Enhancement of Secure and Dependable Storage Services and Security using Third Party Auditing

T Mahesh [α] & A. Raghavendra Rao [σ]

Abstract - **Cloud Computing is the long dreamed vision of computing as a utility, Cloud storage facilitates users to store the data remotely and enjoy the on-demand high quality cloud servers without the burden of local software and hardware systems. By outsourcing the data, users can be comforted from the burden of local data storage and maintenance. By having these many benefits, such a service is also abandon users' physical control of their outsourced data, which unavoidably posture new security risks towards the accuracy of the data in cloud. In order to address this new problem and further achieve a secure and dependable cloud storage service, we propose in this paper a flexible distributed storage integrity auditing mechanism, utilizing 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. 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.**

I. Introduction

In this paper, we propose an effective and flexible distributed scheme with explicit dynamic data support to ensure the correctness of users' data in the cloud. We rely on erasure correcting code in the file distribution preparation to provide redundancies and guarantee the data constancy. This construction drastically reduces the communication and storage overhead as compared to the traditional replication-based file distribution techniques. By utilizing the homomorphic token with distributed verification of erasure-coded data, our scheme achieves the storage correctness insurance as well as data error localization: whenever data corruption has been detected during the storage correctness verification, our scheme can almost guarantee the simultaneous localization of data errors, i.e., the identification of the misbehaving server(s). Our work is among the first few ones in this field to consider distributed data storage in Cloud Computing. Our contribution can be summarized as the following three aspects:

1. Compared to many of its predecessors, which only provide binary results about the storage state across the distributed servers, the challenge-response protocol in our work further provides the localization of data error.
2. Unlike most prior works for ensuring remote data integrity, the new scheme supports secure and efficient dynamic operations on data blocks, including: update, delete and append.
3. Extensive security and performance analysis shows that the proposed scheme is highly efficient and resilient against Byzantine failure, malicious data modification attack, and even server colluding attacks.

II. Problem Statement

a) The System and Threat Model

We consider a cloud data storage service involving three different entities, as illustrated in Fig. 1: the cloud user (U), who has large amount of data files to be stored in the cloud; 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 third party auditor (TPA), who has expertise and capabilities that cloud users do not have and is trusted to assess the cloud storage service security on behalf of the user upon request. Users rely on the CS for cloud data storage and maintenance. They may also dynamically interact with the CS to access and update their stored data for various application purposes. The users may resort to TPA for ensuring the storage security of their outsourced data, while hoping to keep their data private from TPA. We consider the existence of a semi-trusted CS in the sense that in most of time it behaves properly and does not deviate from the prescribed protocol execution. While providing the cloud data storage based services, for their own benefits the CS might neglect to keep or deliberately delete rarely accessed data files which belong to ordinary cloud users. Moreover, the CS may decide to hide the data corruptions caused by server hacks or Byzantine failures to maintain reputation. We

Authors α σ : M.Tech CSE Dept, ASRA, Hyderabad.
E-mails : mahi06538@gmail.com, raghavamay15@gmail.com

assume the TPA, who is in the business of auditing, is reliable and independent, and thus has no incentive to collude with either the CS or the users during the auditing process. TPA should be able to efficiently audit the cloud data storage without local copy of data and without bringing in additional on-line burden to cloud users. However, any possible leakage of user's outsourced data towards TPA through the auditing protocol should be prohibited. Note that to achieve the audit delegation and authorize CS to respond to TPA's audits, the user can sign a certificate granting audit rights to the TPA's public key, and all audits from the TPA are Authenticated against such a certificate.

b) Design Goals

To enable privacy-preserving public auditing for cloud data storage under the aforementioned model, our protocol design should achieve the following security and performance guarantee: 1) Public audit-ability: to allow TPA to verify the correctness of the cloud data on demand without retrieving a copy of the whole data or introducing additional on-line burden to the cloud users; 2) Storage correctness: to ensure that there exists no cheating cloud server that can pass the audit from TPA without indeed storing users' data intact; 3) Privacy-preserving: to ensure that there exists no way for TPA to derive users' data content from the information collected during the auditing process; 4) Batch auditing: to enable TPA with secure and efficient auditing capability to cope with multiple auditing delegations from possibly large number of different users simultaneously; 5) Lightweight: to allow TPA to perform auditing with minimum communication and computation overhead.

