Volume 3, Issue 4, April 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# A Protocol for Ensuring Data Integrity in Cloud Environment

**N.Madhuri*** *Dept of CSE* *Dhanekula Inst of Technology ,VIJ* *India.*

**T.V.Suneetha** *Dept of CSE* *GCE,Gudlavalleru* *India.*

**A.Haritha, P.V.S.Lakshmi** *Dept of Information Technology* *PVP Siddhartha Inst of Technology, VIJ* *India.*

***Abstract*— Cloud computing aims to enable end-users to easily create and use software without a need to worry about the technical implementations and nitty-gritty's such as the software's physical hosting location, hardware specifications, efficiency of data processing. It moves the application software and databases to the centralized large data centres, where the management of the data and services may not be fully trustworthy. This unique paradigm brings about many new security challenges, which have not been well understood. This work studies the problem of ensuring the integrity of data storage in Cloud Computing. In particular, we consider the task of allowing a third party auditor (TPA), on behalf of the cloud client, to verify the integrity of the dynamic data stored in the cloud. The introduction of TPA eliminates the involvement of the client through the auditing of whether his data stored in the cloud is indeed intact. The support for data dynamics via the most general forms of data operation, such as block modification, insertion and deletion, is also a significant step toward practicality, since services in Cloud Computing are not limited to archive or backup data only. While prior works on ensuring remote data integrity often lacks the support of either public audit ability or dynamic data operations, this paper achieves both. In particular, to achieve efficient data dynamics, we improve the existing proof of storage models by manipulating block tag authentication. To support efficient handling of multiple auditing tasks, we further explore the technique of signature to extend our main result into a multi-user setting, where TPA can perform multiple auditing tasks simultaneously. Extensive security and performance analysis show that the proposed schemes are highly efficient and provably secure.***

***Keywords*— Cloud computing, data dynamics, data integrity, error localization, public verifiability.***

## I. INTRODUCTION

Cloud computing is the long dreamed vision of computing as a utility, where users can remotely store their data into the cloud so as to enjoy high quality applications and services from a shared pool of configurable computing resources (fig 1). It has been envisioned as the on-demand self-service, ubiquitous network access, location independent resource pooling, rapid resource elasticity, usage-based pricing and transference of risk [1].

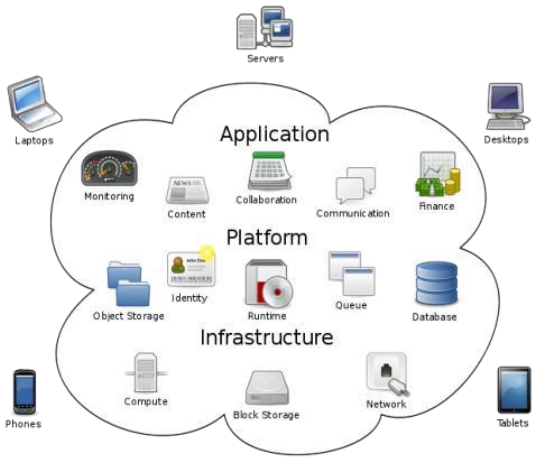


Fig 1.Cloud Computing

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 centres into pools of computing service on a huge scale. Meanwhile, the increasing network bandwidth and reliable yet flexible network connections make it even possible that clients can now subscribe high quality services from data and software that reside solely on remote data centres. Although envisioned as a promising service platform for the Internet, this new data storage paradigm in "Cloud" brings about many challenging

