ISSN:0975-9646

Ensuring and Reliable Storage in Cloud Computing

Katukam Ganesh
MTech(CSE)
JNTU Hyderabad

Maligireddy Saidireddy
HOD (CSIT)
JNTU Hyderabad

KrishnaChaitanya.Katkam
Asst Prof (CSE)
Nigama Engineering College

ABSTRACT-Cloud storage enables users to remotely store their data and enjoy the on-demand high quality cloud applications without the burden of local hardware and software management. Though the benefits are clear, such a service is also relinquishing users' physical possession of their outsourced data, which inevitably poses new security risks towards the correctness 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

1 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) [1] 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 anexample[2]. From the perspective of data security, which has always been an important aspect of quality of service, Cloud Com-puting 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.

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 prov ide 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.

2 PROBLEM STATEMENT

2.1 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, 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).

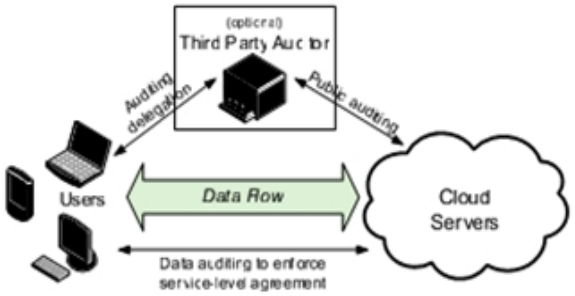
* Third Party Auditor (TPA): an optional TPA, who has expertise and capabilities that users may not have, is trusted to assess and expose risk of cloud storage services on behalf of the users upon request.

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. 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.

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 that 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.

2.2 Adversary Model

From user's perspective, the adversary model has to capture all kinds of threats towards his cloud data integrity. Because cloud data do not reside at user 's local site but at CSP's address domain, these threats can come from two different sources: internal and external attacks. For internal attacks, 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. For external attacks, data integrity threats may come from outsiders who are beyond the control domain of CSP, for example, the economically motivated attackers. They may compromise a number of cloud data storage servers in different time intervals and subsequently be able to modify or delete users' data while remaining undetected by CSP.



2.3 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 verifica - tion 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 effect ively locate the malfunctioning server when data corrupt - tion 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) Light weight: to enable users to perform storage correctness checks with minimum overhead.

3. 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 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. Subsequently, it is also shown how to derive a challenge-response protocol for verifying the storage correctness as well as identifying misbehaving servers. Finally, the procedure for file retrieval and error recovery based on erasure-correcting code is outlined.

3.1 File Distribution Preparation

It is well known that erasure-correcting code may be used to tolerate multiple failures in distributed storage systems. In cloud data storage, we rely on this technique to disperse the data file **F** redundantly across a set of **n = m + k** distributed servers. An**(m, k)** Reed-Solomon erasure-correcting code is used to create **k** redundancy parity vectors from **m** data vectors in such a way that the original **m** data vectors can be reconstructed from any **m** out of the **m + k** data and parity vectors. By placing each of the **m + k** vectors on a different server, the original data file can survive the failure of any **k** of the **m + k** servers without any data loss, with a space overhead of **k/m**. For support of efficient sequential I/O to the original file, our file layout is systematic, i.e., the unmodified **m** data file vectors together with **k** parity vectors is distributed across **m + k** different servers.

Let $F = (F_1, F_2, \ldots, F_m)$ and $F_i = (f_{1i}, f_{2i}, \ldots, f_{li})^T$ ($i \in \{1, \ldots, m\}$). Here T (shorthand for transpose) de-notes that each F_i is represented as a column vector, and l denotes data vector size in blocks. All these blocks are elements of GF (2^p). The systematic layout with parity vectors is achieved with the inform -ation dispersal matrix A, derived from an m×(m+k) Vandermonde matrix [26]:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ \beta_1 & \beta_2 & \beta m & \beta m+1 & \beta n \\ & & & & \\ \beta^{1-1} & \beta^{2-1} & \beta^{m-1} & \beta^{m+1} & \beta^{n-1} \end{pmatrix},$$

where β_j ($j \in \{1, \ldots, n\}$) are distinct elements randomly

After a sequence of elementary row transform ationns, the desired matrix A can be written as

$$\mathbf{A = (I / P)} = \begin{pmatrix} 1 & 0 & \cdots & 0 & p_{11} & p_{12} & \cdots & p_{1k} \\ 0 & 1 & \cdots & 0 & p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{m1} & p_{m2} & \cdots & p_{mk} \end{pmatrix}$$

Where I is a m × m identity matrix and P is the secret parity generation matrix with size m × k. Note that A is derived from a Vander monde matrix, thus it has the prop erty that any m out of the + k columns form an invertible matrix. By multiplying F by A, the user obtains the encoded file:

$$G = F \cdot A = (G^{(1)}, G^{(2)}, \ldots, G^{(m)}, G^{(m+1)}, \ldots, G^{(n)}) = (F_1, F_2, \ldots, F_m, G^{(m+1)}, \ldots, G^{(n)}),$$

where $G^{(j)} = (g^{1j}, g^{2j}, \ldots, g^{lj})^T$ ($j \in \{1, \ldots, n\}$).

