Efficient Public Integrity Checking for Cloud Data Sharing with Multi-User Modification

Jiawei Yuan
Department of Computer Science
University of Arkansas at Little Rock
Email: jxyuan@ualr.edu

Shucheng Yu
Department of Computer Science
University of Arkansas at Little Rock
Email: sxyu1@ualr.edu

***Abstract*—In past years a body of data integrity checking techniques have been proposed for securing cloud data services. Most of these work assume that only the data owner can modify cloud-stored data. Recently a few attempts started considering more realistic scenarios by allowing multiple cloud users to modify data with integrity assurance. However, these attempts are still far from practical due to the tremendous computational cost on cloud users. Moreover, collusion between misbehaving cloud servers and revoked users is not considered. This paper proposes a novel data integrity checking scheme characterized by multi-user modification, collusion resistance and a constant computational cost of integrity checking for cloud users, thanks to our novel design of polynomial-based authentication tags and proxy tag update techniques. Our scheme also supports public checking and efficient user revocation and is provably secure. Numerical analysis and extensive experimental results show the efficiency and scalability of our proposed scheme.**

I. Introduction

The continuous development of cloud techniques has boosted a number of cloud-based applications. In particular, more and more applications are using cloud as a collaboration platform, in which data is not only persisted in cloud for storage but also subject to frequent modifications from multiple users. Real-world examples are cloud-based development platforms such as Cloud9 IDE [1] and Codeanywhere [2], which enable multiple developers around the world to work in group, accessing, modifying and compiling the same file on cloud servers anywhere anytime. For correct execution of this kind of cloud-based collaborative applications, one problem is to assure data integrity, i.e., each data modification operation is indeed performed by an authorized group member and the data remains intact thereafter. This problem is important given the fact that cloud platforms, even well-known cloud platforms, may experience hardware/software failures, human errors and malicious attacks [3], [4].

In order to ensure the integrity of data stored on remote/cloud servers, a series of solutions have been proposed based on various techniques [5], [6], [7], [8], [9], [10], [11], [12], [13]. However, most of these existing solutions only consider the case of single writer – only the data owner, who holds secret keys, can modify the data. If these solutions are trivially extended to support multiple writers with data integrity assurance, the data owner has to stay online, collecting modified data from other users and regenerating authentication tags for them. Obviously, this kind of trivial extension will introduce a tremendous workload to the data owner, especially in scenarios with a large number of writers (users) and/or a high frequency of data modification operations. To allow multiple users to modify data, Wang et al. [14] proposed a public integrity auditing scheme using ring signature-based homomorphic authenticators. Nevertheless, in ref. [14] user revocation is not considered and the auditing cost is linear to the group size and data size. In order to further enhance the previous works, another attempt was made by Wang et al, [11]. However, ref. [11] still suffers from a non-trivial computational cost which is linear to the group size and the number of checking tasks, and thus is limited in scalability. In addition, user revocation in ref. [11] is based on the assumption that no collusion occurs between cloud servers and revoked user. As a matter of fact, collusion between invalid users and cloud servers in ref. [11] will lead to the disclosure of secrets of all other valid users. To our best knowledge, there still lacks an efficient and publicly verifiable integrity checking scheme for cloud data sharing that supports multiple writers with secure user revocation.

To solve this open problem, the following major (not necessarily complete list of) challenges exist: 1) *efficiency and scalability*. Most of existing integrity checking solutions are not efficient nor scalable while supporting multiple writers. The main difficulty is attributed to two facets - on one hand, each data block (assuming data integrity is verified per block) shall have its own authentication tag; on the other hand, each writer (user) needs to generate a separate authentication tag, which is necessary for integrity checking, for each data block that he/she has modified. Consequently, the communication/computational complexity for integrity checking will grow linearly with the number of writers (users) and/or the number of data blocks being checked. 2) *efficient and secure user revocation*. User revocation is a challenging issue in most security systems and usually involves disabling user secret keys. For efficient user revocation, one technique is to delegate the task to the cloud by disclosing partial secret to cloud servers. This method, however, leaves it a challenging issue to detect misbehaving cloud servers who collude with unauthorized users. 3) *public checking*. In practical systems, data integrity check can be performed not only by data owners or writers but also by a third-party auditor or any general user who need to access the data.

