Volume 5, Issue 5, May 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Cloud Data Protection for Secure and Dependable Storage Services

G. Fayaz Hussain, S.Vinod Kumar
Faculty of Engg., CSE Department, Ravindra College of Engg. for Women,
Kurnool, Andhra, Pradesh, India

Abstract: Cloud Computing has become a scalable services consumption and delivery platform in the field of Services Computing. The technical foundations of Cloud Computing include Service-Oriented Architecture (SOA) and Virtualizations of hardware and software. The goal of Cloud Computing is to share resources among the cloud service consumers, cloud partners, and cloud vendors in the cloud value chain. The resource sharing at various levels results in various cloud offerings such as infrastructure cloud (e.g., hardware, IT infrastructure management), software cloud (e.g. SaaS focusing on middleware as a service, or traditional CRM as a service), application cloud (e.g., Application as a Service, UML modeling tools as a service, social network as a service), and business cloud (e.g., business process as a service).

Index Terms: Data integrity, dependable distributed storage, Cloud Computing

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 powerful processors, together with the software as a service (SaaS) computing architecture, are transforming data centers into pools of computing service on a huge scale. The increasing network bandwidth and reliable yet flexible network connections make it even possible that users can now subscribe high quality services from data and software that reside solely on remote data centers.

Although cloud computing promises lower costs, rapid scaling, easier maintenance, and service availability anywhere, anytime, a key challenge is how to ensure and build confidence that the cloud can handle user data securely. A recent Microsoft survey found that “58 percent of the public and 86 percent of business leaders are excited about the possibilities of cloud computing. But more than 90 percent of them are worried about security, availability, and privacy of their data as it rests in the cloud.”[1]

In order to achieve the assurances of cloud data integrity and availability and enforce the quality of cloud storage service, efficient methods that enable on-demand data correctness verification on behalf of cloud users have to be designed. However, the fact that users no
longer have physical possession of data in the cloud prohibits the direct adoption of traditional cryptographic primitives for the purpose of data integrity protection. Hence, the verification of cloud storage correctness must be conducted without explicit knowledge of the whole data files . Meanwhile, cloud storage is not just a third party data warehouse. The data stored in the cloud may not only be accessed but also be frequently updated by the users, including insertion, deletion, modification, appending, etc. Thus, it is also imperative to support the integration of this dynamic feature into the cloud storage correctness assurance, which makes the system design even more challenging. Last but not the least, the deployment of Cloud Computing is powered by data centers running in a simultaneous, cooperated
and distributed manner [2].

In this paper, we propose an effective and flexible distributed storage verification scheme with explicit dynamic data support to ensure the correctness and availability of users’ data in the cloud. We rely on erasure correcting code in the file distribution preparation to provide
redundancies and guarantee the data dependability against Byzantine servers [23], where a storage server may fail in arbitrary ways. 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). In order to strike a good balance between error resilience and data dynamics, we further explore the algebraic property of our token computation and erasure-coded data, and demonstrate how to efficiently support dynamic operation on data blocks, while maintaining the same level of storage correctness assurance. In order to save 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. Our work is among the first few ones in this
field to consider distributed data storage security 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 status across the distributed servers, the proposed scheme achieves the integration of storage correctness insurance and data error localization, i.e., the identification of misbehaving
server(s).
2) Unlike most prior works for ensuring remote data integrity, the new scheme further supports secure and efficient dynamic operations on data blocks, including:update, delete and append.
3) The experiment results demonstrate the proposed scheme is highly efficient. Extensive security analysis shows our scheme is resilient against Byzantine failure, malicious data modification attack, and even server colluding attacks.

II. PROBLEM STATEMENT

2.1 System Model

A representative network architecture for cloud storage service architecture is illustrated in Figure 1. Three different network entities can be identified as follows:
• User: an entity, who has data to be stored in the cloud and relies on the cloud for data storage and computation, can be either enterprise or individual customers.
• Cloud Server (CS): an entity, 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.).
• 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.

