ISSN:0975-9646

A Process for Data Storage security in Cloud Computing

Durgarajesh Rachamsetty, Prof Ramakrishna Rao TK

Department of Information Technology
Aditya Inistitue of Technology and Management
JNTU KAKINADA

Abstract— **Cloud computing can be defined as a new style of computing in which dynamically scalable and often virtualized resources are provided as services over the internet. With cloud computing technology, users use a variety of devices, including PCs, laptops, smart phones, and PDAs to access programs, storage, and application-development platforms over the Internet, via services offered by cloud computing providers. Advantages of cloud computing technology include cost savings, high availability, and easy scalability. Cloud computing moves the application software and data bases to the large data centres, where administration of the data and services may not be fully trustworthy. This unique aspect, however, poses many new security challenges which have not been well understood. In this article, we focus on cloud storage security, which has always been important feature of quality of service. To ensure the correctness of users data in the cloud, we suggest an effective and flexible technique for managing the data storage technology in the secure way. By utilizing the PMAR homorphic encryption algorithm with dispersed authentication of erasure coded data , our scheme achieves the incorporation of storage correctness insurance and data error localization. i.e, the identification of misbehaving server.**

***Keywords*— cloud computing, data integrity, storage process.**

I. INTRODUCTION

Several trends are opening up the era of Cloud Computing, which is an Internet-based development and use of computer technology. The ever cheaper and more dominant processors, together with the software as a service (SaaS) computing architecture, are transforming data centres into pools of computing service on a huge scale. The escalating network bandwidth and reliable yet flexible network connections make it even possible that users can now pledge high quality services from data and software that reside exclusively on secluded data centres. Moving data into the cloud offers great expediency to users since they don't have to care about the complexities of direct hardware management. The pioneer of Cloud Computing vendors, Amazon Simple Storage Service (S3) and Amazon Elastic Compute Cloud (EC2) [1] are both well known examples. While these internet-based online services do offer huge amounts of storage space and customizable computing resources, this computing platform shift, however, is eliminating the responsibility of local machines for data maintenance at the same time. As a result, users are at the mercy of their cloud service providers for the accessibility and veracity of their data. Recent downtime of Amazon's S3 is such an example [2]. From the perception of data security, which has always been an important phase of quality of service, cloud computing predictably poses new challenging security threats for number of reasons.

II. PROBLEM STATEMENT

A. *System Model*

A representative network architecture for cloud data storage is illustrated in Figure 1. Three different network entities can be identified as follows:

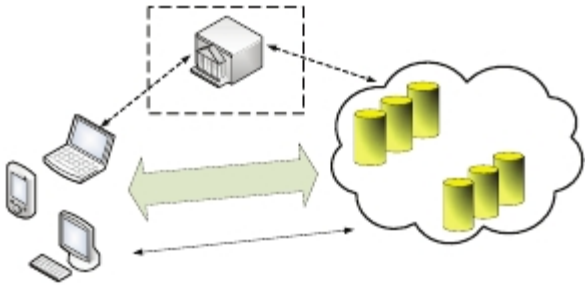

Fig. 1: Cloud data storage architecture

· User: users, who have data to be stored in the cloud and rely on the cloud for data computation, consist of both individual clients and organizations.

· Cloud Service Provider (CSP): a CSP, who has considerable resources and proficiency in building and managing dispersed cloud storage servers, owns and operates live Cloud Computing systems**. Thir**d Party Auditor (TPA): an not obligatory TPA, who has proficiency and capabilities that users may not have, is trusted to review and expose risk of cloud storage services on behalf of the users upon demand.

In cloud data storage, a user stores his data through a CSP into a set of cloud servers, which are running in a synchronized, cooperated and disseminated approach. Data redundancy can be engaged with practice of erasure-correcting code to auxiliary tolerate faults or server crash as user's data grows in size and significance. Thereafter, for application purposes, the user interacts with the cloud servers via CSP to access or recover his data. In some cases, the user may need to execute block level operations on his data. The most common forms of these operations we are considering are block update, delete, insert and append. As users no longer acquire their data in the vicinity, it is of significant importance to guarantee users that their data are being appropriately stored and maintained. That is, users should be outfitted with defines means so that they

can make incessant precision assurance of their stored data even without the subsistence of local copies. In case that users do not inevitably have the time, viability or resources to scrutinize their data, they can entrust the tasks to an optional trusted TPA of their relevant choices. In our model, we assume that the point-to-point communication channels between each cloud server and the user is legitimate and consistent, which can be achieved in practice with little overhead. Note that we don't address the issue of data seclusion in this paper, as in Cloud Computing, data privacy is orthogonal to the dilemma we study here.