In this work, we overcome the above challenges and propose an efficient public integrity checking scheme for cloud data sharing that supports multiple writers. Our proposed scheme is featured by public integrity checking and constant computational cost on user side, thanks to our novel design

on polynomial-based authentication tags which allows aggregation of tags of different data blocks. For system scalability, we further empower the cloud with the ability to aggregation authentication tags from multiple writers into one when sending the integrity proof information to the verifier (who may be general cloud users). As a result, just a constant size of integrity proof information needs to be transmitted to the verifier no matter how many data blocks are being checked and how many writers are associated with the data blocks. Moreover, our novel design allows secure delegation of user revocation operations to the cloud even if cloud servers collude with malicious users. Last but not least, our proposed scheme allows aggregation of integrity checking operations for multiple tasks (files) through our batch integrity checking technique. We formally prove the security of our design based on the Computational Diffie-Hellman (CDH) problem, the Bilinear Diffie-Hellman (BDH) problem and the Strong Diffie-Hellman (SDH) problem. Thorough analysis and extensive experimental results on Amazon EC2 demonstrate the scalability and efficiency of our design. Our contributions are summarized as follows,

- Our proposed scheme is among the first that achieves public integrity checking for cloud data sharing supporting multiple writers. Different from existing solutions, our scheme is efficient, highly scalable, and collusion resistant.
- Our novel design of authentication tags can serve as an independent tool that can easily find application in other related scenarios such as verifiable keyword search.
- We fully implemented our scheme and evaluated its performance through both numerical analysis and experimental results. The security of our design is formally proved.

The rest of this paper is organized as follows: In Section.II, we introduce the system model and threat model of our scheme. Section.III provides our detail construction, which is followed by the security analysis in Section.IV. The performance evaluation of our scheme is presented in Section.V. Section.VI introduces applications that can use our proposed solution. We discuss related work in Section.VII and conclude our work in Section.VIII.

II. Models and Assumptions

A. System Models

We consider a cloud system composed of three major entities as shown in Fig.1: the *cloud server*, *group users* and *the third party auditor* (TPA). The cloud server is the party that provides data storage services to group users. Group users consists of a number of general group users and a *master user*, who is the owner of the shared data and manages the membership of other group users. All the group users can access and modify data. TPA refers to any party that checks the integrity of data being stored on the cloud. As our proposed scheme allows public integrity checking, the *TPA can actually be any cloud user as long as he/she has access to the public keys*. In our design, data can be uploaded/created by either the master user or other group users. We assume data is stored in

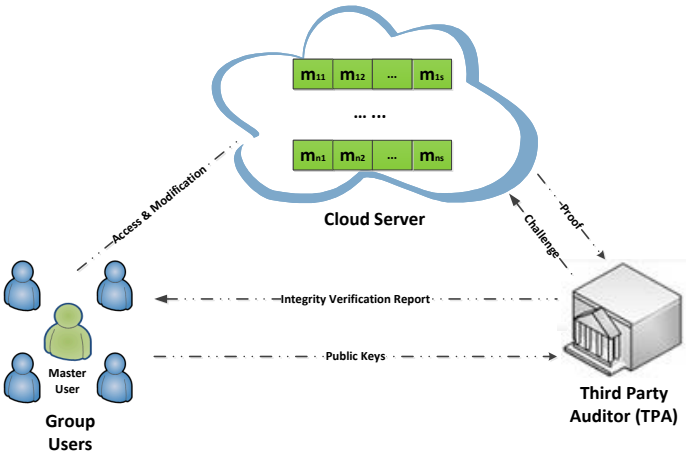

Fig. 1. System Model

form of files which is further divided into a number of blocks. For integrity checking, each data block is attached with an authentication tag that is originally generated by the master user. When a user adds or modifies a block, he/she (the user) updates the corresponding authentication tag with his/her own secret key without contacting the master user.