 

design issues which have profound influence on the security and performance of the overall system. One of the biggest concerns with cloud data storage is that of data integrity verification at un trusted servers. For example, the storage service provider, which experiences Byzantine failures occasionally, may decide to hide the data errors from the clients for the benefit of their own. Consider the large size of the outsourced electronic data and the client's constrained resource capability, the core of the problem can be generalized as how can the client find an efficient way to perform periodical integrity verifications without the local copy of data files. In order to solve this problem, many schemes are proposed under different systems and security models [1–10]. In all these works, great efforts are made to design solutions that meet various requirements: high scheme efficiency, stateless verification, unbounded use of queries and retrievability of data, etc .Considering the role of the verifier in the model, all the schemes presented before fall into two categories: private verifiability and public verifiability. Although schemes with private verifiability can achieve higher scheme efficiency, public verifiability allows anyone, not just the client (data owner), to challenge the cloud server for correctness of data storage while keeping no private information. Then, clients are able to delegate the evaluation of the service performance to an independent third party auditor (TPA), without devotion of their computation resources. In the cloud, the clients themselves are unreliable or cannot afford the overhead of performing frequent integrity checks. Thus, for practical use, it seems more ratio-nal to equip the verification protocol with public verifiability, which is expected to play a more important role in achieving economies of scale for Cloud Com-puting. Moreover, for efficiency consideration, the outsourced data themselves should not be required by the verifier for the verification purpose.

Another major concern among previous designs is that of supporting dynamic data operation for cloud data storage applications. In Cloud Computing, the remotely stored electronic data might not only be accessed but also updated by the clients, e.g., through block modification, deletion and insertion. Unfortunately, the state-of-the-art in the context of remote data storage mainly focus on static data files and the importance of this dynamic data updates has received limited attention in the data possession applications so far [1–4,6,9,11,12]. Moreover, as will be shown later, the direct extension of the current provable data possession (PDP) or proof of retrievability (PoR) [1,3] schemes to support data dynamics may lead to security loopholes. Although there are many difficulties faced by researchers, it is well believed that supporting dynamic data operation can be of vital importance to the practical application of storage outsourcing services. In view of the key role of public verifiability and the supporting of data dynamics for cloud data storage, in this paper we present a framework and an efficient construction for seamless integration of these two components in our protocol design. Our contribution can be summarized as follows: (1) We propose a general formal PoR model with public verifiability for cloud data storage, in which block-less verification is achieved; (2) We equip the proposed PoR construction with the function of supporting for fully dynamic data operations, especially to sup-port block insertion, which is missing in most existing schemes; (3) We prove the security of our proposed construction and justify the performance of our scheme through concrete implementation and comparisons with the state-of-the-art.

## II. PROBLEM STATEMENT

### *A. System Model*

Representative network architecture for cloud data storage is illustrated in Fig. 2. Three different network entities can be identified as follows: Client : an entity, which has large data files to be stored in the cloud and relies on the cloud for data maintenance and computation, can be either individual consumers or organizations; Cloud Storage Server (CSS): an entity, which is managed by Cloud Service Provider (CSP), has significant storage space and computation resource to maintain clients' data; Third Party Auditor (TPA): a TPA, which has expertise and capabilities that clients do not have, is trusted to assess and expose risk of cloud storage services on behalf of the clients upon request.

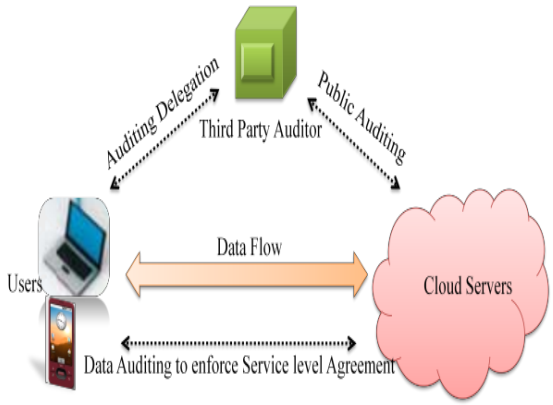


**Fig 2.Existing System**

In the cloud paradigm, the clients can be relieved of the burden of storage and computation by putting the large data files on the remote servers. As clients no longer possess their data locally, it is of critical importance for the clients to ensure that their data are being correctly stored and maintained. That is, clients should be equipped with certain security means so that they can periodically verify the correctness of the remote data even without the existence of local copies. In case that clients do not necessarily have the time, feasibility or resources to monitor their data, they can delegate the