B. Design Goals

To ensure the security and fidelity for cloud data storage under the aforesaid antagonist model, we aim to design efficient mechanisms for dynamic data verification and operation and achieve the following goals: (1) Storage correctness: to ensure users that their data are indeed stored appropriately and kept unharmed all the time in the cloud. (2) Fast localization of data error: to effectively locate the faulty server when data corruption has been erected. (3) Dynamic data support: to maintain the same level of storage correctness assurance even if users modify, delete or append their data files in the cloud. (4) Dependability: to enhance data availability against Intricate failures, malevolent data modification and server colluding attacks, i.e. minimizing the effect brought by data errors or server failures. (5) Lightweight: to enable users to perform storage correctness checks with minimum overhead.

III. ENSURING CLOUD DATA STORAGE

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. Besides, in the distributed case when such inconsistencies are successfully detected, to find which server the data error lies in is also of great significance, since it can be the first step to fast recover the storage errors. To address these problems, our main scheme for ensuring cloud data storage is presented in this section. The first part of the section Then, the homomorphic token is introduced. IT defines The common theme is that a homomorphism is a function between two algebraic objects that respects the algebraic structure. The token computation function we are considering belongs to a family of universal hash function [11], chosen to preserve the homomorphic properties, which can be perfectly integrated with the verification of erasure-coded data [8] [12]. Subsequently, it is also shown how to derive a challenge response protocol for verifying the storage accuracy as well as identifying misbehaving servers. Finally, the method for file reclamation and error recuperation based on erasure-correcting code is outlined.

Algorithm 1 : TOKEN PRE-COMPUTATION

1: procedure

2: Choose parameters l, n and function f, Ø;

3: Choose the number t of tokens;

4: Choose the number r of indices per verification;

5: Generate master key Kprp and challenge kchal;

6: for vector G(j), j ← 1, n do

7: for round i← 1, t do

8: Derive $\alpha_i = fk_{chal}(i)$ and $k_{prp}(i)$ from K_{prp} .

9: Compute $v_i^{(j)} = \sum_{q=1}^{r} \alpha_i^q * G^{(j)} [\varphi k^{(i)}_{prp}(q)]$

10: end for

11: end for

12: Store all the v_is locally.

13: end procedure

Before file distribution the user pre-computes a certain number of short verification tokens on individual vector $G^{(j)}$ (j € {1,2,3 ….. , 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 block indices. Upon receiving challenge, each cloud server computes a short "signature" over the particular blocks and returns them to the user. The values of these signatures should match the corresponding tokens pre-computed by the user. Meanwhile, as all servers operate over the same subset of the indices, the requested response values for integrity check must also be a valid codeword determined by secret matrix P. Suppose the user wants to challenge the cloud servers t times to ensure the correctness of data storage. Then, he must pre-compute t verification tokens for each G(j) (j □ {1, . . . , n}), using a PRF f(·), a PRP f(·), a challenge key kchal and a master Permutation key KPRP . We are applying map reduce algorithm to homomorphic algorithm. A private-key MR-parallel F-homomorphic encryption scheme is a tuple of polynomial-time algorithms PHE = (Gen; Enc; Eval; Dec), where (Gen; Enc; Dec) are as in a private- key encryption scheme and Eval = (Parse; map; Part; Red; Merge) is a MapReduce algorithm. More precisely we have:

Algorithm 2: applying map reduce process to homomorphic.

1. Procedure
2. K Gen(1k): is a probabilistic algorithm that takes as input a security parameter k and that returns a key K.
3. C← Enc(K; x): is a probabilistic algorithm that takes as input a key K and an input x from some message space X, and that returns a ciphertext c. We sometimes write this as c← $E_{nck}(x)$.