As noticed, the multiplication reproduces the original data file vectors of F and the remaining part ($G^{(m+1)}, \ldots, G^{(n)}$) are k parity vectors generated based on F.

3.2 Challenge Token Pre-computation

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-compute s a certain number of short verification tokens on individual ve ctor $G^{(j)}$ ($j \in \{1, \ldots, n\}$), each token covering a random subset of data blocks. Later, when the user wants to make sure the storage correctness for the data in the cloud, he challenges the cloud servers with a set of randomly generated block indices. Upon receiving challenge, each cloud server computes a short "signature" over the specified blocks and returns the m 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.

Algorithm 1 Token Pre-computation

1: **procedure**
2: Choose parameters l, n and function f, φ;
3: Choose the number t of tokens;
4: Choose the number r of indices per verification;
5: Generate master key K_{prp} and challenge k_{chal};
6: **for** vector $G^{(j)}$, j ← 1, n **do**
7: **for** round i← 1, t **do**
Derive α = $\mathbf{f_{kchal}(i)}$
and $\mathbf{k^{(i)}_{prp}}$ **from** $\mathbf{K_{PRP}}$
Compute $\mathbf{V_i^{(j)}} = \sum_{q=1}^{r} \alpha^q_i * G^{(j)}[\emptyset_{k^{(i)}_{prp}}(q)]$
end for
10: **end for**
11: **end for**
12: Store all the v_is locally.
13: **end procedure**

3.3 Correctness Verification and Error Localization

Error localization is key pre requisitefor eliminating errors in storage systems.However, many previous schemes don't explicitly consider the problem of data errorlocalization, Our scheme outperforms those by integrating the correctess verification and error localization in our challenge response protocol: the response values from servers for each challen ge not only determine the correctness of the distributed storage,but also contain information to locate potential data error(s).

Algorithm 2

1) **procedure** CHALLENGE(i)
2) Recompute $\alpha_i = f_{kchal}(i)$ and $k_{prp}^{(i)}$ from K_{PRP} ;

3) Send $\{\alpha_i, k_{prp}^{(i)}\}$ to all the cloud servers;
4) Receive from servers:
$\{R_i^{(j)} = \sum_{q=1}^{r} \boldsymbol{\alpha^q_i}\, G^{(j)}\, \mathbf{[Ø_{kprp}^{(i)}}(q)] | 1 \le j \le n\}$
for(j←m+1,n)do
$R^{(j)} \leftarrow R^{(j)} - \sum_{q=1}^{r} f_{kj}(SI_q, j).\alpha^q_i, I_q = Ø_{kprp}^{(i)}(q)$

7: **end for**
8: **if** $((R_i^{(1)}, \ldots, R_i^{(m)}) \cdot P == (R_i^{(m+1)}, \ldots, R_i^{(n)}))$ **then**
9: Accept and ready for the next challenge.
10: **else**
11: **for** $(j \leftarrow 1, n)$ **do**
12: **if** $(R_i^{(j)} != v_i^{(j)})$ **then**
13: **return** server j is misbehaving.
14: **end if**
15: **end for**
16: **end if**
17: **end procedure**

3.4 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 assurance is a probabilistic one. However, by choosing system param-eters (e.g., r, l, t) appropriately and conducting enough times of verification, we can guarantee the successful file retriev al 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 identificati on 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

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**

3.5 Towards 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 [27], [28], to securely introduce an effective TPA, the auditing process should bring in no new vulnerabilities towards user data pri-vacy. Namely, TPA should not learn user 's data content through the delegated data auditing. Now we show that with only slight modification, our protocol can support privacy-preserving third party auditing

The new design is based on the observation of linear property of the parity vector blinding process. Recall that the reason of blinding process is for protection of the secret matrix P against cloud servers.

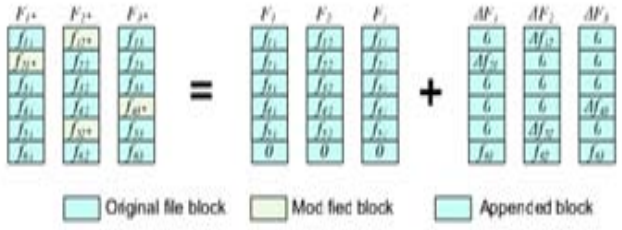
4. PROVIDING DYNAMIC DATA OPERATION SUPPORT

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 perations of update, delete and append to modify the data file while maintaining the storage correctness assurance.