 

monitoring task to a trusted TPA. In this paper, we only consider verification schemes with public verifiability: any TPA in possession of the public key can act as a verifier. We assume that TPA is unbiased while the server is untrusted. Note that we don't address the issue of data privacy in this paper, as the topic of data privacy in Cloud Computing is orthogonal to the problem we study here. For application purposes, the clients may interact with the cloud servers via CSP to access or retrieve their pre-stored data. More importantly, in practical scenarios the client may frequently perform block-level operations on the data files. The most general forms of these operations we consider in this paper are modification, insertion, and deletion.

*B. Security Model*

PoR Security Model is [1]. Generally, the checking scheme is secure if (i) there exists no polynomial-time algorithm that can cheat the verifier with non-negligible probability; (ii) there exists a polynomial-time extractor that can recover the original data files by carrying out multiple challenges-responses. Under the definition of this PoR system, the client can periodically challenge the storage server to ensure the correctness of the cloud data and the original files can be recovered by interacting with the server. The authors in [1] also define the correctness and soundness of PoR scheme: the scheme is correct if the verification algorithm accepts when interacting with the valid prover (e.g., the server returns a valid response) and it is sound if any cheating server that convinces the client it is storing the data file is actually storing that file. Note that in the "game" between the adversary and the client, the adversary has full access to the information stored in the server, i.e., the adversary can play the part of the prover (server). In the verification process, the adversary's goal is to cheat the client successfully, i.e., trying to generate valid responses and pass the data verification without being detected. Our security model has subtle but crucial difference from that of the original PoRs in the verification process. Note that the original PoR schemes [1,3,4,15] do not consider dynamic data operations and the block insert cannot be supported at all. This is because the construction of the signatures is involved with the file index information i. Thus, once a file block is inserted, the computation overhead is unacceptable since the signatures of all the following file blocks should be re-computed with the new indexes. To deal with this limitation, we remove the index information i in generating the signatures and use H ($m_i$ ) as the tag for block $m_i$ (see section 3.3) instead of H (name||i) [1] or h(v||i) [3], so individual data operation on any file block will not affect the others. Recall that H (name||i) or h(v||i) should be generated by the client in the verification process [1, 2]. However, in our new construction the client without the data information has no capability to calculate H ($m_i$ ). In order to successfully perform the verification while achieving blockless, the server should take over the job of computing H ($m_i$ ) and then return it to the prover. The consequence of this variance will lead to a serious problem: it will give the adversary more opportunities to cheat the prover by manipulating H ($m_i$) or $m_i$. Due to this construction; our security model differs from that of the original PoR in both the verification and the data updating process. Specifically, in our scheme tags should be authenticated in each protocol execution other than calculated or pre-stored by the verifier (The details will be shown in section 3). Note that we will use server and prover (or client, TPA and verifier) interchangeably in this paper.

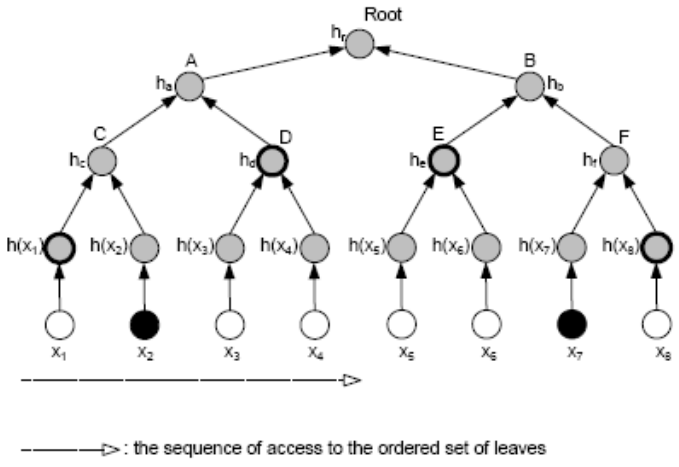


**Fig 3: Merkle hash tree authentication of data elements. We treat the leaf nodes h($x_1$ ), . . . , h($x_n$) as the left-to-right sequence.**

