International Association of Scientific Innovation and Research (IASIR)
*(An Association Unifying the Sciences, Engineering, and Applied Research)*

ISSN (Print): 2279-0047
ISSN (Online): 2279-0055

International Journal of Emerging Technologies in Computational and Applied Sciences(IJETCAS)

www.iasir.net

# Data Storage Security In Cloud Computing

Sujay Shaha[1], Pravin Shinde[2], Shankar Somatkar[3], Prof. D.K. Joshi[4]
Computer Engineering
Sinhgad Academy of Engineering
Kondhwa(bk),Pune-411048
INDIA

***Abstract**: Cloud Computing has been envisioned as the next-generation architecture of IT Enterprise. In contrast to traditional solutions, where the IT services are under proper physical, logical and personnel controls, Cloud Computing moves the application software and databases to the large data centers, where the management of the data and services may not be fully trustworthy. This unique attribute, however, poses many new security challenges which have not been well understood. In this article, we focus on cloud data storage security, which has always been an important aspect of quality of service. To ensure the correctness of users' data in the cloud, we propose an effective and flexible distributed scheme with two salient features, opposing to its predecessors. 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., the identification of misbehaving server(s). Unlike most prior works, the new scheme further supports secure and efficient dynamic operations on data blocks, including: data update, delete and append. 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.*

***Keywords:** cloud computing, data storage security, homomorphic token, byzantine failure*

## 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.

Moving data into the cloud offers great convenience 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) are both well known examples. While these internet-based online services do provide 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 availability and integrity of their data. Recent downtime of Amazon's S3 is such an example .

From the perspective of data security, which has always been an important aspect of quality of service, Cloud Computing inevitably poses new challenging security threats for number of reasons. Firstly, traditional cryptographic primitives for the purpose of data security protection cannot be directly adopted due to the users' loss control of data under Cloud Computing. Therefore, verification of correct data storage in the cloud must be conducted without explicit knowledge of the whole data. Considering various kinds of data for each user stored in the cloud and the demand of long term continuous assurance of their data safety, the problem of verifying correctness of data storage in the cloud becomes even more challenging. Secondly, Cloud Computing is not just a third party data warehouse. The data stored in the cloud may be frequently updated by the users, including insertion, deletion, modification, appending, reordering, etc. To ensure storage correctness under dynamic data update is hence of paramount importance. However, this dynamic feature also makes traditional integrity insurance techniques futile and entails new solutions. Last but not the least, the deployment of Cloud Computing is powered by data centers running in a simultaneous, cooperated and distributed manner. Individual user's data is redundantly stored in multiple physical locations to further reduce the data integrity threats. Therefore, distributed protocols for storage correctness assurance will be of most importance in achieving a robust and secure cloud data storage system in the real world. However, such important area remains to be fully explored in the literature.

Recently, the importance of ensuring the remote data integrity has been highlighted by the following research works . These techniques, while can be useful to ensure the storage correctness without having users possessing data, cannot address all the security threats in cloud data storage, since they are all focusing on single server scenario and most of them do not consider dynamic data operations. As an complementary approach, researchers have also proposed distributed protocols for ensuring storage correctness across multiple servers or peers. Again, none of these distributed schemes is aware of dynamic data operations. As a result, their applicability in cloud data storage can be drastically limited.

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 dependability. 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.

The rest of the paper is organized as follows. Section II introduces the system model, adversary model, our design goal and notations. Then we provide the detailed description of our scheme in Section III and IV. Section V gives the security analysis and performance evaluations, followed by Section VI which overviews the related work. Finally, Section VII gives the concluding remark of the whole paper.

## II. PROBLEM STATEMENT

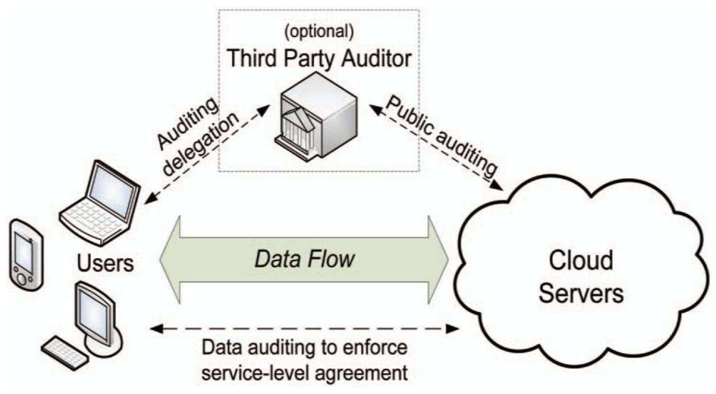
### *A. System Model*

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

- **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.