In this section, we will show how our scheme can explicitly and efficiently handle dynamic data operations for cloud data storage.

4.1 Update Operation

In cloud data storage, sometimes the user may need to modify some data block(s) stored in the cloud, from its current value f_{ij} to a new one, $f_{ij} + f_{ij}$. We refer this operation as data 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 f_{ij} only, without involving any other unchanged blocks.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 f_{ij} in F to be $-f_{ij}$. 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.



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

where $\Delta G^{(j)}$ $(j \in \{m+1, \ldots, n\})$ 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 _fij in _F to be−_fij . Also, al l the affected tokens have to be modified and he updated parity information has to be blinded using thesame 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. In the beginning, there areonly l rows in the file matrix. To simplify the presentation ,we suppose the user wants to append m blocks at the end offile F, denoted as (fl+1,1, fl+1,2, ..., fl+1,m) (We can always use zero padding to make a row of m elements.). With the secret matrix P, the user can directly calculate the append

(fl+1,1, ..., fl+1,m) · P = (g (m+1)), ..., g(n))

To support block append operation, we need a slight modification to our token precomputation. Specifically, we require the user to expect the maximum size in blocks, denoted as lmax, for each of his data vector. The idea of supporting block append, which is similar as adopted in [13], relies on the initial budget for the maximum anticipa ted data size lmaxin each encoded data vector as well as the system parameter rmax = ⌈ r ∗ (lmax/l)⌉ for each pre computed challenge response token. The pre-computation of the i-th token onserver j is modified as follows:

$$G^{(j)}[I_q] = \begin{cases} G^{(j)}_{prp}[\varphi_k(i)(q)] & , [\varphi_k(i)(q)] \le l \\ 0 & , [\varphi_k(i)(q)] > l \end{cases}$$

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., inser ting a block F [j] corresponds to shifting all blocks starting with index j + 1 by one slot. 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. Therefore, an efficient insert operation is difficult to supp ort and thus we leave it for our future work.

5. Security Analysis And Performance Evaluation

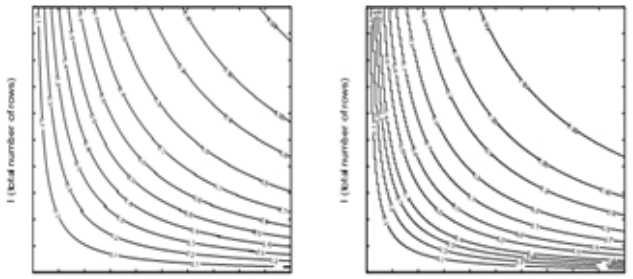
In this section, we analyze our proposed scheme in terms of security and efficiency. Our security analysis focuses on the adversary model defined in Section II. We also evaluate the efficiency of our scheme via implementation of both file dist r ibution preparation and verification tokenpre computation.

5.1 Correctness Analysis

First, we analyze the correctness of th verification procedure. Upon obtaining all the response R^{ij}s from servers and taking a way the random blind values from R^{ij} ($j \in \{m+1, \ldots, n\}$), the user relies on the equation $(R^{i1}, \ldots, R^{im}) \cdot P = (R^{im+1)}, \ldots, R^{in)})$ to ensure the storage currectness

5.2 Security Strength

In this section, we analyze the security strength of our schemes against server colluding attack and explain why blind-ing the parity blocks can help improve the security strength of our proposed scheme.Recall that in the file distribution preparation, the redun-dancy parity vectors are calculated via multiplying the file matrix **F** by **P**, where **P** is the secret parity generation matrix we later rely on for storage correctness assurance. If we disperse all the generated vectors directly after token precomputation, i.e., without blinding, malicious servers that collaborate can reconstruct the secret P matrix easily: they can pick blocks from the same rows among the data and parity vectors to establish a set of $m \cdot k$ linear equations and solve for the $m \cdot k$ entries of the parit generation matrix **P**. Once they have the knowledge of **P**, those malicious servers can consequently modify any part of the data blocks and calculate the corresponding parity blocks, and vice versa, making their codeword relationship always consistent. Therefore, our stor-age correctness challenge scheme would be undermined—even if those modified blocks are covered by the specified rows, the storage correctness check equation would always hold.