*C. Design Goals*

Our design goals can be summarized as the following: (1) Public verification for storage correctness assurance: to allow anyone, not just the clients who originally stored the file on cloud servers, to have the capability to verify the correctness of the stored data on demand; (2) Dynamic data operation support: to allow the clients to perform block-level operations on the data files while maintaining the same level of data correctness assurance. The design should be as efficient as possible so as to ensure the seamless integration of public verifiability and dynamic data operation support; (3) Blockless verification: no challenged file blocks should be retrieved by the verifier (e.g., TPA) during verification process for both efficiency and security concerns. (4) Stateless verification: to eliminate the need for state information maintenance at the verifier side between audits throughout the long term of data storage.

## III. THE PROPOSED SCHEME

 

*A. Modules*

Our proposal consists of various modules like

- **Data Dynamics**

Data dynamics means after clients store their data at the remote server, they can dynamically update their data at later times. At the block level, the main operations are block insertion, block modification and block deletion.

**Block Insertion:** The Server can insert anything on the client's file.
**Block Deletion:** The Server can delete anything on the client's file.
**Block Modification:** The Server can modify anything on the client's file.

- **Public verifiability**

Each and every time the secret key sent to the client's email and can perform the integrity checking operation. In this definition, we have two entities: a challenger that stands for either the client or any third party verifier, and an adversary that stands for the un trusted server. Client doesn't ask any secret key from third party.

- **Metadata key Generation:** Let the verifier V wishes to the store the file F. Let this file F consist of n file blocks. We initially pre-process the file and create metadata to be appended to the file. Let each of the n data blocks have m bits in them. A typical data file F which the client wishes to store in the cloud.

Each of the Meta data from the data blocks $m_i$ is encrypted by using a suitable algorithm to give a new modified Meta data $M_i$. Without loss of generality we show this process. The encryption method can be improvised to provide still stronger protection for Client's data. All the Meta data bit blocks that are generated using the procedure are to be concatenated together. This concatenated Meta data should be appended to the file F before storing it at the cloud server. The file F along with the appended Meta data with the cloud.

- **Privacy against Third Party Verifiers:**

Under the semi-honest model, a third party verifier cannot get Any information about the client's data m from the protocol execution. Hence, the protocol is private against third party verifiers. If the server modifies any part of the client's data, the client should be able to detect it; furthermore, any third Party verifier should also be able to detect it. In case a third party verifier verifies the integrity of the client's data, the data should be kept private against the third party verifier.

- **RSA & Metadata Generation:**

The input, and outputs $R = gs\_n\ i{=}1\ aimi \bmod N$, in which $ai = fr(i)$ for $i \in [1, n]$. Because *A* can naturally computes $P = g\_n\ i{=}1\ aimi \bmod N$ from *Dm*, *P* is also treated as *A*'s output. So *A* is given (*N, g, gs*) as input, and outputs (*R, P*) that satisfies $R = Ps$. From the KEA1-r assumption, *B* can construct an extractor $\overline{A}$, which given the same input as *A*, outputs *c* which satisfies $P = gc \bmod N$. As $P = g\_n\ i{=}1\ aimi \bmod N$, *B* extracts $c = \_ni{=}1\ aimi \bmod p\_q\_$.Now *B* generates *n* challenges $\_r1, gs1\_, \_r2, gs2\_, ..., \_rn, gsn\_$ using the method described in section III. *B*computes $aji = frj\ (i)$ for $i \in [1, n]$ and $j \in [1, n]$. Because $\{r1, r2, ..., rn\}$ are chosen by *B*, now *B* chooses them so that $\{aj\ 1, aj\ 2, ..., aj\ n\}, j = 1, 2, ..., n$.