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. In some cases, the user may need to perform block level operations on his data. The most general forms of these operations we are considering are block update, delete, insert and append.

**Figure 1: Cloud data storage architecture**



As users no longer possess their data locally, it is of critical importance to assure users that their data are being correctly stored and maintained. That is, users should be equipped with security means so that they can make continuous correctness assurance of their stored data even without the existence of local copies. In case those users do not necessarily have the time, feasibility or resources to monitor their data, they can delegate the tasks to an optional trusted TPA of their respective choices. In our model, we assume that the point-to-point communication channels between each cloud server and the user is authenticated and reliable, which can be achieved in practice with little overhead. Note that we don't address the issue of data privacy in this paper, as in Cloud Computing, data privacy is orthogonal to the problem we study here.

### *B. Adversary Model*

Security threats faced by cloud data storage can come from two different sources. On the one hand, a CSP can be self-interested, untrusted and possibly malicious. Not only does it desire to move data that has not been or is rarely accessed to a lower tier of storage than agreed for monetary reasons, but it may also attempt to hide a data loss incident due to management errors, Byzantine failures and so on. On the other hand, there may also exist an economically-motivated adversary, who has the capability to compromise a number of cloud data storage servers in different time intervals and subsequently is able to modify or delete users' data while remaining undetected by CSPs for a certain period. Specifically, we consider two types of adversary with different levels of capability in this paper:

***Weak Adversary*:** The adversary is interested in corrupting theuser's data files stored on individual servers. Once a server is comprised, an adversary can pollute the original data files b y modifying or introducing its own fraudulent data to prevent the original data from being retrieved by the user.

***Strong Adversary*:** This is the worst case scenario, in whichwe assume that the adversary can compromise all the storage servers so that he can intentionally modify the data files as long as they are internally consistent. In fact, this is equivalent to the case where all servers are colluding together to hide a data loss or corruption incident.

### *C. Design Goals*

To ensure the security and dependability for cloud data storage under the aforementioned adversary 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 intact all the time in the cloud.

(2) **Fast localization of data error**: to effectively locate the mal-functioning server when data corruption has been detected.

(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 Byzantine failures, malicious 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.

### *D. Notation and Preliminaries*

• **F** – the data file to be stored. We assume that **F** can be denoted as a matrix of m equal-sized data vectors, each consisting of l blocks. Data blocks are all well represented as elements in Galois Field GF ($2^p$) for p = 8 or 16.

• A – The dispersal matrix used for Reed-Solomon coding.

• G – The encoded file matrix, which includes a set of n = m + k vectors, each consisting of l blocks.

• $f_{key}(\cdot)$ – pseudorandom function (PRF), which is defined as $f : \{0, 1\}^{*} \times \text{key} \rightarrow GF(2^p)$.

• Økey (·) – pseudorandom permutation (PRP), which is defined as $Ø : \{0, 1\}\log_2(l) \times \text{key} \rightarrow \{0, 1\}\log_2(l)$.

• ver – a version number bound with the index for individual blocks, which records the times the block has been modified. Initially we assume ver is 0 for all data blocks.

• $s_{ij}^{ver}$– the seed for PRF, which depends on the file name, block index i, the server position j as well as the optional block version number ver.

## 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 o n 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 fin d which server the data error lies in is also of great significance, since it can be the first step to fast recover the storage error s.