(number of queried rows) r (number of queried rows)
(as a percentage of l)0 (as a percentage of l)0

5.3 Performance Evaluation

We now assess the performance of the proposed storage auditing scheme. We focus on the cost of file distribu tion preparation as well as the token generation. Our experiment

is conducted on a system with an Intel Core 2 processor running at 1.86 GHz, 2048 MB of RAM, and a 7200 RPM Western Digital 250 GB Serial ATA drive Algorithms are implemented using open-source erasure coding library Jerasure [31] written in C. All results represent the mean of 20 trials.

1) File Distribution Preparation:

We implemented the gen- As discussed, file distribution preparation includes the generation of parity vectors (the encoding part) as well as the corresponding parity blinding part. We consider two sets of different parameters for the (m, k) Reed-Solomon encoding ,both of which work over GF (2^{16}). Figure 4 shows the total cost for preparing a 1 GB file before outsourcing. In the figure on the left, we set the number of data vectors m constant at10,while decreasing the number of parity vec tors k from 10 to 2. In the one on the right, we keep the total number of data and parity vectors m + k fixed at 22, and change the number of data vectors m from18 to10.

2) Challenge Token Pre-computation

Although in our scheme the number of verification token t is a fixed priori determined before file distribution, we can overcome this issue by choosing sufficient large t in practice. For example, when t is selected to be 7300 and 14600, the data file can be verified every day for the next 20 years and 40 years, respectively, which should be of enough use in practice. Note that instead of directly computing ach token, our implementation uses the Horner algorithm suggested in [21] to calculate token $v^{ij)}$ from the back, and achieves a slightly faster which only require sr multiplication and (r− 1) X OR operations. With Jerasure library [31],the multiplication over GF (2^{16}) in our experiment is based on discrete logarithms.

RELATED WORK

Juels et al. [9] described a formal "proof of retrieve ability" (POR) model for ensuring the remote data integrity. Their scheme combines spot-checking and error correcting code to ensure both possession and retrieve ability offiles on archive service systems. Shacham etal. [16] 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. [17]proposed an improved framework for POR protocols that generalizes both Juels and Shacham's work. Later in their subsequent work, Bowers et al. [20] 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. [19] gave theoretical studies on generalized framework different variants of existing POR work.

CONCLUSION

In this paper, we investigate the problem of data security yin cloud data storage, which is essentially a distribute dstorage 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 thefile 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).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 beworry-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.

ACKNOWLEDGMENTS:

This work was supported in part by the US National Science Foundation under grant CNS-0831963, CNS-0626601, CNS-0716306, and CNS-0831628.

REFERENCES

[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.19.

[2] Amazon.com, "Amazon web services (aws)" Online at http://aws.amazon.com/, 2009.

[3] Sun Microsystems, Inc., "Building customer trust in cloud com-puting with transparent security," Online at https://www.sun. com/offers/details/sun transparency.xml, Nov2009

[4]M.Arrington,"Gmail disaster: Reports or mass email deletions"Online at 06/12/28/gmail- disasterreports-of-mass-email-deletions/,December2006

[5]Amazon.com, "Amazon s3 availability event: July 20,2008," Online at http://status.aws.amazon.com/s3-20080720.html, July2008.

[6]S.Wilson,"Appengine outage," Online at http://www.cio-weblog.com/50226711/appengine outage.php, June 2008.

[7] B. Krebs, "Payment Processor Breach May Be Largest Ever," On-line at http://voices.washingtonpost.com/ securityfix/2009/ 01/payment processor breach may b.html, Jan. 2009.

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

[9] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peter-son, and D. Song, "Provable data possession at untrusted stores,"in Proc. of CCS'07, Alexandria, VA, October 2007,pp. 598-609

[10] M.A.Shah, M. Baker, J. C.Mogul, and R. Swaminath an, "Audit-ing to keep online storage services honest," in Proc. of HotOS'07.Berkeley, CA, USA: USENIX Association, 2007, pp. 16.

[11] M. A. Shah, R. Swaminathan, and M. Baker, "Priva cy-preserving audit and extraction of digital contents," Cryptology ePrintArchive, Report 2008/186, 2008, http://eprint.iacr.org/

[12] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "S calable and

Efficient Provable Data Possession," Proc. of SecureComm '08, pp. 1–10, 2008.

[13]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.

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

[15] H. Shacham and B. Waters, "Compact proofs of retrievability," inProc. of Asiacrypt'08, volume 5350 of LNCS, 2008, pp. 90-107

Krishnachaitanya.Katkam MTech(CSE)Completed MTech (CSE) from JNTUH in 2011. Having 4+ years ofExperience in Teaching. Present working as a Asst Prof (CSE) in Nigama EngineeringCollege. Published international journals also. Interested in Mobile Computing, Computer Networks & Mobile Application Development