# OPoR: Enabling Proof of Retrievability in Cloud Computing with Resource-Constrained Devices

Jin Li, Xiao Tan, Xiaofeng Chen, Duncan S. Wong, Fatos Xhafa

***Abstract*—Cloud Computing moves the application software and databases to the centralized large data centers, where the management of the data and services may not be fully trustworthy. In this work, we study the problem of ensuring the integrity of data storage in Cloud Computing. To reduce the computational cost at user side during the integrity verification of their data, the notion of public verifiability has been proposed. However, the challenge is that the computational burden is too huge for the users with resource-constrained devices to compute the public authentication tags of file blocks. To tackle the challenge, we propose OPoR, a new cloud storage scheme involving a cloud storage server and a cloud audit server, where the latter is assumed to be semi-honest. In particular, we consider the task of allowing the cloud audit server, on behalf of the cloud users, to pre-process the data before uploading to the cloud storage server and later verifying the data integrity. OPoR outsources the heavy computation of the tag generation to the cloud audit server and eliminates the involvement of user in the auditing and in the pre-processing phases. Furthermore, we strengthen the Proof of Retrievabiliy (PoR) model to support dynamic data operations, as well as ensure security against reset attacks launched by the cloud storage server in the upload phase.**

## I. Introduction

Cloud Computing has been envisioned as the next generation architecture of the IT enterprise due to its long list of unprecedented advantages: on-demand self-service, ubiquitous network access, location-independent resource pooling, rapid resource elasticity, and usage-based pricing. In particular, 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.

Jin Li is with the School of Computer Science, Guangzhou University, China, e-mail: (jinli71@gmail.com).

Xiao Tan and Duncan S. Wong are with the Department of Computer Science, City University of Hong Kong, Hong Kong.

Xiaofeng Chen is with the State Key Laboratory of Integrated Service Networks (ISN), Xidian University, Xi'an, China, e-mail: (xfchen@xidian.edu.cn).

Fatos Xhafa is with the Department of Languages and Informatics Systems, Technical University of Catalonia, Barcelona, Spain, email: (fatos@lsi.upc.edu).

Although having appealing advantages as a promising service platform for the Internet, this new data storage paradigm in "Cloud" brings many challenging issues which have profound influence on the usability, reliability, scalability, security, and performance of the overall system. One of the biggest concerns with remote data storage is that of data integrity verification at untrusted servers. For instance, the storage service provider may decide to hide such data loss incidents as the Byzantine failure from the clients to maintain a reputation. What is more serious is that for saving money and storage space the service provider might deliberately discard rarely accessed data files which belong to an ordinary client. Considering 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 verification without the local copy of data files.

In order to overcome this problem, many schemes have been proposed under different system and security models [1]–[10]. In all these works, great efforts have been made to design solutions that meet various requirements: high scheme efficiency, stateless verification, unbounded use of queries and retrievability of data, etc. According to the role of the verifier in the model, all the schemes available fall into two categories: private verifiability and public verifiability. Although achieving higher efficiency, schemes with private verifiability impose computational burden on clients. On the other hand, public verifiability alleviates clients from performing a lot of computation for ensuring the integrity of data storage. To be specific, clients are able to delegate a third party to perform the verification without devotion of their computation resources. In the cloud, the clients may crash unexpectedly or cannot afford the overload of frequent integrity checks. Thus, it seems more rational and practical to equip the verification protocol with public verifiability, which is expected to play a more important role in achieving better efficiency for Cloud Computing.

What's more, there is another major concern among previous designs, that is the support of dynamic data

operation for cloud data storage applications. In Cloud Computing, the remotely stored electronic data might not only be accessed but also be updated by the clients, e.g., through block modification, deletion, insertion etc. Unfortunately, the-state-of-the-art in the context of remote data storage mainly focus on static data files and this dynamic data updates has received limited attention in the data possession applications so far [1]–[3], [9], [11]. Though such problem also has been addressed in [12]–[14], 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 dynamic data operation support 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. In addition, most of existing works adopt weaker security models which do not take into account the reset attack. Specifically, the cloud storage server can trigger reset attacks in the upload phase to violate the soundness of the scheme.

To the best of our knowledge, it seems that no existing scheme can simultaneously provide provable security in the enhanced security model and enjoy desirable efficiency, that is, no scheme can resist reset attacks while supporting efficient public verifiability and dynamic data operations simultaneously.

**Contributions:** Our contribution can be summarized as follows:

- We propose OPoR, a new PoR scheme with two independent cloud servers. Particularly, one server is for auditing and the other for storage of data. The cloud audit server is not required to have high storage capacity. Different from the previous work with auditing server and storage server, the user is relieved from the computation of the tags for files, which is moved and outsourced to the cloud audit server. Furthermore, the cloud audit server also plays the role of auditing for the files remotely stored in the cloud storage server.
- We develop a strengthened security model by considering the reset attack against the storage server in the upload phase of an integrity verification scheme. It is the first PoR model that takes reset attack into account for cloud storage system.
- We present an efficient verification scheme for ensuring remote data integrity in cloud storage. The proposed scheme is proved secure against reset attacks in the strengthened security model while supporting efficient public verifiability and dynamic data operations simultaneously.

### A. Related Work

Recently, much research effort has been devoted largely to ensure the security of cloud computing [15]–[18] and remotely stored data [1]–[3], [9], [11], [19]–[21]. Ateniese *et al.* [1] defined the "provable data possession" (PDP) model for ensuring possession of files on untrusted storages. They also proposed the first proof-of-storage scheme that supports public verifiability. The scheme utilizes RSA-based homomorphic tags for auditing outsourced data, such that a linear combination of file blocks can be aggregated into a single block and verified by employing homomorphic property of RSA. However, the data owner has to compute a large number of tags for those data to be outsourced, which usually involves exponentiation and multiplication operations. Furthermore, The case of dynamic data storage has not been considered by Ateniese *et al.*, and the direct extension of their scheme from static data storage to dynamic case brings many security problems. In their subsequent work [11], Ateniese *et al.* proposed a dynamic version of the prior PDP scheme. However, the system imposes a priori bound on the number of queries and do not support fully dynamic data operations. In [22], Wang et al. considered dynamic data storage in distributed scenario, and the proposed challenge-response protocol can both determine the data correctness and locate possible errors. Similar to [11], they only considered partial support for dynamic data operation. In [21], they also considered how to save storage space by introducing deduplication in cloud storage. Recently, Zhu et al. [19] introduced the provable data possession problem in a cooperative cloud service providers and designed a new remote integrity checking system.

Juels *et al.* [2] introduced a "proof of retrievability" (PoR) model, where spot-checking and error-correcting codes are adopted to ensure both "possession" and "retrievability" of data files in archive service systems. However, public verifiability is not supported in their scheme and the data owner also has to make many computational efforts to generate tags for those data to be outsourced. Shacham *et al.* [3] designed an improved PoR scheme with public verifiability based on BLS signature and the proofs are given in a stronger security model defined in [2]. Similar to the construction in [1], they used publicly verifiable homomorphic authenticators that are built from BLS signatures and proven secure in the random oracle model.