*B. Notations and Preliminaries*

Bilinear Map: A bilinear map is a map $e : G \times G \rightarrow G_T$ , where G is a Gap Diffe-Hellman (GDH) group and $G_T$ is another multiplicative cyclic group of prime order p with the following properties [16]: (i) Computable: there exists an effciently computable algorithm for computing e; (ii) Bilinear: for all $h_1, h_2 \in G$ and $a, b \in Z_p$ , $e(h^a{}_1, h^b{}_2) = e(h_1, h_2)^{ab}$; (iii) Non-degenerate: $e(g, g) 6= 1$, where g is a generator of G.

Merkle Hash Tree: A Merkle Hash Tree (MHT) is a well-studied authenti-cation structure [17], which is intended to effciently and securely prove that a set of elements are undamaged and unaltered. It is constructed as a binary tree where the leaves in the MHT are the hashes of authentic data values. Fig. 2 depicts an example of authentication. The verifier with the authentic $h_r$ re-quests for $\{x_2 , x_7\}$ and requires the authentication of the received blocks. The prover provides the verifier with the auxiliary authentication information (AAI) $\Omega_2 =< h(x_1 ), h_d >$ and $\Omega_7 =< h(x_8), h_e >$. The verifier can then verify $x_2$ and $x_7$ by first computing $h(x_2), h(x_7 ), h_c = h(h(x_1 )||h(x_2 ))), h_f = h(h(x_7 )||h(x_8 ))),$

$h_a = h(h_c||h_d), h_b = h(h_e ||h_f )$ and $h_r = h(h_a||h_b)$, and then checking if the calculated $h_r$ is the same as the authentic one. MHT is commonly used to au-thenticate the values of data blocks. However, in this paper we further employ MHT to authenticate both the values and the positions of data blocks. We treat the leaf nodes as the left-to-right sequence, so any leaf node can be uniquely determined by following this sequence and the way of computing the root in MHT.

*C. Definitions*

$(pk, sk) \leftarrow KeyGen(1^k)$. This probabilistic algorithm is run by the client. It takes as input security parameter $1^k$ , and returns public key pk and private key sk. $(\Phi, sig_{sk} (H (R))) \leftarrow SigGen(sk, F )$. This algorithm is run by the client. It takes

 

as input private key sk and a file F which is an ordered collection of blocks $\{m_i\}$, and outputs the signature set Φ, which is an ordered collection of signatures $\{\sigma_i\}$ on $\{m_i\}$. It also outputs metadata-the signature $sig_{sk}$ (H (R)) of the root R of a Merkle hash tree. In our construction, the leaf nodes of the Merkle hash tree are hashes of H ($m_i$).

(P ) ← GenP roof (F, Φ, chal). This algorithm is run by the server. It takes as input a file F , its signatures Φ, and a challenge chal. It outputs a data integrity proof P for the blocks specified by chal.
{T RU E, F ALSE} ← V erif yP roof (pk, chal, P ). This algorithm can be run by either the client or the third party auditor upon receipt of the proof P . It takes as input the public key pk, the challenge chal, and the proof P returned from the server, and outputs T RU E if the integrity of the file is verified as correct, or F ALSE otherwise.

## IV. OUR CONSTRUCTION

Given the above discussion, in our construction, we use BLS signature [16] as a basis to design the system with data dynamics support. As will be shown, the schemes designed under BLS construction can also be implemented in RSA construction. In the discussion of section 3.4, we will show that direct extensions of previous work [1,2] have security problems and we believe that protocol design for supporting dynamic data operation is a major challenging task for cloud storage systems.