B. Security Models

In this work we consider the following factors that may impact data integrity: 1) hardware/software failures and operational errors of system administrator; 2) external adversaries that corrupts data stored on the cloud; 3) revoked users who no longer have data access privilege but try to illegally modify data. We assume the cloud server to be curious but honest – the cloud server will follow the protocol but try to disclose the private information as much as possible. Different from previous work, we allow collusion between the cloud server and revoked group users. Since valid users are always allowed to modify data, we assume that they never collude with the cloud servers for corrupting data. However, the cloud and the valid users may collude with revoked users for helping them retain the modification privilege or impersonate other valid users (for generating tags). Therefore, our *security goals* are defined as follows: 1) if the data is corrupted, the cloud servers are not able to produce valid integrity proof information; 2) collusion among the cloud and any group users other than the master user will not give users any chance to forge authentication on behave of other users. In particular, revoked users shall not be able to impersonate valid users and generate legitimate tags by collusion attacks.

III. Our Scheme

A. Overview

In our scheme, there are K users u_k, $0 \leq k \leq K-1$ in a group sharing data stored on cloud servers and u_0 is the master user. u_0 is also the owner of data and manages the membership of the group. Therefore, u_0 can revoke any other group users when necessary. All users in the group can access and modify data stored on cloud servers. Specifically, our scheme consists of 7 algorithms:

$\{KeyGen, Setup, Update, Challenge, Prove, Verify, User\ Revocation\}$. In the $KeyGen$ procedure, the master user u_0 generates the public keys and master keys for the system, and other group users generate secret keys for themselves. Then, u_0 processes files for storage in the $Setup$ procedure and generates the corresponding authentication tags. To modify the data stored on cloud servers, any group user u_k can run the $Update$ algorithm without the help of the master user u_0. The TPA can use the $Challenge$ algorithm to challenge the cloud servers and let them show the proof that they actually store the data correctly by running the $Prove$ algorithm. With proof information and public keys, the TPA can verify the integrity of data stored on cloud servers using the $Verify$ algorithm. When a user leaves the group or misbehavior of a user is detected, the master user u_0 runs the $User\ Revocation$ algorithm to revoke the user.

B. Scheme Description

In this section, we describe the design details of each algorithm. Let $H(\cdot)$ denote the one-way hash function [15], G be a multiplicative cyclic group of prime order q, g and u be two random generators of G. $e : G \times G \to G_1$ is a bilinear map that has the properties of bilinearity, computability and non-degenerate. F is the file to be outsourced to cloud servers and is split into n blocks, and each block has s elements: $\{F_{ij}\}, 1 \le i \le n, 0 \le j \le s-1$. $f_{\vec{a}(x)}$ denotes a polynomial with coefficient vector $\vec{a} = (a_0, a_1, \cdots, a_{s-1})$.

KeyGen: Each user $u_k, 0 \le k \le K-1$ in the group first randomly chooses $\epsilon_k \leftarrow Z_q^*$ and generates $\kappa_k = g^{\epsilon_k}$. For group users $u_k, 1 \le k \le K-1$, they also generate $g^{\frac{1}{\epsilon_k}}$ and send $g^{\frac{1}{\epsilon_k}}$ to the master user u_0. Then, the master user u_0 computes $(g^{\frac{1}{\epsilon_k}})^{\epsilon_0} = g^{\frac{\epsilon_0}{\epsilon_k}}$. u_0 also randomly chooses $\alpha \leftarrow Z_q^*$ and generates $\{g^{\alpha^j}\}, 0 \le j \le s-1$. The public keys (PK), master keys (MK) of the system and secret keys (SK) of users are:

$$
\begin{aligned}
&PK = \{g, u, q, \{g^{\alpha^j}\}_{0 \le j \le s-1}, \{\kappa_k, g^{\frac{\epsilon_0}{\epsilon_k}}\}_{0 \le k \le K-1}\} \\
&MK = \{\epsilon_0, \alpha\} \\
&SK_k = \{\epsilon_k\}_{1 \le k \le K-1}
\end{aligned}
$$