For the first time, Erway *et al.* [23] explored constructions for dynamic provable data possession. They extended the PDP model in [1], [24] to support provable updates to stored data files using rank-based authenticated skip lists. This scheme is essentially a fully dynamic version of the PDP solution. In particular, to support updates, especially for block insertion, they tried

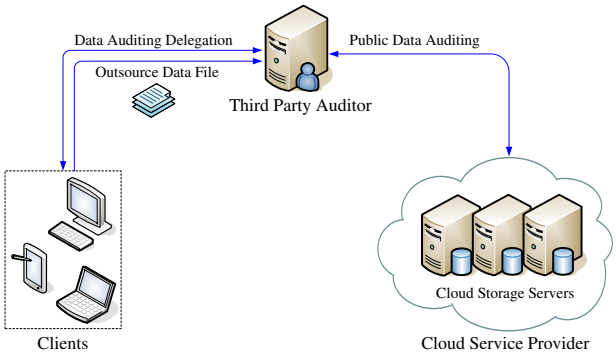


Fig. 1: Cloud data storage architecture

to eliminate the index information in the "tag" computation in Ateniese's PDP model [1]. However, any update on the stored file $F$, even few blocks, will result in the inevitable updates of rank and interval information of all nodes along the path from the updated blocks to the top leftmost node, thus introducing significant computational complexity and losing desirable efficiency. In general, all of the above works do not take the reset attack into account, and impose heavy computation overhead at the client side. In [25], [26], they proposed a new PoR scheme which supports dynamics, however, the users still have to compute all the tags before uploading.

In our solution, we propose an efficient remote data verification scheme simultaneously supporting public verifiability and fully dynamic data operations for PoR systems. As an extension of [27], this paper firstly formally defines the system model and security model for the cloud storage. Different from the previous works, the users are not required to compute the tags for the outsourced data. Thus, the computational overhead at the user side is very low. Furthermore, we also present the detailed security analysis and efficiency analysis for OPoR in this paper under the new security model. In particular, our construction can resist reset attacks triggered by the cloud storage server in the upload phase, and alleviate clients from performing a lot of computation for ensuring the integrity of data storage.

## II. Problem Statement

### A. System Model

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

- Client: an entity that 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 client's data. The CSS is required to provide integrity proof to the clients or cloud audit server during the integrity checking phase.
- Cloud Audit Server (CAS): 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. In this system, the cloud audit server also generates all the tags of the files for the users before uploading to the cloud storage server.

In the cloud paradigm, by putting the large data files on the remote servers, the clients can be relieved of the burden of storage and computation. 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 cloud audit server of their respective choices.

In this paper, we only consider verification schemes with public verifiability: any party in possession of the public key can act as a verifier. We assume that the cloud audit server is unbiased, however, the storage server is untrusted.

### B. Security Model

Shacham and Waters proposed a security model for PoR system in [3]. Generally, the checking scheme is secure if (i) there exists no efficient algorithm that can cheat the verifier with non-negligible probability; (ii) there exists a polynomial-time extractor that can recover the original data file by carrying out multiple challenges-responses. Under the definition of a PoR system, the client periodically challenges the storage server to ensure the correctness of the cloud data and the original files can be recovered by interacting with the server. The definitions of correctness and soundness was given in [3]: 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 that 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 plays the role of the prover (server). In the verification process, the goal of adversary is to cheat the client, 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 prior works. Though some previous works also considered the architecture with two servers, our construction achieves the outsourcing of the tag generation. Thus, the new scheme also requires to prevent the cloud audit server from generating invalid tags for the client's files stored in the cloud storage server. The authentication from the cloud servers is used in the new system to achieve this security requirement. In order to successfully perform the verification while achieving blockless, the server should take over the job of computing. 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 by the client (prover) in each protocol execution other than calculated or pre-stored by the client.

Besides, our PoR model is the first to support dynamic update operations and security against reset attack in a verification scheme. The robustness against reset attack ensures that a malicious storage server can never gain any advantage of passing the verification of an incorrectly stored file by resetting the client (or the audit server) in the upload phase. We will see that most of existing PoR schemes can not ensure this strong security for cloud storage.

### C. Design Goals

Our design goals can be summarized as the following: (1) Public verifiability: to allow anyone, not just the clients originally stored the file, to have the capability to verify correctness of the remotely stored data; (2) Low computation overhead at the client side: to upload data to the cloud server while supporting verifiability, the data owner does not have heavy additional computation; (3) 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; (4) Stateless verification: to eliminate the need for state information maintenance at the verifier side between audits and throughout the long term of data storage. This is also the basic requirement for achieving public verifiability. In particular, we aim to achieve enhanced security against reset attacks in our construction.

## III. Construction of PoR Schemes with Public Verifiability and Dynamic Data Operation Support

We start with some notations and definitions of our scheme, followed by the construction details and discussion of dynamic data operation support.

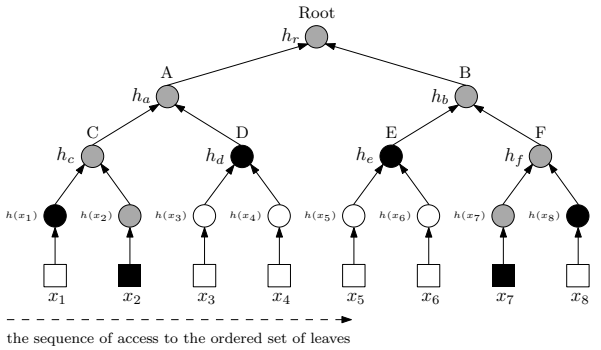


Fig. 2: Merkle hash tree authentication of data elements. The access sequence of leaf nodes $h(x_1), \ldots, h(x_n)$ is defined as the search order from left to right with depth first priority.

### A. Notation and Preliminaries

**Bilinear Map.** A bilinear map is a map $e : G \times G \rightarrow G_T$, where $G$ is a Gap Diffie-Hellman (GDH) group and $G_T$ is another multiplicative cyclic group of prime order $p$ with the following properties [28], [29]: (i) Computable: there exists an efficiently computable algorithm for computing $e$; (ii) Bilinear: for all $h_1, h_2 \in G$ and $a, b \in \mathbb{Z}_p$, $e(h_1^a, h_2^b) = e(h_1, h_2)^{ab}$; (iii) Non-degenerate: $e(g, g) \neq 1$, where $g$ is a generator of $G$.

**Merkle Hash Tree.** A Merkle Hash Tree (MHT) is a well-studied authentication structure [30], which is intended to efficiently 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. Figure 2 depicts an example of authentication. The verifier with the authentic $h_r$ requests 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 authenticate the values of data blocks. However, in this paper we further employ MHT to authenticate both the values and the positions of data blocks. By the way of computing the root in MHT, the leaf node positions can be unique determined through the left-to-right and depth-first sequence.

### B. Definition

In our scheme, both public verifiability and fully dynamic data operation are supported. We now show the definitions and parameters used in our construction.