Now we start to present the main idea behind our scheme. As in the previous PoR systems [1,3], we assume the client encodes the raw data file F into F using Reed-Solomon codes and divides the encoded file F into n blocks $m_1, \ldots, m_n$[3], where $m_i \in Z_p$ and p is a large prime. Let $e : G \times G \rightarrow G_T$ be a bilinear map, with a hash function $H : \{0, 1\}^* \rightarrow G$, viewed as a random oracle [1]. Let g be the generator of G. h is a cryptographic hash function.

**The procedure of our protocol execution is as follows:**
A cloud storage system in which there is a client and an untrusted server is considered. The client stores her data in the server without keeping a local copy. Hence, it is of critical importance that the client should be able to verify the integrity of the data stored in the remote untrusted server. If the server modifies any part of the client's data, the client should be able to detect it and should not be detected by any third party verifier. In this case, when a third party verifier verifies the integrity of the client's data, is the length of each file block. Denote by fK( · ) a pseudo-random function which is defined as:$f : \{0, 1\}k \times \{0, 1\}log2(n) \rightarrow \{0, 1\}d$, in which k and d are two security parameters. Furthermore, denote the length of N in bits by |N|.

**(a) SetUp** $(1^k) \rightarrow$ (pk, sk): Given the security parameter k, this function generates the public key pk and the secret key sk. pk is public to everyone, while sk is kept secret by the client.

**(b) TagGen** (pk, sk,m) → Dm: Given pk, sk and m, this function computes a verification tag $D_m$ and makes it publicly known to everyone. This tag will be used for public verification of data integrity.

**(c) Challenge** $(pk,D_m) \rightarrow$ chal: Using this function, the verifier generates a challenge chal to request for the integrity proof of file m. The verifier sends chal to the server.
**(d) GenProof** $(pk,D_m,m,$ chal) → R: Using this function, the server computes a response R to the challenge chal. The server sends R back to the verifier.

**(e) CheckProof** $(pk,D_m,$ chal,R) → {"success", "failure"}: The verifier checks the validity of the response R. If it is valid, the function outputs "success", otherwise the function outputs "failure". The secret key sk is not needed in the CheckProof function. These functions are used for data dynamics.

**(f) CheckMisbehave**(r, $en_f$,m') → n: Let r be the number of different rows for which the user asks for

the data should be kept private against the third party verifier.The proposed protocol is correct in the sense that the server can pass the verification of data integrity as long as both the client and the server are honest. Then the protocol is secure against the untrusted server. The protocol guarantee is that, assuming the client is honest, if and only if the server has access to the complete and uncorrupted data, it can pass the verification process successfully. Finally the protocol is private against third party verifiers. To design the remote data integrity checking, Seb´e et al.'s protocol the following five functions needed are

(a) SetUp, (b) TagGen, (c) Challenge (d) Gen-Proof
(e) Check-Proof

Let m be the file that will be stored in the untrusted server, which is divided into n blocks of equal lengths: m = $m_1,m_2...m_n$, where n = ⌈|m|/l⌉.
Here checking in a challenge for the encrypted file matrix $en_f$ and m' be the matching factor. Using the function, the verifier can detect the unusual behaving server and if none of the specified rows in the process are deleted or

 

modified, the adversary avoids the detection.
There are three security requirements for the remote data integrity checking protocol:

- Security against the server with public verifiability.
- Privacy against third party verifiers.
- Identifying misbehaviour server.

When the verifier is not the client themselves, the protocol must ensure that no private information about the client's data is leaked to the third party verifier.

A. *Security against Server with Public Verifiability*
If CheckProof(pk,$D_m$, chal,R) = "success", then the remote data integrity checking protocol is said to be secure against the server for any PPT (probabilistic Polynomial time).

B. *Privacy against Third Party Verifiers*
For the remote data integrity checking protocol Π, if there exists a PPT simulator $S_V$ such that $\{S_V(x, f_V(x, y))\}_{x,y\in\{0,1\}^*} \equiv \{\text{view } \Pi_V(x, y)\}_{x,y\in\{0,1\}^*}$, then Π is the protocol that ensures privacy against third- party verifiers,where ≡ denotes computational in distinguishability.