4. $(l_i, vi)_i$ ←Parse(f; c): is a deterministic algorithm that takes as input a function f € F and a ciphertext c, and that returns a sequence of input pairs.
5. (i ; j)j← Map(l, v): is a (possibly probabilistic) algorithm that takes an input pair (l, v) and that returns a sequence of intermediate pairs.
6. H←Part(I, j): is a (possibly probabilistic) algorithm that takes as input an intermediate pair (I,j) and that returns a value h in some space H.
7. (I, z)←Red(I, P): is a (possibly probabilistic) algorithm that takes a label _ and a partition P of intermediate values and returns an output pair (I, z).
8. C^1← Merge((i_t, z_t)): is a deterministic algorithm that takes as input a set of output pairs and returns a ciphertext c^1.
9. Y←Dec(K; c^1): is a deterministic algorithm that takes a key K and a ciphertext c0 and that returns an output y. We sometimes write this as y DecK(c^1).
10. End procedure.

for all k € N, for all f € Fk, for all K output by Gen(1k), for all x € X, for all c output by EncK(x), DecK (Eval(f, c))= f(x).

Algorithm 3: *File Retrieval and Error Recovery*

1. **Procedure**
   % Assume the block corruptions have been detected among
   % the specified r rows;
   % Assume s ≤ k servers have been identified Misbehaving.
2. Download r rows of blocks from servers;
3. Treat s servers as erasures and recover the Blocks.
4. Resend the recovered blocks to corresponding servers.
5. End procedure.

Since our layout of file matrix is systematic, the user can reconstruct the original file by downloading the data vectors from the first m servers, assuming that they return the correct response values. Notice that our verification scheme is based on random spot-checking, so the storage correctness assurance is a probabilistic one. However, by choosing system parameters (e.g., r, l, t) appropriately and conducting enough times of verification, we can guarantee the successful file retrieval with high probability. On the other hand, whenever the data corruption is detected, the comparison of pre-computed tokens and received response values can guarantee the identification of misbehaving server(s), again with high probability, which will be discussed shortly. Therefore, the user can always ask servers to send back blocks of the r rows specified in the challenge and reinforce the correct blocks by erasure correction, shown in Algorithm 3, as long as there are at most k misbehaving servers are acknowledged. The newly recovered blocks can then be re dispersed to the misbehaving servers to maintain the correctness of storage.

IV. PROVIDING DYNAMIC DATA OPERATION SUPPORT

So far, we assumed that **F** represents static or archived data. This model may fit some application scenarios, such as libraries and scientific datasets. However, 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 block-level 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. In this section, we will show how our scheme can explicitly and efficiently handle dynamic data operations for cloud data storage.

V. RELATED WORK

Juels et al. [3] described a formal "proof of retrievability" (POR) model for ensuring the remote data integrity. Their scheme combines spot-checking and error-correcting code to ensure both possession and retrievability of files on archive service systems. Shacham et al. [4] built on this model and constructed a random linear function based homomorphic authenticator which enables unlimited number of queries and requires less communication overhead. Bowers et al. [5] roposed an improved framework for POR protocols that generalizes both Juels and Shacham's work. Later in their subsequent work, Bowers et al. [10] extended POR model to dispersed systems. However, all these schemes are focusing on static data. The effectiveness of their schemes rests primarily on the preprocessing steps that the user conducts before outsourcing the data file F. Any change to the contents of F, even few bits, must propagate through the error-correcting code, thus introducing considerable computation and communication complexity. Ateniese et al. [6] defined the "provable data possession" (PDP) model for ensuring possession of file on untrusted storages. Their scheme utilized public key based homomorphic tags for auditing the data file, thus providing public verifiability. However, their scheme requires sufficient computation overhead that can be expensive for an entire file. In their subsequent work, Ateniese et al. [7] described a PDP scheme that uses only symmetric key cryptography. This routine has lower-overhead than their previous scheme and allows for block updates, deletions and appends to the stored file, which has also been supported in our work. However, their scheme focuses on single server scenario and does not address small

data corruptions, leaving both the dispersed scenario and data error recovery issue unexplored. Curtmola et al. [15] aimed to ensure data possession of multiple replicas across the dispersed storage system. They extended the PDP scheme to cover multiple replicas without encoding each replica separately, providing guarantee that multiple copies of data are actually maintained.