III. System Development

a) System Model

User: users, who have data to be stored in the cloud and rely on the cloud for data computation, consist of both individual consumers and organizations. Cloud Service Provider (CSP): a CSP, who has significant resources and expertise in building and managing distributed cloud storage servers, owns and operates live Cloud Computing systems. Third Party Auditor (TPA): an optional TPA, who has expertise and capabilities that users may not have, is trusted to assess and expose risk of cloud storage services on behalf of the users upon request.

b) File Retrieval and Error Recovery

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. 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).

c) Third Party Auditing

As discussed in our architecture, 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. However, as pointed out by the recent work, to securely introduce an effective TPA, the auditing process should bring in no new vulnerabilities towards user data privacy. Namely, TPA should not learn user's data content through the delegated data auditing.

d) Cloud Operations

i. *Update Operation*

In cloud data storage, sometimes the user may need to modify some data block(s) stored in the cloud, we refer this operation as data update. In other words, for all the unused tokens, the user needs to exclude every occurrence of the old data block and replace it with the new one.

ii. *Delete Operation*

Sometimes, after being stored in the cloud, certain data blocks may need to be deleted. The delete operation we are considering is a general one, in which user replaces the data block with zero or some special reserved data symbol. From this point of view, the delete operation is actually a special case of the data update operation, where the original data blocks can be replaced with zeros or some predetermined special blocks.

iii. *Append Operation*

In some cases, the user may want to increase the size of his stored data by adding blocks at the end of the data file, which we refer as data append. We anticipate that the most frequent append operation in cloud data storage is bulk append, in which the user needs to upload a large number of blocks (not a single block) at one time.

iv. *Insert Operation*

An insert operation to the data file refers to an append operation at the desired index position while maintaining the same data block structure for the whole data file, i.e., inserting a block F[j] corresponds to shifting all blocks starting with index $j + 1$ by one slot. Thus, an insert operation may affect many rows in the logical data file matrix F, and a substantial number of computations are required to renumber all the subsequent blocks as well as re-compute the challenge-response tokens. Hence, a direct insert operation is difficult to support.

IV. The Proposed Schemes

In the introduction we motivated the public auditability with achieving economies of scale for cloud computing. This section presents our public auditing scheme for cloud data storage security. We start from the overview of our public auditing system and discuss two straightforward schemes and their demerits. Then we present our main result for privacy-preserving public auditing to achieve the aforementioned design goals. We also show how to extent our main scheme to support batch auditing for TPA upon delegations from multi-users. Finally, we discuss how to adapt our main result to support data dynamics.

a) Definitions and Framework of Public Auditing System

We follow the similar definition of previously proposed schemes in the context of remote data integrity checking and adapt the framework for our privacy-preserving public auditing system. A public auditing scheme consists of four algorithms (KeyGen, SigGen, Gen Proof, Verify Proof). KeyGen is a key generation algorithm that is run by the user to setup the scheme. SigGen is used by the user to generate verification metadata, which may consist of MAC, signatures, or other related information that will be used for auditing. Gen Proof is run by the cloud server to generate a proof of data storage correctness, while Verify Proof is run by the TPA to audit the proof from the cloud server. Our public auditing system can be constructed from the above auditing scheme in two phases, Setup and Audit:

i. *Setup*

The user initializes the public and secret parameters of the system by executing KeyGen, and pre-processes the data file F by using SigGen to generate the verification metadata. The user then stores the data file F at the cloud server, delete its local copy, and publish the verification metadata to TPA for later audit. As part of pre-processing, the user may alter the data file F by expanding it or including additional metadata to be stored at server.

ii. *Audit*

The TPA issues an audit message or challenge to the cloud server to make sure that the cloud server has retained the data file F properly at the time of the audit. The cloud server will derive a response message from a function of the stored data file F by executing Gen Proof. Using the verification metadata, the TPA verifies the response via Verify Proof. Note that in our design, we do not assume any additional property on the data file, and thus regard error-correcting codes as orthogonal to our system. If the user wants to have more error-resiliency, he/she can first redundantly encode the data file and then provide us with the data file that has error-correcting codes integrated.

