1, n do
    For round i← 1, t do
      Derive αi = fkchal (i) and k(i)prp from KPRP.
      Compute vi(j) = Σrq=1 αiq * G(j) [Φki prp (q)]
    End for
  End for
  Store all the vi's locally.
End procedure
```

Fig. 3.2 Algorithm for Challenge token precomputation

3.4 Correctness Verification and Error Localization

Error localization is a key prerequisite for eliminating errors in storage systems. It is also of critical importance to identify potential threats from external attacks.

However, many previous schemes do not explicitly consider the problem of data error localization, thus only providing binary results for the storage verification. This scheme outperforms those by integrating the correctness verification and error localization (misbehaving server identification) in challenge-response protocol: the response values from servers for each challenge not only determine the correctness of the distributed storage, but also contain information to locate potential data error(s).

```
procedure CHALLENGE(i)
  Recompute α_i = f_kchal(i) and k^(i)_prp from K_PRP ;
  Send { α_i, k^(i)_prp} to all cloud servers;
  Receive from servers:
   { R_i^(j) = Σ^r_q=1 α_i^q * G^(j)[Φk^i_prp (q)]|1<j<n}
   for (j← m + 1, n) do
     R (j) ← R (j) - Σ^r_q=1 fk_j (S_Iq,j ). α_i^q ,Iq = Φk^(i)_prp(q)
   end for
   If ((R_i^(1)...R_i^(m)). P == (R_i^(m+1),...R_i^(n))) then
     Accept and ready for the next challenge.
   else
     for (j ← 1, n) do
       if (R_i^(j) != v_i^(j)) then
         Return server j is misbehaving.
       end if
     end for
   end if
end procedure
```

Fig 3.3 Algorithm for correctness verification and error localization

3.5 File Retrieval and Error Recovery

Whenever the data corruption is detected, the comparison of precomputed tokens and received response values can guarantee the identification of misbehaving server(s). Therefore, the user can always ask servers to send back blocks of the r rows specified in the challenge and regenerate the correct blocks by erasure correction. The newly recovered blocks can then be redistributed to the misbehaving servers to maintain the correctness of storage.

```
Procedure
% Assume the block corruptions have been
detected among the specified r rows;
    Download r rows of blocks from servers;
    Treat s servers as erasures and recover the
    Blocks.
    Resend the recovered blocks to corresponding
    Servers.
End procedure
```

Fig 3.4 Algorithm for File Retrieval and Error Recovery

IV. CONCLUSION

A flexible distributed storage integrity auditing mechanism, utilizing the Raptor erasure-coded data and homomorphic token has proposed to ensure data integrity and availability in cloud storage.

Using Raptor codes,as many encoding symbols as needed can be produced and is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols. Hence it surpasses the existing erasure correcting codes like Reed-Solomon codes.

By utilizing the homomorphic token with distributed verification of erasure coded data, this scheme achieves the integration of storage correctness insurance and data error localization, i.e., whenever data corruption has been detected during the storage correctness verification across the distributed servers, this scheme can almost guarantee the simultaneous identification of the misbehaving server(s).

REFERENCES

[1] http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[2] Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D.Song, "Provable Data Possession at Untrusted Stores," Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), pp. 598-609, Oct. 2007.

[3] G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," Proc. Fourth Int'l Conf. Security and Privacy in Comm. Netowrks (SecureComm '08), pp. 1-10,2008.

[4] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," Proc. 16th ACM Conf. Computer and Comm. Security (CCS '09), pp. 213-222, 2009.

[5] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession," Proc. IEEE 28th Int'l Conf. Distributed Computing Systems (ICDCS '08), pp. 411-420, 2008.

[6] A. Juels and B.S. Kaliski Jr., "PORs: Proofs of Retrievability for Large Files," Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), pp. 584-597, Oct. 2007.

[7] K.D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," Proc. ACM Workshop Cloud Computing Security (CCSW'09), pp. 43-54, 2009.

[8] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of Retrievability via Hardness Amplification," Proc. Sixth Theory of Cryptography Conf. (TCC '09), Mar. 2009.

[9] K.D. Bowers, A. Juels, and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," Proc. ACM Conf. Computer and Comm. Security (CCS '09), pp. 187-198, 2009.

[10]Lanxiang Chen, "Using algebraic signatures to check data possession in cloud storage", Future Generation Computer Systems, 2012.

[11] Computing Curricula 2005: The Overview Report (pdf), The Joint Task Force for Computing Curricula 2005.

[12] Andrews, Gregory R, Foundations of Multithreaded, Parallel, and Distributed Programming, Addison–Wesley, 2000.

[13] Torry Harris, Cloud Computing – An Overview, 2007.

[14] Klaus Julisch and Michael Hall, "Security and Control in the Cloud", Information Security Journal: A Global Perspective (299-309), 2010.

[15] George Demarest, Rex Wang, Oracle Cloud Computing, Oracle Corporation, 2010.

[16] H. Shacham and B. Waters, "Compact Proofs of Retrievability," Proc. 14th Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (Asiacrypt '08), pp. 90- 107, 2008.

[17] M.A. Shah, M. Baker, J.C. Mogul, and R. Swaminathan, "Auditing to Keep Online Storage Services Honest," Proc. 11th USENIX Workshop Hot Topics in Operating Systems (HotOS '07), pp. 1-6, 2007.

[18] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing," IEEE Trans. Parallel and Distributed Systems, vol. 22, no. 5, pp. 847-859, 2011.

[19] C. Wang, S.S.M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy- Preserving Public Auditing for Secure Cloud Storage," IEEE Trans. Computers, preprint, 2012.

[20] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Raj Kumar Buyya," CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms"Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, Vol.3, No.27, Pages 93-109, 2009.