$(pk, sk) \leftarrow \mathbf{Setup}(1^k)$. *It takes as input security parameter $1^k$, returns public parameters and the key pair of the cloud audit server.*

$(F^*, t) \leftarrow \mathbf{Upload}(sk, F)$. *There are two phases in this algorithm. In the first phase, the client uploads its data file $F$ to the cloud audit server, where $F$ is an ordered collection of blocks $\{M_i\}$. In the second phase, the file $F$ is re-uploaded to the cloud storage server by the cloud audit server: it takes as input the private key $sk$ and $F$, and outputs the signature set $\Phi$, which is an ordered collection of signatures $\{\sigma_i\}$ on $\{M_i\}$. We denote the stored file $F^* = \{F, \Phi\}$. It also outputs metadata-the root $R$ of a Merkle hash tree from $\{M_i\}$ and the signature $t = sig_{sk}(h(R))$ as the tag of $F^*$. Notice that the storage server stores $(F^*, t)$, but the audit server (the client) only keeps $t$ as receipt.*

$1/0 \leftarrow \mathbf{IntegrityVerify}\{P(pk, F^*, t) \rightleftharpoons V(pk, t)\}$. *This is an interactive protocol for integrity verification of a file $F^*$ with tag $t$. The cloud storage server plays the role of prover $P$ with input the public key $pk$, a stored file $F$ and a file tag $t$. The cloud audit server plays the role of verifier $V$ with input $pk$ and $t$. At the end of the protocol, $V$ outputs $TRUE$ $(1)$ if $F^*$ passes the integrity verification, or $FALSE$ $(0)$ otherwise.*

$(F^*, t) \leftarrow \mathbf{Update}\{P(pk, \hat{F}^*, \hat{t}) \rightleftharpoons V(sk, \hat{t}, update)\}$. *This is an interactive protocol for dynamic update of a file $\hat{F}^*$ with tag $\hat{t}$. The cloud storage server plays the role of prover $P$ with input the public key $pk$, a stored file $\hat{F}^*$, and a file tag $\hat{t}$. The cloud audit server plays the role of verifier $V$ with input the private key $sk$, $\hat{t}$, and an data operation request "$update$" from the client. At the end of the protocol, $V$ outputs a file tag $t$ of the updated file $F^*$ if $P$ gives a valid proof for the update, or $FALSE$ $(0)$ otherwise.*

**Correctness**. A PoR scheme is correct if the following two conditions hold:

- If $(F^*, t) \leftarrow \text{Upload}(sk, F)$, then $\text{IntegrityVerify}\{P(pk, F^*, t) \rightleftharpoons V(pk, t)\} = 1$.
- If $(F^*, t) \leftarrow \text{Update}\{P(pk, \hat{F}^*, \hat{t}) \rightleftharpoons V(sk, \hat{t}, update)\}$, then $\text{IntegrityVerify}\{P(pk, F^*, t) \rightleftharpoons V(pk, t)\} = 1$.

*Remarks*. Since the cloud audit server is fully trusted in the two-server architecture, we allow it to generate the key pairs on behalf of the clients in the setup phase. However, it might be undesirable to place full trust on the cloud audit server in some outsourcing tasks. Consider the following scenario: one storage service is available to the clients on a pay-per-use basis, and the audit server may upload a file, intentionally or mistakenly, on behalf of one client who did not ask for storing that file. One solution for such applications is utilizing a proxy signature scheme supporting delegation by warrant [31]–[33] to delegate the signing right of the clients to the cloud audit server for each usage. The warrant to the audit server can be the hashed value of the uploaded file as a credential of the delegation.

### C. The Core Construction

Now we start to present the main idea behind our scheme. As in the previous PoR systems [2], [3], we assume the client encodes the raw data file $\widetilde{F}$ into $F$ using some rate-$\rho$ error correcting codes, e.g. Reed-Solomon codes. To further reduce the computation load of the client, we can require that $\widetilde{F}$ is pre-processed by the cloud audit server. The encoded file $F$ is divided into $n$ blocks $M_1, \ldots, M_n$, and each block has $s$ sectors, i.e. $M_i = (M_{i1}, M_{i2}, \cdots, M_{is})$, where $M_{ij} \in \mathbb{Z}_p$ for $i = 1, \ldots, n$, $j = 1, \ldots, s$, and $p$ is a large prime. Let $e : G \times G \rightarrow G_T$ be a bilinear map, with three cryptographic hash functions $H, h : \{0,1\}^* \rightarrow G$ and $f : \{0,1\}^* \rightarrow \mathbb{Z}_p$, viewed as random oracles [3]. Let $g$ be the generator of $G$. The procedure of our protocol execution is as follows:

**Setup**: The cloud audit server chooses a random $\alpha \leftarrow \mathbb{Z}_p$, $u_1, u_2, \cdots, u_s \leftarrow \mathbb{G}$, and computes $v \leftarrow g^\alpha$. The secret key is $sk = (\alpha)$ and the public key is $pk = (v, \{u_j\}_{1 \le j \le s})$.

**Upload** (**Phase 1: Client$\rightarrow$ Cloud Audit Server**): The client uploads $F = (M_1, \ldots, M_n)$ to the cloud audit server. Given the file $F$, the cloud audit server generates a root $R$ based on the construction of Merkle Hash Tree (MHT), where the leave nodes of the tree are an ordered set of hashes of file blocks $H(M_i)$ $(i = 1, \ldots, n)$. Next, he signs the root $R$ under his private key $\alpha$ as $h(R)^\alpha \leftarrow sig_{sk}(R)$. The file tag $t = sig_{sk}(R)$ is sent back to the client as a receipt.

(**Phase 2: Cloud Audit Server$\rightarrow$ Cloud Storage Server**): The homomorphic authenticators together with metadata are produced as follows: for each block $M_i = (M_{i1}, M_{i2}, \cdots, M_{is})$, the cloud audit server computes a signature $\sigma_i$ as

$$\sigma_i \leftarrow \left( H(M_i) \cdot \prod_{j=1}^{s} u_j^{M_{ij}} \right)^{\alpha}. \quad (1)$$

Denote the set of signatures by $\Phi = \{\sigma_i\}_{1 \le i \le n}$. The cloud audit server sends $F^* = \{F, \Phi\}$ to the cloud storage server. Then, the cloud audit server keeps the receipt $t$ and deletes $F^*$ from its local storage.

**Integrity Verification**: Either the client or the cloud audit server can verify the integrity of the outsourced data by challenging the cloud storage server. To generate the challenge query, the cloud audit server (verifier) picks

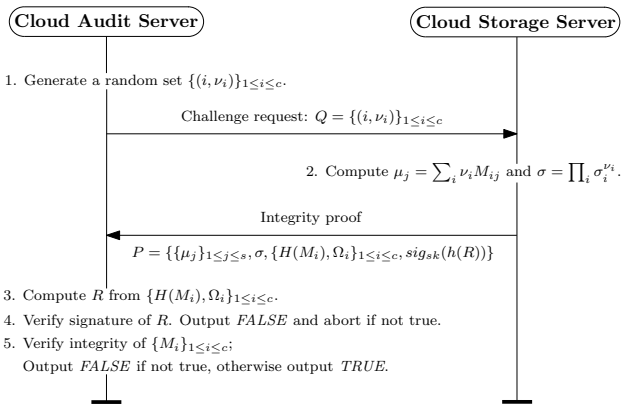


Fig. 3: Protocols for Integrity Verification

a random $c$-element subset $I$ of set $[1,n]$ that denote the positions of the blocks to be checked. For each $i \in I$, picks a random element $\nu_i \leftarrow f(t,i,\tau)$, where $\tau$ denotes the time of query. Let $Q$ be the set $\{(i,\nu_i)\}$, which is sent to the cloud storage server. Upon receiving the challenge query $Q=\{(i,\nu_i)\}_{1\le i\le c}$, the cloud storage server computes

$$\mu_j = \sum_{\{(i,\nu_i)\}\in Q} \nu_i M_{ij} \in \mathbb{Z}_p \quad (2)$$

for $j=1,\ldots,s$, and

$$\sigma = \prod_{\{(i,\nu_i)\}\in Q} \sigma_i^{\nu_i} \in G. \quad (3)$$

In addition, the cloud storage server will also provide the cloud audit server with a small amount of auxiliary information. The auxiliary values are the nodes siblings to the nodes $\{H(M_i)\}_{1\le i\le c}$ to the root $R$. Let $\{\Omega_i\}_{1\le i\le c}$ denote the auxiliary information, the the cloud storage server responds the cloud audit server with proof $P=\{\{\mu_j\}_{1\le j\le s},\sigma,\{H(M_i),\Omega_i\}_{1\le i\le c}\}$. Upon receiving the responses from the cloud storage server, the cloud audit server performs the following computations: (1) generates root $R$ using $\{H(M_i),\Omega_i\}_{1\le i\le c}$ and checks the consistency; (2) checks if $e(t,g)=e(h(R),v)$. (3) checks whether

$$e(\sigma,g) = e\left(\prod_{\{(i,\nu_i)\}\in Q} H(M_i)^{\nu_i}\cdot\prod_{j=1}^{s} u_j^{\mu_j}, v\right) \quad (4)$$

If all the checking holds, output $TRUE$; otherwise, output $FALSE$. The whole protocol procedures are illustrated in Fig. 3.