b) The Basic Schemes

Before giving our main result, we first start with two warm up schemes. The first one does not ensure privacy-preserving guarantee and is not as lightweight as we would like. The second one overcomes the first one, but suffers from other undesirable systematic demerits for public auditing: bounded usage and auditor state fullness, which may pose additional on-line burden to users as will be elaborated shortly. We believe the analysis of these basic schemes will lead us to our main result, which overcomes all these drawbacks.

Basic Scheme I The cloud user pre-computes MACs _i = MACsk(i||mi) of each block mi (i ∈ {1, . . . , n}), sends both the data file F and the MACs {_i}1≤i≤n onto the cloud server, and releases the secret key sk to TPA. During the Audit phase, the TPA requests from the cloud server a number of randomly selected blocks and their corresponding MACs to verify the correctness of the data file. The insight behind this approach is that auditing most of the file is much easier than the whole of it. However, this simple solution suffers from the following severe drawbacks:

1. The audit from TPA demands retrieval of users' data, which should be prohibitive because it violates the privacy-preserving guarantee;
2. Its communication and computation complexity are both linear with respect to the sampled data size, which may result in large communication overhead and time delay, especially.

When the bandwidth available between the TPA and the cloud server is limited. Basic Scheme II To avoid retrieving data from the cloud server, one may improve the above solution as follows: Before data outsourcing, the cloud user chooses s random message authentication code keys {sk_}1≤_≤s, pre-computes s MACs, {MACsk_ (F)}1≤_≤s for the whole data file F, and publishes these verification metadata to TPA. The TPA can each time reveal a secret key sk_ to the cloud server and ask for a fresh keyed MAC for comparison, thus achieving privacy-preserving auditing. However, in this method: 1) the number of times a particular data file can be audited is limited by the number of secret keys that must be a fixed priori. Once all possible secret keys are exhausted, cloud user then has to retrieve data from the server in order to re-compute and re-publish new MACs to TPA. 2) The TPA has to maintain and update state between audits, i.e., keep a track on the possessed MAC keys. Considering the potentially large number of audit delegations from multiple users, maintaining such states for TPA can be difficult and error prone. Note that another common drawback of the above basic schemes is that they can only support the case of static data, and none of them can deal with data dynamics. For the reason of brevity and clarity, our main result will focus on the static data, too. In Section 3.5, we

will show how to adapt our main result to support dynamic data update.

c) The Privacy-Preserving Public Auditing Scheme

To effectively support public audit ability without having to retrieve the data blocks themselves, we resort to the homomorphic authenticator technique. Homomorphic authent-icators are unforgivable verification metadata generated from individual data blocks, which can be securely aggregated in such a way to assure an auditor that a linear combination of data blocks is correctly computed by verifying only the aggregated authenticator. However, the direct adoption of these techniques is not suitable for our purposes, since the linear combination of blocks may potentially reveal user data information, thus violating the privacy-preserving guarantee. Specifically, if enough number of the linear combinations of the same blocks are collected, the TPA can simply derive the user's data content by solving a system of linear equations. Overview To achieve privacy-preserving public auditing, we propose to uniquely integrate the homomorphic authenticator with random mask technique. In our protocol, the linear combination of sampled blocks in the server's response is masked with randomness generated by a pseudo random function (PRF). With random mask, the TPA no longer has all the necessary information to build up a correct group of linear equations and therefore cannot derive the user's data content, no matter how many linear combinations of the same set of file blocks can be collected. Meanwhile, due to the algebraic property of the homomorphic authenticator, the correctness validation of the block-authenticator pairs will not be affected by the randomness generated from a PRF, which will be shown shortly. Note that in our design, we use public key based homomorphic authenticator, specifically, the one in which is based on BLS signature, to equip the auditing protocol with public audit ability. Its flexibility in signature aggregation will further benefit us for the multitask auditing.