Ateniese et al. [7] are the first to consider public auditability in their defined "provable data possession" (PDP) model for ensuring possession of data files on untrusted storages. Their scheme utilizes the RSA-based homomorphic authenticators for auditing outsourced data and suggests randomly sampling a few blocks of the file. However, the public auditability in their scheme demands the linear combination of sampled blocks exposed to external auditor.When used directly, their protocol is not provably privacy preserving, and thus may leak user data information to the auditor. Ateniese et al. [25] propose a partially dynamic version of the prior PDP scheme that uses only symmetric key cryptography. However, the system imposes a priori bound on the number of audits and does not support public auditability.

VI. CONCLUSION

In this paper, we investigated the problem of data security in cloud data storage, which is fundamentally a dispersed storage system. To ensure the correctness of users' data in cloud data storage, we proposed an effective and flexible dispersed scheme with explicit dynamic data sustain, including block update, delete, and append. We rely on erasure-correcting code in the file distribution preparation to provide redundancy parity vectors and guarantee the data dependability. By utilizing the parallel Homomorphic token with dispersed verification of erasure coded data, our 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 dispersed servers, we can almost guarantee the simultaneous identification of the misbehaving server(s). MapReduce allows for distributed processing of the map and reduction operations. Provided each mapping operation is independent of the others, all maps can be performed in parallel – though in practice it is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase - provided all outputs of the map operation that share the same key are presented to the same reducer at the same time. While this process can often appear inefficient compared to algorithms that are more sequential, MapReduce can be applied to significantly larger datasets than "commodity" servers can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours. The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available. Through detailed security and performance analysis, we show that our scheme is highly efficient and resilient to Intricate failure, malicious data modification attack, and even server colluding attacks. We believe that data storage security in Cloud Computing, an area full of challenges and of overriding importance, is still in its formative years now, and many research problems are yet to be identified. We foresee several possible directions for future research on this area. The most promising one we believe is a model in which public verifiability is enforced. Public. verifiability, supported in [6] [4] [17], allows TPA to audit the cloud data storage without challenging users' time, possibility or resources. An interesting question in this model is if we can construct a scheme to achieve both public verifiability and storage correctness assurance of vibrant data. Besides, along with our research on dynamic cloud data storage, we also plan to inspect the quandary of fine-grained data error localization.

REFERENCES

[1] Amazon.com, "Amazon Web Services (AWS)," Online at http://aws.amazon.com, 2008.

[2] N. Gohring, "Amazon's S3 down for several hours,"Onlineat http://www.pcworld.com/businesscenter/article/142549/amazons s3 down for several hours.html, 2008.

[3] A. Juels and J. Burton S. Kaliski, "PORs: Proofs of Retrievability for Large Files," *Proc. of CCS '07*, pp. 584–597, 2007.

[4] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *Proc. of Asiacrypt '08*, Dec. 2008.

[5] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theoryand Implementation," Cryptology ePrint Archive, Report 2008/175,
2008, http://eprint.iacr.org/.

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," *Proc. ofCCS '07*, pp. 598–609, 2007.

[7] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," *Proc. of SecureComm '08*, pp. 1– 10, 2008.

[8] T. S. J. Schwarz and E. L. Miller, "Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage," *Proc. of ICDCS '06*, pp. 12–12, 2006.

[9] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard, "A Cooperative Internet Backup Scheme," *Proc. of the 2003 USENIX Annual Technical Conference (General Track)*, pp. 29–41, 2003.

[10] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," Cryptology ePrint Archive, Report
2008/489, 2008, http://eprint.iacr.org/.

[11] L. Carter and M. Wegman, "Universal Hash Functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[12] J. Hendricks, G. Ganger, and M. Reiter, "Verifying Distributed Erasurecoded Data," *Proc. 26th ACM Symposium on Principles of Distributed*
Computing, pp. 139–146, 2007.

[13] J. S. Plank and Y. Ding, "Note: Correction to the 1997 Tutorial on Reed-Solomon Coding," University of Tennessee, Tech. Rep. CS-03-
504, 2003.

[14] Q. Wang, K. Ren, W. Lou, and Y. Zhang, "Dependable and SecureSensor Data Storage with Dynamic Integrity Assurance," *Proc. of IEEE*
INFOCOM, 2009

[15].http://eprint.iacr.org/2011/596.pdf