**Dynamic Update**: In the following, we consider the most general operations involved in dynamic update, that is, data modification, data insertion and data deletion.

- ***Data Modification***: Suppose a client intends to modify the $i$-th block $M_i$ to $M_i'$, then the following procedures have to been performed:
  1) The client sends an *update request* message "$update = (\mathcal{M}, i, M_i')$" to the cloud audit server, where $\mathcal{M}$ denotes the modification operation.
  2) Upon receiving the request, the cloud audit server generates the corresponding signature $\sigma_i' = \left(H(M_i')\cdot\prod_{j=1}^{s} u_j^{M_{ij}'}\right)^{\alpha}$, and sends $update' = (update, \sigma_i')$ to the storage server.
  3) Upon receiving $update'$, the storage server performs the following operations.
     - He replaces the block $M_i$ with $M_i'$ and outputs $F'$.
     - Replaces the $\sigma_i$ with $\sigma_i'$ and outputs $\Phi'$.
     - Replaces $H(M_i)$ with $H(M_i')$ in the Merkle hash tree construction and generates the new root $R'$.
     - For the modification operation, replies the client with a proof $P_{update} = (\Omega_i, H(M_i), R')$, where $\Omega_i$ is the AAI of $M_i$.
  4) After receiving the proof $P_{update}$ from the storage server, the cloud audit server operates as follows.
     - He generates root $R$ using $\{\Omega_i, H(M_i)\}$.
     - Authenticates $R$ by checking if $e(t,g) = e(h(R),v)$.
     - Computes the new root value $\hat{R}$ using $\{\Omega_i, H(M_i')\}$ and checks if $\hat{R}=R'$.
     - Signs the new root metadata $R'$ by $t' = sig_{sk}(R')$ and sends it to the server for storage.
- ***Data Insertion***: Suppose the data owner wants to insert block $M^*$ after the $i$-th block $M_i$. The protocol procedures are similar to the data modification case.
  1) After receiving the proof for insert operation from the storage server, the client first generates root $R$ using $\{\Omega_i, H(M_i)\}$ and authenticates $R$ by checking if $e(t,g)=e(h(R),v)$.
  2) If it is not true, output *FALSE*, otherwise the client can now check whether the server has perform the insertion as required or not, by further computing the new root value using $\{\Omega_i, H(H(M_i)\|H(M^*))\}$ and comparing it with $R'$.
  3) If not, output *FALSE*, otherwise output *TRUE*.
  4) The cloud auditor server signs the new root metadata $R'$ by $sig_{sk}(R')$ and sends it to the server for storage.

- ***Data Deletion***: Data deletion is just the opposite operation of data insertion. For single block deletion, it refers to deleting the specified block and moving all the latter blocks one block forward. Suppose the server receives the $update$ request of deleting block $M_i$, it will delete $M_i$ from its storage space, delete the leaf node $H(M_i)$ in the MHT and generate the new root metadata $R'$. The details of the protocol procedures are similar to those of data modification and insertion, which are thus omitted here.

## IV. Security Analysis

### A. Security Modeling

We analyze the security of our scheme under a variant of Shacham and Waters' PoR model [3] which supports public verifiability and dynamic update operations. Besides, our model offers strengthened security by allowing a malicious storage server to perform reset attack against the client (the cloud audit server) in upload phase.

The basic goal of PoR model is to achieve proof of retrievability. Informally, this property ensures that if an adversary can generate valid integrity proofs of any file $F$ for a non-negligible fraction of challenges, we can construct a PPT machine to extract $F$ with overwhelming probability. It is formally defined by the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$, where $\mathcal{C}$ plays the role of the audit server (the client) and $\mathcal{A}$ plays the role of the storage server:

- **Setup Phase**: The challenger $\mathcal{C}$ runs the Setup algorithm to generate its key pair $(pk, sk)$, and forwards $pk$ to the adversary $\mathcal{A}$.
- **Upload Phase**: $\mathcal{C}$ initiates an empty table called R-list. $\mathcal{A}$ can adaptively query an upload oracle with reset capability as follows:
  - ***Upload***: When a query on a file $F$ and a state index $i$ comes, $\mathcal{C}$ checks if there is an entry $(i, r_i)$ in the R-list. If the answer is yes, $\mathcal{C}$ overwrites $r_i$ onto its random tape; otherwise, $\mathcal{C}$ inserts $(i, r_i)$ into R-list where $r_i$ is the content on its random tape. Then $\mathcal{C}$ runs $(F^*, t) \leftarrow \text{Upload}(sk, F; r_i)$, and returns the stored file $F^*$ and the file tag $t$. Here $Upload(\cdot\,; r_i)$ denotes an execution of the upload algorithm using randomness $r_i$.
- **Challenge Phase**: $\mathcal{A}$ can adaptively make the following two kinds of oracle queries:
  - ***IntegrityVerify***: When a query on a file tag $t$ comes, $\mathcal{C}$ runs the integrity verification protocol IntegrityVerify$\{\mathcal{A} \rightleftharpoons \mathcal{C}(pk, t)\}$ with $\mathcal{A}$.
  - ***Update***: When a query on a file tag $\hat{t}$ and a data operation request "$update$" comes, $\mathcal{C}$ runs the update protocol Update$\{\mathcal{A} \rightleftharpoons \mathcal{C}(sk, \hat{t}, update)\}$ with $\mathcal{A}$.
- **Output Phase**: $\mathcal{A}$ outputs a file tag $t$ and the description of a prover $P_t$.

We say that a prover $P_t$ on $t$ is $\beta$-admissible, if the following two conditions hold:

(1) $t$ is a file tag output by a previous upload query.
(2) Pr[IntegrityVerify$\{P_t \rightleftharpoons \mathcal{C}(pk, t)\} = 1] \geq \beta$.

Then we can define the soundness of PoR scheme.

*Definition 1:* (**Proof of Retrievability**) A PoR scheme is $(\beta, \gamma)$-sound if for any $\beta$-admissible prover $P_t$ output by $\mathcal{A}$ in the above game, there exists an extractor $\mathcal{E}$ that can recover the original file of tag $t$ with probability at least $1 - \gamma$.

Since our model considers reset attack in upload phase, it provides higher security for PoR schemes. To prove this result, we show that Shacham and Waters' PoR scheme, which is proven secure in previous PoR model, is insecure in the new model.

*Theorem 1:* Shacham and Waters' PoR scheme is not sound in our new model.