To address these problems, our main scheme for ensuring cloud data storage is presented in this section. The first par t 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 , chosen to pre-serve the homomorphic properties, which can be perfectly

**Algorithm 1: Token Pre-computation:**

```
Procedure
    Choose parameters l, n and function f, Ø;
    Choose the number t of tokens;
    Choose the number r of indices per verification;
    Generate master key $K_{prp}$ and challenge $k_{chal}$;
    For vector $G^{(j)}$, $j \leftarrow 1, n$ do
        For round $i \leftarrow 1, t$ do
            Derive $\alpha_i = f\ k_{chal}(i)$ and $k^{(i)}_{prp}$ from $K_{PRP}$.
            Compute $v_i^{(j)} = \sum^{r}_{q=1} \alpha^{q}_{i} * G^{(j)}[\text{Ø}_{k}{}^{(i)}{}_{prp}(q)]$.
        End For
    End For
    Store all the $v_i$s locally.
End Procedure
```

### *A. Challenge Token Precomputation*

In order to achieve assurance of data storage correctness and data error localization simultaneously, our scheme entirely relies on the pre-computed verification tokens. The main idea is as follows: before file distribution the user pre-computes a certain number of short verification tokens on individual vector $G^{(j)}$ ($j \in \{1, . . . , n\}$), 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 thecorresponding tokens pre-computed by the user. Meanwhile, as all servers operate over the same subset of the indices, the requestedresponse values for integrity check must also be a valid code word 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 $G^{(j)}$ ($j \in \{1, \ldots, n\}$), using a PRF $f(\cdot)$, a PRP $\varnothing(\cdot)$, a challenge key $k_{chal}$ and a master permutation key $K_{PRP}$ . To generate the i<sup></sup>th token for server j, the user acts as follows:

1) Derive a random challenge value $\alpha_i$ of GF $(2^p)$ by $\alpha i = = f\, k_{chal}$ (i) and a permutation key $k^{(i)}_{prp}$ based on $K_{PRP}$

2) Compute the set of r randomly-chosen indices: $\{I_q \in [1, \ldots, l] | 1 \le q \le r\}$,where $I_q = \varnothing_k{}^{(i)}{}_{prp}$ (q) .

3) Calculate the token as:

$v_i^{(j)} = \sum^r_{q=1} \alpha^q_i * G^{(j)}[I_q],\ G^{(j)}[I_q] = g_{Iq}^{(j)}$.

Note that $v_i^{(j)}$, which is an element of $GF(2^p)$ with small size, is the response the user expects to receive from server j when he challenges it on the specified data blocks.

After token generation, the user has the choice of either keeping the pre-computed tokens locally or storing them in encrypted form on the cloud servers. In our case here, the user stores them locally to obviate the need for encryption and lower the bandwidth overhead during dynamic data operation which will be discussed shortly. The details of token generation are shown in Algorithm 1.

Once all tokens are computed, the final step before file distribution is to blind each parity block $g_i^{(j)}$ in $(G^{(m+1)}, \ldots, G^{(n)})$ by

$g_i^{(j)} \leftarrow g_i^{(j)} + f k_j\, (s_{ij}),\ i \in \{1, \ldots, l\}$,

Where kj is the secret key for parity vector G(j) ($j \in \{m + 1, \ldots, n\}$). This is for protection of the secret matrix P. We will discuss the necessity of using blinded parities in detail in Section V. After blinding the parity information, the user disperses all the n encoded vectors $G^{(j)}$ ($j \in \{1, \ldots, n\}$) across the cloud servers $S_1, S_2, S_3, \ldots\ldots, S_n$.

**Algorithm 2: Correctness Verification and localization**

```
Procedure CHALLENGE (i)
    Recompute=$fk_{chal}$ (i) and $k^{(i)}_{prp}$ from $K_{PRP}$ ;
    Send { $\alpha_i$, $k^{(i)}_{prp}$} to all cloud servers;
        Receive from servers:
{$R_i^{(j)} = \sum^r_{q=1} \alpha^q_i * G^{(j)}[\varnothing_k{}^{(i)}{}_{prp}\ (q)] | 1 < j \le n$}
    For ($j \leftarrow m + 1, n$) do
        $R^{(j)} \leftarrow R^{(j)} - \sum^r_{q=1} f k_j(s_{Iq,j}) \cdot \alpha_i^q, I_q = \varnothing_k{}^{(i)}{}_{prp}\ (q)$
    End For
    if (($R_i^{(l)}, \ldots, R_i^{(m)}$) $\cdot P == (R_i^{(m+1)}, \ldots, R_i^{(n)}$) then
        Accept and ready for the next challenge.
    Else
        For ($j \leftarrow 1, n$) do
            If ($R_i^{(j)}\ != v_i^{(j)}$ ) then
                returnserver j is misbehaving.
        End If
        End For
    End If
End Procedure
```