V. Multiple Levels Of Security

a) Virtual Private Cloud

Each VPC is a distinct, isolated network within the cloud. At creation time, an IP address range for each VPC is selected by the customer. Network traffic within each VPC is isolated from all other VPCs; therefore, multiple VPCs may use overlapping (even identical) IP address ranges without loss of this isolation. By default, VPCs have no external connectivity. Customers may create and attach an Internet Gateway, VPN Gateway, or both to establish external connectivity, subject to the controls below.

b) API

Calls to create and delete VPCs, change routing, security group, and network ACL parameters, and perform other functions are all signed by the customer's Amazon Secret Access Key, which could be either the AWS Accounts Secret Access Key or the Secret Access key of a user created with AWS IAM. Without access to the customer's Secret Access Key, Amazon VPC API calls cannot be made on the customer's behalf. In addition, API calls can be encrypted with SSL to maintain confidentiality. Amazon recommends always using SSL protected API endpoints. AWS IAM also enables a customer to further control what APIs a newly created user has permissions to call.

c) Subnets

Customers create one or more subnets within each VPC; each instance launched in the VPC is connected to one subnet. Traditional Layer 2 security attacks, including MAC spoofing and ARP spoofing, are blocked.

d) Route Tables and Routes

Each Subnet in a VPC is associated with a routing table, and all network traffic leaving a subnet is processed by the routing table to determine the destination.

e) VPN Gateway

A VPN Gateway enables private connectivity between the VPC and another network. Network traffic within each VPN Gateway is isolated from network traffic within all other VPN Gateways. Customers may establish VPN Connections to the VPN Gateway from gateway devices at the customer premise. Each connection is secured by a preshared key in conjunction with the IP address of the customer gateway device.

f) Internet Gateway

An Internet Gateway may be attached to a VPC to enable direct connectivity to Amazon S3, other AWS services and the Internet. Each instance desiring this access must either have a n Elastic IP associated with it or route traffic through a NAT instance. Additionally, network routes are configured to direct traffic to the Internet Gateway. AWS provides reference NAT AMIs that can be extended by customers to perform network logging, deep packet inspection, application layer filtering.

VI. Input Design

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data

from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded. The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

VII. Output Design

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
2. Select methods for presenting information.
3. Create document, report, or other formats that contain information produced by the system. The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

VIII. Related Work

Juels et al. described a formal "proof of irretrievability" (POR) model for ensuring the remote data integrity. Their scheme combines spot-checking and error correcting code to ensure both possession and irretrievability of files on archive service systems. Shacham et al. built on this model and constructed a random linear function based homomorphic authenticator which enables unlimited number of challenges and requires less communication overhead due to its usage of relatively small size of BLS signature. Bowers et al. proposed an improved framework for POR protocols that generalizes both Juels and Shacham's work. Later in their subsequent work, Bowers et al. extended POR model to distributed 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 and the corresponding random shuffling process, thus introducing significant computation and communication complexity. Recently, Dodis et al. gave theoretical studies on generalized framework for different variants of existing POR work. Ateniese et al. 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. However, the pre-computation of the tags imposes heavy computation overhead that can be expensive for an entire file. In their subsequent work, Ateniese et al. described a PDP scheme that uses only symmetric key based cryptography. This method 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 provide data availability guarantee against server failures, leaving both the distributed scenario and data error recovery issue unexplored. The explicit support of data dynamics has further been studied in the two recent work and. Wang et al. proposed to combine BLS based homomorphic authenticator with Markel Hash Tree to support fully data dynamics, while Erway et al. developed a skip list based