*Proof:* In Shacham and Waters' PoR scheme, the tag $t$ of an uploaded file $F$ is computed as $t_0 \| SSig_{ssk}(t_0)$. Here $t_0 = name \| n \| u_1 \| \ldots \| u_s$, where $name \xleftarrow{R} \mathbb{Z}_p$, $n$ is the number of blocks in $F$, $u_1, \ldots, u_s \xleftarrow{R} G$; $SSig_{ssk}(\cdot)$ is the signing algorithm of a signature scheme, where the signing key $ssk$ is included in $sk$.

We construct an adversary $\mathcal{A}$ that wins the soundness game with non-negligible probability as follows:

- **Upload Phase**: $\mathcal{A}$ chooses two different files of $n$ blocks, say $F = \{M_1, \ldots, M_n\}$ and $\hat{F} = \{M'_1, \ldots, M'_n\}$. Then $\mathcal{A}$ makes an Upload query on $(F, 1)$, gets $(F^*, t)$, but stores $t$ only. Finally $\mathcal{A}$ makes an Upload query on $(\hat{F}, 1)$, gets and stores $(\hat{F}^*, \hat{t})$ honestly.
- **Output Phase**: $\mathcal{A}$ outputs $t$ and a prover $P_t = P(pk, \hat{F}^*, \hat{t})$, where $P$ is the prover algorithm of an honest storage server.

$\mathcal{A}$ does nothing in Setup Phase and Challenge Phase, so we skip these two phases above. Notice the following two observations:

(1) The tag generation only depends on the randomness used for choosing $name, u_1, \ldots, u_s$ and the number of file blocks $n$. The uploading queries on $(F, 1)$ and $(\hat{F}, 1)$ use the same randomness $r_1$ (an entry $(1, r_1)$ will be inserted into R-list after querying $(F, 1)$ to the upload oracle). Besides, $F$ and $\hat{F}^*$ both have $n$ blocks. Therefore, we have $\hat{t} = t$.
(2) $(\hat{F}^*, \hat{t})$ is the honestly stored file of $\hat{F}$, so $P_t$ is a valid proof of $\hat{F}$ with tag $\hat{t}$, i.e. IntegrityVerify$\{P(pk, \hat{F}^*, \hat{t}) \rightleftharpoons \mathcal{C}(pk, \hat{t})\} = 1$.

From (1) and (2), we have IntegrityVerify$\{P_t \rightleftharpoons \mathcal{C}(pk, t)\} = 1$ with probability $\beta = 1$, by correctness

of the scheme. That is, $\mathcal{A}$ outputs a 1-admissible prover $P_t$ on the file tag $t$.

Since that all the information $\mathcal{A}$ stores are $\hat{F}^*$ and $\hat{t}$ (where $t = \hat{t}$), only $\hat{F}$ can be extracted from $\mathcal{A}$'s storage. However, the original file of tag $t$ is $F \neq \hat{F}$. By the knowledge of $\hat{F}$, we can know nothing about $F$ but that $F$ has $n$ blocks. Therefore, given $P_t = P(pk, \hat{F}^*, \hat{t})$, it is impossible to construct a $(1, \gamma)$-extractor of the file $F$ (even when assume that the extractor has unlimited computing power) for any $\gamma$. This completes the proof. ■

### B. Security Proofs

The security of our PoR scheme proposed in Section 3.3 assumes the hardness of the Computational Diffie-Hellman problem.

*Definition 2:* (**CDH Problem**) The Computational Diffie-Hellman problem is that, given $g$, $g^x, g^y \in \mathbb{G}_1$ for unknown $x, y \in \mathbb{Z}_p^*$, to compute $g^{xy}$.

We say that the $(t, \epsilon)$-CDH assumption holds in $\mathbb{G}_1$ if no $t$-time algorithm has the non-negligible probability $\epsilon$ in solving the CDH problem.

*Theorem 2:* If the computational Diffie-Hellman problem is hard in bilinear groups, no adversary against the soundness of our public-verifiable PoR scheme could cause the verifier to accept an integrity proof of any file $F$ with non-negligible probability in the random oracle model, except by responding with correctly computed values.

*Proof:*

The security of soundness is given by reduction. We assume that there is an adversary who can break the soundness. Another simulator will be constructed by interacting with the adversary. The simulator also answers all the queries for PoR protocol, including the tag generation and integrity proof. After the simulation, if the adversary outputs a valid tag without the help of client, the simulator breaks the assumption of computational Diffie-Hellman problem. Suppose that an adversary outputs the description of a prover that causes the verifier to accept an integrity proof with non-negligible probability, by responding with values that are not correctly computed. Let $F = (M_1, \ldots, M_n)$ be the file for integrity verification, $\Phi = \{\sigma_i\}_{1 \leq i \leq n}$ be the signatures of file blocks, $Q = \{(i, \nu_i)\}_{1 \leq i \leq c}$ be the verification query. Denote the expected response from a honest prover by $P = \{\{\mu_j\}_{1 \leq j \leq s}, \sigma, \{H(M_i), \Omega_i\}_{1 \leq i \leq c}\}$, and denote the proof generated by the adversary be $P' = \{\{\mu'_j\}_{1 \leq j \leq s}, \sigma', \{H(M'_i), \Omega'_i\}_{1 \leq i \leq c}\}$ where $P' \neq P$.

First, we show that $\{H(M'_i), \Omega'_i\}_{1 \leq i \leq c} = \{H(M_i), \Omega_i\}_{1 \leq i \leq c}$ if the hash functions $H$ and $h$ are collision resistent. Denote by $R'$ the Merkle Hash Tree root generated from $\{H(M'_i), \Omega'_i\}$, and denote by $R$ the MHT root $R$ generated from $\{H(M_i), \Omega_i\}$. Suppose $\{H(M'_i), \Omega'_i\}_{1 \leq i \leq c} \neq \{H(M_i), \Omega_i\}_{1 \leq i \leq c}$, there must be $R \neq R'$ except with negligible probability due to the collision resistance of $H$. The signature on $R'$ can pass the verification, so (1) $e(t, g) = e(h(R'), v)$. Since $t = h(R)^\alpha$, we have (2) $e(t, g) = e(h(R), v)$. From (1) and (2), $h(R') = h(R)$, implying a collision $(R, R')$ of $h$ which occurs with negligible probability. So the adversary outputs the correct hashes of file blocks $\{H(M_i), \Omega_i\}_{1 \leq i \leq c}$ in the proof $P'$ except with probability $1 - (1 - 1/p)^{\lg n + 1}$, when $H$ and $h$ are modeled as random oracles.

Then we show that $\sigma' = \sigma$ if the computational Diffie-Hellman problem is hard. Otherwise, we construct a simulator, that given $g, \tilde{g} = g^\alpha, h \in \mathbb{G}$ where $\alpha$ is unknown, outputs $h^\alpha$. In the setup phase, the simulator sets $v$ as $\tilde{g}$, chooses two vectors of randomness $\beta_1, \ldots, \beta_s \in \mathbb{Z}_p$ and $\gamma_1, \ldots, \gamma_s \in \mathbb{Z}_p$, and sets $u_j = g^{\beta_j} h^{\gamma_j}$ for $j = 1, \ldots, s$. It initiates three empty hash tables $H$-table, $h$-table, $f$-table, and simulates the oracle queries as follows:

- *$H$ oracle*: When a query of $M_i$ comes where there exists $(M_i, r_i)$ in the $H$-table for some $r_i$, returns $r_i$. The $h$ oracle and $f$ oracle process such queries similarly. When a query of a fresh $M_i$ comes, performs the following:
  (1) Randomly chooses $r_i \in \mathbb{Z}_p$.
  (2) If there exists $(M', r_i)$ in the $H$-table, go to step (1).
  (3) Inserts $(M_i, r_i)$ in the $H$-table and returns $H(M_i) = g^{r_i} / (g^{\sum_{j=1}^{s} \beta_j M_{ij}} h^{\sum_{j=1}^{s} \gamma_j M_{ij}})$.
- *$h$ oracle*: when a oracle query of a fresh $R$ comes, performs the following:
  (1) Randomly chooses $r \in \mathbb{Z}_p$.
  (2) If there exists $(R', r)$ in the $h$-table, go to step (1).
  (3) Inserts $(R, r)$ in the $H$-table and returns $h(R) = g^r$.