C. *Identification Probability for Misbehaving Serverss*
From user's perspective, the model has to capture all kinds of threats towards the 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, the domain 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. For external attacks, data integrity threats may come from outsiders who are beyond the control domain, 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. Therefore, the proposed model has the capabilities, which captures both external and internal threats towards the cloud data integrity. Suppose n servers are misbehaving due to the possible compromise failure, assume the adversary modifies the data blocks in z rows out of the l rows in the encoded file matrix. Let r be the number of different rows for which the user asks for checking in a challenge. Let X be a discrete random variable that is defined to be the number of rows chosen by the user that matches the rows modified by the adversary. The matching probability that at least one of the rows picked by the user matches one of the rows modified by the adversary is analysed first.

$$Prm' = 1\text{-}P\{X=0\}$$
$$= 1-(\mathbf{1}-\mathbf{min}\{\boldsymbol{r}-\mathbf{1}\boldsymbol{i}=\mathbf{0}\boldsymbol{z}\hat{}\boldsymbol{l}-\boldsymbol{i},1\}) \geq 1-\boldsymbol{lm}-\boldsymbol{zl}\,\boldsymbol{r}$$

If none of the specified r rows in the ith verification process are deleted or modified, the adversary avoids the detection.

This can be achieved by comparing the response values Ri(j) with the pre-stored tokens vi(j) , where j∈{1, . . . , n}. The probability for error localization or identifying misbehaving server(s) is computed in a similar way. It is the product of the matching probability for sampling check and the probability of complementary event for the false negative result..

$$\hat{}Prm' = 1-(\mathbf{1}-\mathbf{min}\{\boldsymbol{r}-\mathbf{1}\boldsymbol{i}=\mathbf{0}\boldsymbol{z}\hat{}\boldsymbol{l}-\boldsymbol{i},1\})$$

The matching probability is where $z\hat{} \leq z$, ^ Prm' matching probability of the modified rows in encoded file matrix. Next, the false negative probability Prf is considered such that Ri(j) = v i (j) when at least one of z^ blocks is modified. When two different data vectors collide, the probability is

$$\hat{}Prf = 2\text{-}p$$

Thus, the identification probability for misbehaving server(s) is

$$\hat{}Pd = \hat{}Prm' \,.\, (1 - \hat{}Prf)$$

where Pd is the detection probability against data modification .
The above formulation for localization of a misbehaving server is integrated to the remote data integrity checking, Seb´e et al.'s protocol, thus making it more efficient and secured protocol. The protocol can be easily extended into a probabilistic one by using the probabilistic framework. The proposed protocol has very good efficiency in the aspects of communication, computation and storage costs.

 

## V. RELATED WORK

Shah et al. [8], [9] propose 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. The auditor verifies both the integrity of the data file and the server's possession of a previously committed decryption key. This scheme only works for encrypted files and it suffers from the auditor statefulness and bounded usage, which may potentially bring in on-line burden to users when the keyed hashes are used up. Ateniese 6] were the first who defined the "provable data possession" (PDP) model for ensuring possession of file on untrusted storages. . Their scheme utilizes the RSA-based homomorphic authenticators for auditing outsourced data and suggests randomly sampling a few blocks of the file. However, the public audit ability in their scheme demands the linear combination of sampled blocks exposed to external auditor. When used directly, their protocol is not provably privacy preserving, and thus may leak user data information to the auditor. In their subsequent work, Ateniese et al. [10] 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 works [11] and [12]. Schwarz et al. [13] proposed the concept which would ensure static file integrity across multiple distributed servers, using erasure-coding and block-level file integrity checks. Some ideas of their distributed storage verification protocol are being adopted. However, the scheme further support data dynamics and explicitly studies the problem of misbehaving server identification, while theirs did not. Zhuo Hao et.al [14] proposed the remote data integrity checking protocol that supports public verifiability without the support of TPA and compared the properties of the proposed protocol with the then existing protocols. Wang et al.[15] in their work proposed a flexible distributed cloud storage integrity auditing mechanism utilizing the homomorphic token and distributed erasure coded data that detects the Byzantine failure, malicious data modification attack and server clouding attacks. All the above schemes provide efficient methods for secured data verifiability, data storage integrity and detection of server attacks in the cloud based storage separately. In this paper the proposed Seb'e et al's protocol combines the mentioned characteristic functions together making it more efficient and secured when compared to other protocols.