scheme to enable provable data possession with fully dynamics support. The incremental cryptography work done by Bellare et al. also provides a set of cryptographic building blocks such as hash, MAC, and signature functions that may be employed for storage integrity verification while supporting dynamic operations on data. However, this branch of work falls into the traditional data integrity protection mechanism, where local copy of data has to be maintained for the verification. It is not yet clear how the work can be adapted to cloud storage scenario where users no longer have the data at local sites but still need to ensure the storage correctness efficiently in the cloud. In other related work, Curtmola et al. aimed to ensure data possession of multiple replicas across the distributed 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. Lilli bridge et al. presented a P2P backup scheme in which blocks of a data file are dispersed across m + k peers using an (m, k)- erasure code. Peers can request random blocks from their backup peers and verify the integrity using separate keyed cryptographic hashes attached on each block. Their scheme can detect data loss from free-riding peers, but does not ensure all data is unchanged. Filho et al. proposed to verify data integrity using RSA-based hash to demonstrate uncheatable data possession in peerto-peer file sharing networks. However, their proposal requires exponentiation over the entire data file, which is clearly impractical for the server whenever the file is large. Shah et al. proposed allowing a TPA to keep online storage honest by first encrypting the data then sending a number of pre-computed symmetric keyed hashes over the encrypted data to the auditor.

However, their scheme only works for encrypted files, and auditors must maintain long-term state. Schwarz et al. proposed to ensure static file integrity across multiple distributed servers, using erasure-coding and block level file integrity checks. We adopted some ideas of their distributed storage verification protocol. However, our scheme further support data dynamics and explicitly studies the problem of misbehaving server identification, while theirs did not. Very recently, Wang et al. gave a study on many existing solutions on remote data integrity checking, and discussed their pros and cons under different design scenarios of secure cloud storage services. Portions of the work presented in this paper have previously appeared as an extended abstract in. We have revised the article a lot and add more technical details as compared to. The primary improvements are as follows: Firstly, we provide the protocol extension for privacy-preserving third-party auditing, and discuss the application scenarios for cloud storage service. Secondly, we add correctness analysis of proposed storage verification design. Thirdly, we completely redo all the experiments in our performance evaluation part, which achieves significantly improved result as compared to. We also add detailed discussion on the strength of our bounded usage for protocol verifications and its comparison with state-of-the-art.

IX. Conclusion

In this paper, we examine the complications of the data stored in the cloud storage and security of the data in cloud data storage system, To achieve the affirmation of cloud data goodness and availability and accomplish the quality of dependable cloud storage service for users, we propose an adequate and formative allocated scheme with explicit dynamic data support, including block update, delete, append and insert. We rely on erasure-correcting code in the file distribution preparation to provide redundancy parity vectors and guarantee the data constancy. By utilizing the homomorphic token with allocated 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 allocated servers, we can almost guarantee the simultaneous identification of the misbehaving server(s). Considering the time, computation resources, and even the related online burden of users, we also provide the extension of the proposed main scheme to support third-party auditing, where users can safely delegate the integrity checking tasks to third-party auditors and be worry-free to use the cloud storage services. Through detailed security and extensive experiment results, we show that our scheme is highly efficient and resilient to Byzantine failure, malicious data modification attack, and even server colluding attacks.

References Références Referencias

1. C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in Proc. of IWQoS'09, July 2009, pp. 1–9.
2. Amazon.com, "Amazon web services (aws)," Online at http://aws.amazon.com/, 2009.
3. Sun Microsystems, Inc., "Building customer trust in cloud computing with transparent security," Online at https://www.sun.com/offers/details/suntransparency.xml, November 2009.
4. M. Arrington, "Gmail disaster: Reports of mass email deletions," Online at http://www.techcrunch.com/2006/12/28/ Gmail-disaster reports-of-mass-email-deletions/,December 2006.
5. J.Kincaid, "Media ax/The Linkup Closes Its Doors," Online at http://www.techcrunch.com/2008/07/10/mdiamaxthelinkup-closes-its-doors/, July 2008.
6. Amazon.com, "Amazon s3 availability event: July 20, 2008," Online at http://status.aws.amazon.com/s3-20080720.html, July 2008.

7. S. Wilson, "Appengine outage," Online at http://www.cio-weblog.com/50226711/appengineoutage.php, June 2008.
8. B. Krebs, "Payment Processor Breach May Be Largest Ever," Online http://voices.washingtonpost.com/securityfix/2009/01/payment processor breach may b.html, Jan. 2009.
9. 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.
10. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in Proc. of CCS'07, Alexandria, VA, October 2007, pp. 598–609.

This page is intentionally left blank