  The step (2) in the simulation of $H$ oracle and $h$ oracle ensures that $H$, $h$ are collision resistant, and the reset attack by uploading different files using the same randomness will not work.
- *$f$ oracle*: when a oracle query of a fresh $\xi$ comes, performs the following:
  (1) Randomly chooses $\nu \in \mathbb{Z}_p$.
  (2) If there exists $(\xi, \nu)$ in the $f$-table, go to step (1).
  (3) Inserts $(\xi, \nu)$ in the $f$-table, and returns $f(\xi) = \nu$.
- *Upload Oracle*: when a oracle query of $F = \{M_1, \cdots, M_n\}$ and a state index $i$ comes, sets the random tape as the challenger does in the game defined in Section 4.1, and performs the following computation:
  (1) Query $M_i$ to $H$ oracle for $i = 1, \cdots, n$;

(2) Construct MHT root $R$ from $\{M_i\}$, and query $R$ to $h$ oracle;
(3) Compute $\Phi = \{\sigma_i\}_{1\leq i\leq n}$ where $\sigma_i = (H(M_i)\cdot\prod_{j=1}^{s} u_j^{M_{ij}})^{\alpha} = (g^{r_i})^{\alpha} = (g^{\alpha})^{r_i} = \tilde{g}^{r_i}$ and $(M_i, r_i)$ is an entry in $H$-table;
(4) Compute $t = h(R)^{\alpha} = g^{r\alpha} = \tilde{g}^{r}$ where $(R, r)$ is an entry in $h$-table;
(5) Return $F^* = (F, \Phi)$ and $t$.

- *Integrity Verify Oracle*: when a oracle query of a file tag $t$, starts an execution of IntegiryVerify protocol with the adversary, and performs as an honest verifier.
- *Update Oracle*: when a oracle query of a file tag $t$ for an operation "$update$" comes, the simulation is similar to Upload oracle. For instance, when $update = (\mathcal{M}, i, M_i')$, the simulator computes $\sigma_i'$ and produces the new file tag $t'$ similarly as answering an Upload query.

Eventually the adversary outputs $\sigma' \neq \sigma$, such that:

$$e(\sigma', g) = e\left(\prod_{\{(i,\nu_i)\}\in Q} H(M_i)^{\nu_i}\cdot\prod_{j=1}^{s} u_j^{\mu_j'}, v\right). \quad (5)$$

Combining with the equation 4, we have:

$$\begin{aligned} e(\sigma'/\sigma, g) &= e\left(\prod_{j=1}^{s} u_j^{\mu_j'-\mu_j}, v\right) \\ &= e\left(\prod_{j=1}^{s} (g^{\beta_j}h^{\gamma_j})^{\mu_j'-\mu_j}, v\right). \quad (6)\end{aligned}$$

We denote $\Delta\mu_j = \mu_j' - \mu_j$ where at least one $\Delta\mu_j \neq 0$, then above equation yields:

$$e(h, v)^{\sum_{j=1}^{s}\gamma_j\Delta\mu_j} = e(\sigma'\sigma^{-1}v^{-\sum_{j=1}^{s}\beta_j\Delta\mu_j}, g). \quad (7)$$

The simulator computes $h^{\alpha} = (\sigma'\sigma^{-1}v^{-\sum_{j=1}^{s}\beta_j\Delta\mu_j})^{\frac{1}{\sum_{j=1}^{s}\gamma_j\Delta\mu_j}}$ as the solution to the given CDH instance. Since that at least one $\Delta\mu_j \neq 0$ and the random values $\gamma_j$ are information theoretically hiding, the probability that $\sum_{j=1}^{s}\gamma_j\Delta\mu_j = 0$ is about $1/p$.

Now assume that $\sigma' = \sigma$ in the proof $P'$, we show that $\{\mu_j'\} = \{\mu_j\}$ for $j = 1, \ldots, s$ if the discrete logarithm problem is hard. Otherwise, we construct a simulator, that given $g, h \in \mathbb{G}$, outputs $x$ such that $h = g^x$, by acting as the verifier to play with the adversary. The simulation for the oracle queries is similar as above except that $v = g^{\alpha}$ is honestly generated in the Setup phase, such that the simulator can produce signatures of file blocks by itself with the knowledge of $\alpha$ to answer an Upload query.

Eventually the adversary outputs $\{\mu_j'\}$ for $j = 1, \ldots, s$, such that at least one of them is not equal to the corresponding value in $\{\mu_j\}$. Since that $e(\sigma', g) = e(\sigma, g)$, the following equation holds:

$$\begin{aligned} & e\left(\prod_{\{(i,\nu_i)\}\in Q} H(M_i)^{\nu_i}\cdot\prod_{j=1}^{s} u_j^{\mu_j'}, v\right) \\ = \; & e\left(\prod_{\{(i,\nu_i)\}\in Q} H(M_i)^{\nu_i}\cdot\prod_{j=1}^{s} u_j^{\mu_j}, v\right). \quad (8)\end{aligned}$$

So we have $\prod_{j=1}^{s} u_j^{\mu_j'} = \prod_{j=1}^{s} u_j^{\mu_j}$, therefore:

$$\begin{aligned} \prod_{j=1}^{s} u_j^{\Delta\mu_j} &= \prod_{j=1}^{s}(g^{\beta_j}h^{\gamma_j})^{\Delta\mu_j} \\ &= g^{\sum_{j=1}^{s}\beta_j\Delta\mu_j}h^{\sum_{j=1}^{s}\gamma_j\Delta\mu_j} = 1. \quad (9)\end{aligned}$$

The simulator computes $x = -\frac{\sum_{j=1}^{s}\beta_j\Delta\mu_j}{\sum_{j=1}^{s}\gamma_j\Delta\mu_j}$ as the solution to the given DL instance. The simulation fails when $\sum_{j=1}^{s}\gamma_j\Delta\mu_j = 0$ with probability about $1/p$.

In conclusion, if the proof $P' \neq P$ causes the verifier to accept, we can either break the computational Diffie-Hellman assumption, or the discrete logarithm assumption, or collision resistance of the hash functions $H$ or $h$. Notice that the hardness of discrete logarithm assumption is implied by the computational Diffie-Hellman assumption, and $H$, $h$ are modeled as random oracles that are inherently collision resistant. Hence we complete the proof for this theorem in the random oracle model based on the computational Diffie-Hellman assumption. ■

Theorem 1 ensures that all the provers winning the adversarial game are *well-behaved*, i.e. any valid proof must be exactly the same as the proof computed by an honest storage server except with a negligible probability.