### *B. Correctness Verification and Error Localization*

Error localization is a key prerequisite for eliminating errors in storage systems. However, many previous schemes do not explicitly consider the problem of data error localization, thus only provide binary results for the storage verification. Our scheme outperforms those by integrating the correctness verification and error localization in our 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). Specifically, the procedure of the $i^{th}$ challenge-response for a cross-check over the n servers is described as follows:

1) The user reveals the $\alpha_i$ as well as the $i^{th}$ permutation key $k^{(i)}{}_{prp}$ to each servers.

2) The server storing vector $G^{(j)}$ aggregates those r rows specified by index $k^{(i)}{}_{prp}$ into a linear combination

$R_i^{(j)} = \sum^{r}_{q=1} \alpha^q{}_i * G^{(j)}[\text{Ø}_k{}^{(i)}{}_{prp}\ (q)]$

3) Upon receiving R(j)i s from all the servers, the user takes away blind values in

R(j) ($j \in \{m+1, \ldots, n\}$) by

$R^{(j)} \leftarrow R^{(j)} - \sum^{r}_{q=1} f k_j(s_{Iq,j}).\ \alpha_i{}^q$, where $I_q = \text{Ø}_k{}^{(i)}{}_{prp}\ (q)$

4) Then the user verifies whether the received values remain a valid codeword determined by secret matrix

$P:(R_i^{(l)}, \ , \ldots, R_i^{(m)},).P?= ==(R_i^{(m+1)}, \ldots, R_i^{(n)})$

Because all the servers operate over the same subset of indices, the linear aggregation of these r specified rows $(R_i^{(l)}, \ldots, R_n^{(j)}\ )$ has to be a codeword in the encoded file matrix. If the above equation holds, the challenge is passed. Otherwise, it indicates that among those specified rows, there exist file block corruptions. Once the inconsistency among the storage has been successfully detected, we can rely on the pre-computed verification tokens to further determine where the potential data error(s) lies in. Note that each response R(j)i is computed exactly in thesame way as token v(j)i , thus the user can simply find which server is misbehaving by verifying the following n equations:

$R_i^{(j)} = v_i^{(j)}\ , j \in \{1, \ldots, n\}.$

### *C. File Retrieval and Error Recovery*

Since our layout of file matrix is systematic, the user can reconstruct the original file by downloading the data vector s from the first m servers, assuming that they return the correct response values. Notice that our verification scheme is base d 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 regenerate the correct blocks by erasure correction, shown in Algorithm 3, as long as there are at most k misbehaving servers are identified. The newly recovered blocks can then be redistributed to the misbehaving servers to maintain the correctness of storage.

**Algorithm 3: 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
```

## 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. CONCLUSION

In this paper, we investigated the problem of data security in cloud data storage, which is essentially a distributed storage system. To ensure the correctness of users' data in cloud data storage, we proposed 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, we can almost guarantee the simultaneous identification of the misbehaving server(s). Through detailed security and performance analysis, we show that our scheme is highly efficient and resilient to Byzantine 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 paramount importance, is stillin its infancy now, and many research problems are yet to be identified. We envision 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 , allows TPA to audit the cloud data storage without demanding users' time, feasibility 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 dynamic data. Besides, along with our research on dynamic cloud data storage, we also plan to investigate the problem of fine-grained data err or localization.

## VI. REFERENCES

1. Cong Wang, Qian Wang, KuiRen and Ning Cao, “Toward Secure And Dependable Storage Services in Cloud Computing” in IEEE Transactions on Services Computing April/June-2012.
2. 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.

3. Hans P. Reiser, University of Lisbon Faculty of Science, Portugal, " Byzantine Fault Tolerance for the Cloud"
4. Sara Qaisar, "Cloud Computing: Network/Security Threats AndCountermeasures".
5. Amazon.com, "Amazon Web Services (AWS)," Online at http://aws.amazon.com, 2008.