In cloud data storage, a user stores his data through a CSP into a set of cloud servers, which are running in a simultaneous, cooperated and distributed manner. Data redundancy can be employed with technique of erasure correcting code to further tolerate faults or server crash as user's data grows in size and importance. Thereafter, for application purposes, the user interacts with the cloud servers via CSP to access or retrieve his data.

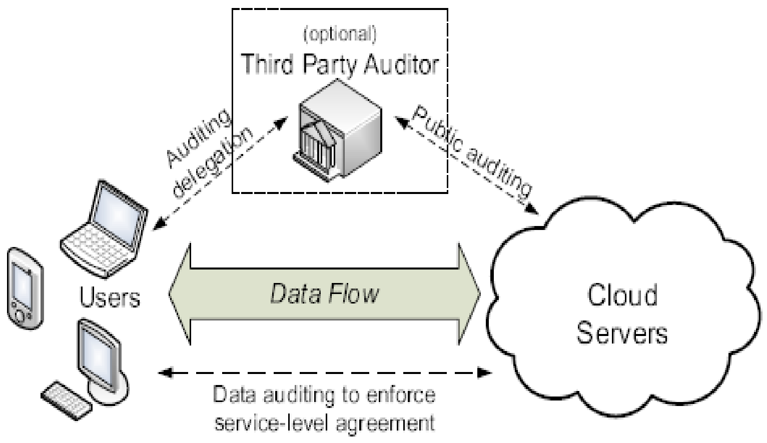


Fig. 1: Cloud storage service architecture

2.2 Data Protection As a Service

Currently, users must rely primarily on legal agreements and implied economic and reputational harm as a proxy for application trustworthiness. As an alternative, a cloud
platform could help achieve a robust technical solution by
• making it easy for developers to write maintainable applications that protect user data in the cloud, thereby providing the same economies of scale for security and privacy as for computation and storage; and
• enabling independent verification both of the platform's operation and the runtime state of applications on it, so users can gain confidence that their data is being handled properly.

Much as an operating system provides isolation between processes but allows substantial freedom inside a process, cloud platforms could offer transparently verifiable
partitions for applications that compute on data units, while still allowing broad computational latitude within those partitions. DPaaS enforces fine-grained access control policies on data units through application confinement and information flow checking. It employs cryptographic protections at rest and offers robust logging and auditing to provide accountability. Crucially, DPaaS also directly addresses the issues of rapid development and maintenance. To truly support this vision, cloud platform providers would have to offer DPaaS in addition to their existing hosting environment, which could be especially beneficial
for small companies or developers who don't have much in-house security expertise, helping them build user confidence much more quickly than they otherwise might. Figure 2 illustrates an example architecture for exploring the DPaaS design space.[3] Here, each server contains
a *trusted platform module* (TPM) to provide secure and verifiable boot and dynamic root of trust. This example architecture demonstrates at a high level how it's potentially possible to combine various technologies such as application confinement, encryption, logging, code attestation, and information flow checking to realize DPaaS.