*Theorem 3:* For any well behaved $\beta$-admissible prover $P_t$ on a $n$-block file of tag $t$, there exists an extractor that can recover the original file in time $O(n^2(s+1) + (1+\beta n^2)n/\omega)$ by running $O(n/\omega)$ interactions with $P_t$, where $\omega = \beta - 1/p - (\rho n/n - c + 1)^c$. Following Theorem 1, we can construct an extractor to interact with the prover $P_t$ and get correctly computed proofs for $\beta$-fraction of the verification queries in the query space $\mathbb{Z}_p$. Applying combinatorial techniques, a $\rho$-fraction of encoded file blocks are retrievable after at most $O(n/\omega)$ interactions. Then these file blocks suffice for recovering the original file using the decoding procedure of rate-$\rho$ error correcting codes. Since that our modeling for integrity verification protocol is the same as in Shacham and Waters' PoR model, the simulation for extracting the original file is similar to that in [3], so we omit the details of the proof here.

The following theorem is straightly forward combining Theorem 1 and Theorem 2.

*Theorem 4:* The proposed PoR scheme is $(\beta, \gamma)$-sound for any $\beta$-admissible prover $P_t$ output by $\mathcal{A}$ in the soundness game, where $\gamma = 1 - (1 - 1/p)^{lgn+1} + 1/p$.

## V. Performance Analysis

In this section, we will provide a thorough experimental evaluation of the construction proposed. We build our testbed by using 64-bit M2 high-memory quadruple extra large Linux servers in Amazon EC2 platform as the auditing server and storage server, and a Linux machine with Intel(R) Core(TM)2 Duo CPU clocked at 2.40 GHz and 2 GB of system memory as the user.

In order to achieve $\lambda = 80$ bit security, the prime order $p$ of the GDH group $G$ of the bilinear mapping should be 160 bits in length. Note that in all the evaluations, the groups $\mathbb{G}$ and $\mathbb{G}_T$ are selected in 160-bit and 512-bit length respectively. Suppose there is a 4 GB file with block size 4 KB, then it has $n = 1000000$ blocks and $s = 25$ sectors each block. When it is uploaded onto the storage server, the set of signatures on the file blocks only requires for an additional storage of 20 MB for data integrity verification.

For the integrity verification protocol, the query $Q = \{(i, \nu_i)\}_{1 \le i \le c}$ is $c(\lg n + 2\lambda)$ bits long. However, in the random oracle model, we can use a seed of $2\lambda$ bits to replace the $c$-element query, and the storage server can use the hash oracle to generate the full query after receiving the seed from the client. When $\lambda = 80$, the query length is only 20 bytes. The response $P = \{\{\mu_j\}_{1 \le j \le s}, \sigma, \{H(M_i), \Omega_i\}_{1 \le i \le c}\}$ is $2\lambda(c\lg n + c + s + 1)$ bits long (34 KB for the 4 GB file). We can see that the communication cost grows almost linearly as the block size increases, this is mainly caused by the increasing in size of the verification block. In the protocol of dynamic data operation, the request $update'$ from the audit server is $2\lambda(s+1) + \lg n$ bits (540 bytes for the 4 GB file), and the response from the storage server is $P_{update} = 2\lambda(\lg n + 2)$ bits long (440 bytes for the 4 GB file).

Moreover, we evaluate the communication cost for each user's communication cost in Amazon EC2 cloud environment, which is 92 ms. Note that such an overhead includes the time consuming for transmission and authentication at Amazon EC2 cloud platform.

In our experiment, we use $\rho$ to denote the various erasure code rate while maintaining high detection probability of file corruption. In our schemes, rate $\rho$ denotes that any $\rho$-fraction of the blocks suffices for file recovery. According to [1], if $t$ fraction of the file is corrupted, by asking proof for a constant $c$ blocks of the file, the verifier can detect this server misbehavior with probability $1 - (1 - t)^c$. When $t$ is set to be $1 - \rho$, the probability could be $(1 - \rho)^c$. Similar to [1], 460 blocks are enough for the integrity verification algorithm.

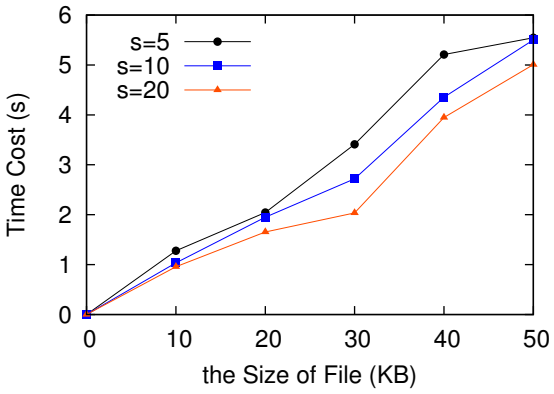


Fig. 4: Tag generation time

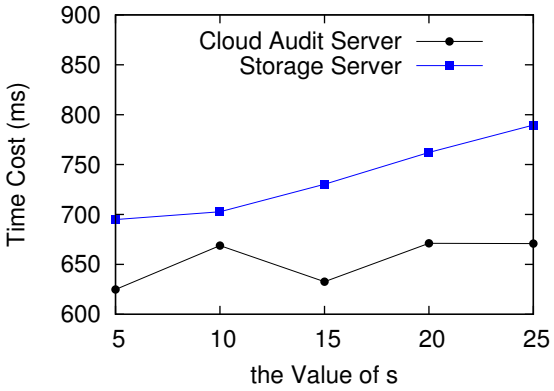


Fig. 5: Verification time

In the first experiment, the computational overhead for the tag generation of files at the cloud audit server is evaluated. We have not checked the computational overhead at users because it only needs the computation of a digital signature, which is very small compared with the computation of the tags. The reason is that the most overhead computation has been delivered to the cloud audit server. Three different numbers of $s$ are chosen in the experiment to show the effect on the efficiency of the time cost. From Fig. 4, we can see that the time cost grows when the number of $s$ decreases. The average time cost for file with size 50KB is $5s$. Compared with the previous related work [12], [25], the computational overhead at users in [12], [25] is outsourced to the cloud audit server.

The response time at the user side including the time cost of uploading files, tag generation at the audit server, construction of Merkle-hash tree, the communication cost and signature generation. To evaluate the response time, the cost of uploading file to the cloud audit server is tested. For file with $10MB$, the average time cost is $25s$. The time cost of construction for the Merkle-hash tree is $27s$ for the file with size $10MB$. The

signature generation for the root is $3ms$, which can be omitted compared with the time cost of uploading and construction of Merkle-hash tree. Note that the time cost of uploading files cannot be avoided in any applications. Though the tag generation cost is also close to the time cost of uploading files, the cloud audit server can process these tag generation during the file uploading. Thus, the additional time for the response is very small. This is acceptable for the users because the time cost would be double at the user side if the tag is computed by the users.

We further evaluate the performance for verification at both cloud audit server and cloud storage server in a scalable system in Fig. 5. Obviously, as the growth of the number of $s$ in system, the time cost for response value at cloud storage server is increasing. This is because it needs to compute all the exponentiations for each block in a tag. Whereas, such cost at cloud audit server is almost constant (nearly $650\ ms$) because $460$ blocks are enough for the integrity verification no matter what is the size of file to be checked.

## VI. Conclusion

This paper proposes OPoR, a new proof of retrievability for cloud storage, in which a trustworthy audit server is introduced to preprocess and upload the data on behalf of the clients. In OPoR, the computation overhead for tag generation on the client side is reduced significantly. The cloud audit server also performs the data integrity verification or updating the outsourced data upon the clients' request. Besides, we construct another new PoR scheme proven secure under a PoR model with enhanced security against reset attack in the upload phase. The scheme also supports public verifiability and dynamic data operation simultaneously.