## VI. CONCLUSIONS

In this paper, a privacy-preserving protocol for data storage in the cloud is been proposed investigating the data security and integrity in cloud storage. The focus is on stopping data being disclosed by un-trusted service providers when data owners distribute their database entries. To achieve the assurances of cloud data integrity and availability and enforce the quality of dependable cloud storage service for users, an effective and flexible distributed scheme with explicit dynamic data support, including block update, delete and append is being implemented. The future research aims to extend the protocol to support data level dynamics at minimal costs. This includes the extensive security and resilient to the failures like Byzantine failure and other malicious data attacks.

## REFERENCES


[1] P. Mell and T. Grance, "*Draft nist working definition of cloud computing," Referenced on June.3rd,2009*.Online at http:// csrc.nist. gov/groups/SNS/cloud-computing/index.html, 2009.

[2] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, 2006.

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

[4] S.Wilson*,"Application engine outage*,"Online- http://www.cio-weblog.com/50226711/appengine outage.php, June 2008.

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

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L.Kissner, Z. Peterson, and D. Song, "*Provable data possession at untrusted stores," Cryptology ePrint Archive, Report 2007/202, 2007,* http://eprint.iacr.org/.

[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, Saint Malo, France, Sep. 2009.M.*

[8] A. Shah, R. Swaminathan, and M. Baker, "*Privacy- preserving audit and extraction of digital contents," Cryptology ePrint Archive, Report 2008/186, 2008*, http://eprint.iacr.org/.

[9] M. A. Shah, M. Baker, J. C. Mogul, and R.Swaminathan, *"Auditing to keep online storageservices honest," in Proc. of HotOS'07. Berkeley,CA,USA: USENIX Association, 2007, pp. 1–6.*

[10] G.Ateniese, R. D. Pietro, L. V. Mancini, and G.Tsudik, *"Scalable and efficient provable data possession," in Proc. of SecureComm'08, 2008, pp.1–10.*

[11] 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.*

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

[13] 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.

[14] Zhuo Hao, Sheng Zhong, Member, IEEE, and Nenghai Yu, Member, *"A Privacy-Preserving Remote Data Integrity Checking Protocol with Data Dynamics and Public Verifiability", IEEE-2011*

[15] "*Towards Secure and Dependable Storage Services in Cloud Computing*" Cong Wang, Student Member, IEEE, Qian Wang, Student Member, IEEE, Kui Ren, Member, IEEE,Ning Cao, Student Member, IEEE, and Wenjing Lou, Senior Member, IEEE-2011

[16] Wang, Q. Wang, K. Ren, and W. Lou, "*Privacy- preserving public auditing for data storage security in cloud computing," in InfoCom2010, IEEE, March 2010.*

[17] H. Shacham and B. Waters, "Compact proofs of retrievability," in Proc. of ASI-ACRYPT'08. Springer-Verlag, 2008, pp. 90–107.

[18] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, andD. Song, "Provable data possession at untrusted stores," in Proc. of CCS'07. NewYork, NY, USA: ACM, 2007, pp. 598–609.

[19] A. Juels and B. S. Kaliski, Jr., "Pors: proofs of retrievability for large files," inProc. of CCS'07. New York, NY, USA: ACM, 2007, pp. 584–597.

 