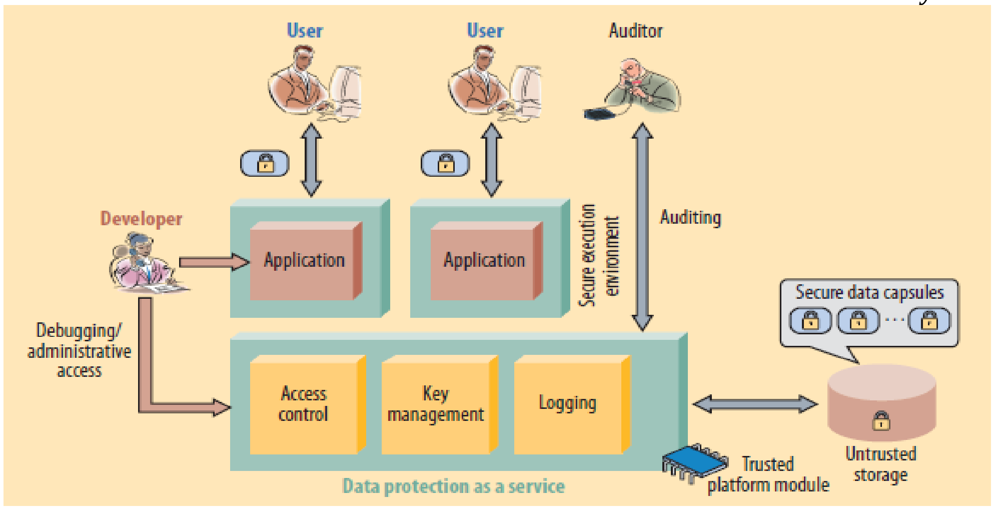


Figure 2: Sample architecture for data protection as a service

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 always be the first step to fast recover the storage errors and/or identifying potential threats of external attacks.

To address these problems, our main scheme for ensuring cloud data storage is presented in this section. The first part of the section is devoted to a review of basic tools from coding theory that is needed in our scheme for file distribution across cloud servers. Then, the homomorphic token is introduced. The token computation function we are considering belongs to a family of universal hash function [4], chosen to preserve the homomorphic properties, which can be perfectly integrated with the verification of erasure-coded data [5] [6].Subsequently, it is shown how to derive a challenge response protocol for verifying the storage correctness as well as identifying misbehaving servers. The procedure for file retrieval and error recovery based on erasure correcting code is also outlined. Finally, we describe how to extend our scheme to third party auditing with only slight modification of the main design.

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.

Since data do not reside at users' local site but at cloud service provider's address domain, supporting dynamic data operation can be quite challenging. On the one hand, CSP needs to process the data dynamics request without knowing the secret keying material. On the other hand, users need to ensure that all the dynamic data operation request has been faithfully processed by CSP. To address this problem, we briefly explain our approach methodology here and provide the details later. For any data dynamic operation, the user must first generate the corresponding resulted file blocks and parities. This part of operation has to be carried out by the user, since only he knows the secret matrix P. Besides, to ensure the changes of data blocks correctly reflected in the cloud address domain, the user also needs to modify the corresponding storage verification tokens to accommodate the changes on data blocks. Only with the accordingly changed storage verification tokens, the previously discussed challenge-response protocol can be carried on successfully even after data dynamics. In other words, these verification tokens help ensure that CSP would correctly execute the processing of any dynamic data operation request. Otherwise, CSP would be caught cheating with high probability in the protocol execution later on. Given this design methodology, 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, by utilizing the linear property of Reed-Solomon code and verification token construction.

4.1 Update Operation

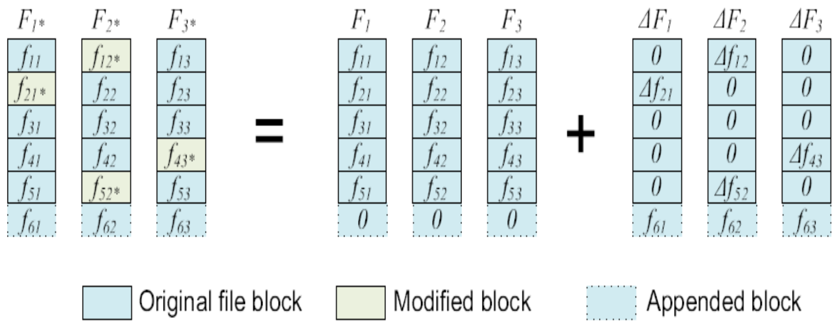
In cloud data storage, a user may need to modify some data block(s) stored in the cloud, from its current value fij to a new one, fij + Δfij . We refer this operation as data update. Figure 2 gives the high level logical representation of data block update. Due to the linear property of Reed-Solomon code, a user can perform the update operation and generate the updated parity blocks by using Δfij only, without involving any other unchanged blocks. Specifically, the user can construct a general update matrix Δ**F** as

$$\Delta F = \begin{pmatrix} \Delta f_{11} & \Delta f_{12} & \ldots & \Delta f_{1m} \\ \Delta f_{21} & \Delta f_{22} & \ldots & \Delta f_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \Delta f_{l1} & \Delta f_{l2} & \ldots & \Delta f_{lm} \end{pmatrix} = (\Delta F_1, \Delta F_2, \ldots, \Delta F_m).$$