There are several interesting topics to do along this research line. For instance, we can (1) reduce the trust on the cloud audit server for more generic applications, (2) strengthen the security model against reset attacks in the data integrity verification protocol, and (3) find more efficient constructions requiring for less storage and communication cost. We leave the study of these problems as our future work.

## References

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 598–609.

[2] A. Juels and B. S. K. Jr., "Pors: proofs of retrievability for large files," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 584–597.

[3] H. Shacham and B. Waters, "Compact proofs of retrievability," in *ASIACRYPT '08: Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 90–107.

[4] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," in *Proceedings of CCSW 2009*. ACM, 2009, pp. 43–54.

[5] M. Naor and G. N. Rothblum, "The complexity of online memory checking," *J. ACM*, vol. 56, no. 1, pp. 2:1–2:46, Feb. 2009. [Online]. Available: http://doi.acm.org/10.1145/1462153.1462155

[6] E.-C. Chang and J. Xu, "Remote integrity check with dishonest storage server," in *Proceedings of ESORICS 2008, volume 5283 of LNCS*. Springer-Verlag, 2008, pp. 223–237.

[7] M. 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/.

[8] A. Oprea, M. K. Reiter, and K. Yang, "Space-efficient block storage integrity," in *In Proc. of NDSS 2005*, 2005.

[9] T. S. J. Schwarz and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2006.

[10] Q. Wang, K. Ren, S. Yu, and W. Lou, "Dependable and secure sensor data storage with dynamic integrity assurance," *ACM Transactions on Sensor Networks*, vol. 8, no. 1, pp. 9:1–9:24, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/1993042.1993051

[11] L. V. M. Giuseppe Ateniese, Roberto Di Pietro and G. Tsudik, "Scalable and efficient provable data possession," in *International Conference on Security and Privacy in Communication Networks (SecureComm 2008)*, 2008.

[12] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *INFOCOM*, 2010, pp. 525–533.

[13] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Dynamic audit services for integrity verification of outsourced storages in clouds," in *SAC*, 2011, pp. 1550–1557.

[14] Q. Zheng and S. Xu, "Fair and dynamic proofs of retrievability," in *CODASPY*, 2011, pp. 237–248.

[15] J. Li, X. Chen, J. Li, C. Jia, J. Ma, and W. Lou, "Fine-grained access control system based on attribute-based encryption," *ESORICS*, 2013.

[16] J. Li and K. Kim, "Hidden attribute-based signatures without anonymity revocation," *Information Sciences*, vol. 180, no. 9, pp. 1681–1689, 2010.

[17] J. Li, C. Jia, J. Li, and X. Chen, "Outsourcing encryption of attribute-based encryption with mapreduce," *ICICS*, 2012.

[18] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms of outsourcing modular exponentiations," *ESORICS*, pp. 541–556, 2012.

[19] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 12, pp. 2231–2244, 2012.

[20] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen, "Towards end-to-end secure content storage and delivery with public cloud," in *CODASPY*, 2012, pp. 257–266.

[21] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," in *CODASPY*, 2012, pp. 1–12.

[22] C. Wang, Q. Wang, and K. Ren, "Ensuring data storage security in cloud computing," in *Proceedings of IWQoS 2009*, Charleston, South Carolina, USA, 2009.

[23] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," Cryptology ePrint Archive, Report 2008/432, 2008, http://eprint.iacr.org/.

[24] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, "Outsourcing large matrix inversion computation to a public cloud,," in *IEEE Transactions on Cloud Computing*, 2013, pp. vol. 1, no. 1.

[25] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *ESORICS*, 2009, pp. 355–370.
[26] H. Li, B. Wang, and B. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," in *IEEE Transactions on Cloud Computing*, 2014, pp. vol. 2, no. 1, 43–56.
[27] J. Li, X. Tan, X. Chen, and D. S. Wong, "An efficient proof of retrievability with public auditing in cloud computing," in *INCoS*, 2013, pp. 93–98.
[28] D. Boneh and C. Gentry, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proceedings of Eurocrypt 2003, volume 2656 of LNCS*. Springer-Verlag, 2003, pp. 416–432.
[29] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2001, pp. 514–532.
[30] R. C. Merkle, "Protocols for public key cryptosystems," in *Proceedings of IEEE Symposium on Security and Privacy 1980*. IEEE, 1980, pp. 122–133.
[31] Z. Liu, Y. Hu, X. Zhang, and H. Ma, "Provably secure multi-proxy signature scheme with revocation in the standard model," *Computer Communications*, vol. 34, no. 3, pp. 494–501, 2011.
[32] T. Malkin, S. Obana, and M. Yung, "The hierarchy of key evolving signatures and a characterization of proxy signatures," in *Proceedings of Eurocrypt 2004, volume 3027 of LNCS*. Springer-Verlag, 2004, pp. 306–322.
[33] J. C. Schuldt, K. Matsuura, and K. G. Paterson, "Proxy signatures secure against proxy key exposure," in *Proceedings of PKC 2008, volume 4939 of LNCS*. Springer-Verlag, 2008, pp. 141–161.

**Jin Li** received his B.S. (2002) in Mathematics from Southwest University. He got his Ph.D degree in information security from Sun Yat-sen University at 2007. Currently, he works at Guangzhou University as a professor. He has been selected as one of science and technology new star in Guangdong province. His research interests include Applied Cryptography and Security in Cloud Computing. He has published over 50 research papers in refereed international conferences and journals and has served as the program chair or program committee member in many international conferences.

**Xiao Tan** received his B.S. and M.S. degrees from Fudan University in 2007 and 2010, respectively, and his Ph.D. degree from City University of Hong Kong, in 2014. He had served as a senior research associate at the Department of Computer Science, City University of Hong Kong, during Oct 2013 - Jun 2014. His main research interests include cryptography and information security, in particular, far exchange protocols, cloud storage systems, digital signature and encryption schemes.

**Xiaofeng Chen** received his B.S. and M.S. on Mathematics in Northwest University, China. He got his Ph.D degree in Cryptography from Xidian University at 2003. Currently, he works at Xidian University as a professor. His research interests include applied cryptography and cloud computing security. He has published over 80 research papers in refereed international conferences and journals. His work has been cited more than 1000 times at Google Scholar. He has served as the program/general chair or program committee member in over 20 international conferences.

**Duncan S. Wong** received the B.Eng. degree from the University of Hong Kong in 1994, the M.Phil. degree from the Chinese University of Hong Kong in 1998, and the Ph.D. degree from Northeastern University, Boston, MA, U.S.A. in 2002. He is currently an associate professor in the Department of Computer Science at the City University of Hong Kong. His primary research interest is cryptography; in particular, cryptographic protocols, encryption and signature schemes, and anonymous systems. He is also interested in other topics in information security, such as network security, wireless security, database security, and security in cloud computing.

**Fatos Xhafa** holds a PhD in Computer Science from the Department of Languages and Informatics Systems (LSI) of the Technical University of Catalonia (UPC), Barcelona, Spain. Currently he holds a permanent position of *Professor Titular d'Universitat* at LSI/UPC. He was a Visiting Professor at Birkbeck College, University of London (UK) during academic year 2009-2010 and Research Associate at Drexel University, Philadelphia (USA) during academic term 2004/2005. He has widely published in peer reviewed international journals, conferences/workshops, book chapters and edited books and proceedings in the field. He has an extensive editorial and reviewing service and is actively participating in the organization of several international conferences. His research interests include parallel and distributed algorithms, security, optimization, networking and distributed computing.