Note that we use zero elements in ΔF to denote the unchanged blocks and thus ΔF should only be a sparse matrix most of the time (we assume for certain time epoch, the user only updates a relatively small part of file **F**). To maintain the corresponding parity vectors as well as be consistent with the original file layout, the user can multiply Δ**F** by A and thus generate the update information for both the data vectors and parity vectors as follows:

$$\begin{aligned} \Delta F \cdot A &= (\Delta G^{(1)}, \ldots, \Delta G^{(m)}, \Delta G^{(m+1)}, \ldots, \Delta G^{(n)}) \\ &= (\Delta F_1, \ldots, \Delta F_m, \Delta G^{(m+1)}, \ldots, \Delta G^{(n)}), \end{aligned}$$

where ΔG(j) (j ∈ {m + 1, . . . , n}) denotes the update information for the parity vector G(j).



4.2 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. Therefore, we can rely on the update procedure to support delete operation, i.e., by setting Δfij in ΔF to be −Δfij . Also,
all the affected tokens have to be modified and the updated parity information has to be blinded using the same method specified in update operation.

4.3 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. Given the file matrix F illustrated in file distribution preparation, appending blocks towards the end of a data file is equivalent to concatenate corresponding rows at the bottom of the matrix layout for file F (See Figure
2). In the beginning, there are only l rows in the file matrix. To simplify the presentation, we suppose the user wants to append m blocks at the end of file F, denoted as (fl+1,1, fl+1,2, ..., fl+1,m) (We can always use zeropadding to make a row of m elements.).

4.4 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. In order to fully support block insertion operation, recent work [7], [8] suggests utilizing additional data structure (for example, Merkle Hash Tree [9]) to maintain and enforce the block index information. Following this line of research, we can circumvent the dilemma of our block insertion by viewing each insertion as a logical append operation at the end of file F. Specifically, if we also use additional data structure to maintain such logical to physical block index mapping information, then all block insertion can be treated via logical

append operation, which can be efficiently supported. On the other hand, using the block index mapping information, the user can still access or retrieve the file as it is. Note that as a tradeoff, the extra data structure information has to be maintained locally on the user side.

V. CONCLUSION

In this paper, we investigate the problem of data security in cloud data storage, which is essentially a distributed storage system. To achieve the assurances of cloud data integrity and availability and enforce the quality of dependable cloud storage service for users, we propose
an effective and flexible distributed scheme with explicit dynamic data support, 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 homomorphic token with distributed 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 distributed servers

REFERENCES

[1] C. Dwork, "The Differential Privacy Frontier Extended Abstract," *Proc. 6th Theory of Cryptography Conf.* (TCC 09),

[2] Sun Microsystems, Inc., "Building customer trust in cloud computing with transparent security," Online at https://www.sun.com/offers/details/sun transparency.xml,

[3] P. Maniatis et al., "Do You Know Where Your Data Are? Secure Data Capsules for Deployable Data Protection,"*Proc. 13th Usenix Conf. Hot Topics in Operating Systems* (HotOS 11), Usenix, 2011

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

[5] T. Schwarz and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in *Proc. of ICDCS'06,* 2006, pp. 12–12.

[6] J. Hendricks, G. Ganger, and M. Reiter, "Verifying distributed erasure-coded data," in *Proc. of 26th ACM Symposium on Principles of Distributed Computing*, 2007, pp. 139–146

[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. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proc. of CCS'09*, 2009, pp. 213–222

[9] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. Of IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA, 1980.