Setup: To outsource a file F, the master user u_0 splits it into n data blocks, and each block has s elements: $\{m_{ij}\}, 1 \le i \le n, 0 \le j \le s-1$. Then, u_0 generates authentication tag σ_i for each data block as:

$$\sigma_i = (u^{B_i} \cdot \prod_{j=0}^{s-1} g^{m_{ij}\alpha^j})^{\epsilon_0} = (u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)})^{\epsilon_0} \tag{1}$$

where $\vec{\beta_i} = \{\beta_{i,0}, \beta_{i,1}, \cdots, \beta_{i,s-1}\}$ and $\beta_{i,j} = m_{i,j}$. $B_i = H(\{z_i||t_i||0\})$, z_i is the index of m_i, $t_i = H(m_i)$. u_0 uploads data blocks and the corresponding authentication tags σ_i to cloud servers, and also sends B_i to the TPA (in practical implementation B_i can be first uploaded to cloud servers and later on downloaded by TPA).

Update: Suppose a group user $u_k, k \ne 0$ modifies a data block m_i to m_i'. He computes the tag for m_i' with his own secret key ϵ_k as:

$$\sigma_i' = (u^{B_i'} \cdot \prod_{j=0}^{s-1} g^{m_{ij}'\alpha^j})^{\epsilon_k} = (u^{B_i'} \cdot g^{f_{\vec{\beta_i'}}(\alpha)})^{\epsilon_k} \tag{2}$$

where $\vec{\beta_i'} = \{\beta_{i,0}', \beta_{i,1}', \cdots, \beta_{i,s-1}'\}$ and $\beta_{i,j}' = m_{i,j}'$. $B_i' = H(\{z_i||t_i'||k\})$. User u_k uploads m_i' and σ_i' to cloud servers and B_i' is sent to TPA (in practical implementation B_i' can be first uploaded to cloud servers and later on downloaded by TPA). The cloud and/or TPA can verify whether these tags are indeed updated by u_k through checking the following:

$$e(u^{B_i'}, \kappa_k^R) \cdot e(g^{f_{\vec{\beta_i'}}(\alpha)}, \kappa_k^R) \stackrel{?}{=} e(\sigma_i', g) \tag{3}$$

Challenge: To challenge the cloud server for checking data integrity, the TPA first randomly chooses d data blocks, indices of which compose a set D. Suppose the chosen d blocks are collectively modified by a set of C users, $0 \le |C| \le K-1$. The TPA generates a random number R and produces $X = (g^{\frac{\epsilon_0}{\epsilon_k}})^R$, where $k \in C$ (TPA is aware of the set C by looking at B_i and/orB_i'). Finally, the TPA produces the challenging message $CM = \{D, X, g^R\}$ and sends it to the cloud server.

Prove: On receiving the challenging message $CM = \{D, X, g^R\}$, the cloud server first computes

$$\psi = \prod_{j=0}^{s-1} (g^{\alpha^j})^{A_j} = g^{f_{\vec{A}}(\alpha)} \tag{4}$$

where $\vec{A} = (\sum_{i \in D} m_{i,0}, \cdots, \sum_{i \in D} m_{i,s-1})$, and $A_j = \sum_{j \in D} m_{i,j}$. Then, for data blocks in challenged set D modified by user $u_k, k \in C$, the cloud server computes

$$\pi_i = e(\sigma_i, g^{\frac{\epsilon_0 R}{\epsilon_k}}) = e((u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)}), g)^{\epsilon_0 R} \tag{5}$$

For those blocks not modified or modified by u_0, the cloud server computes

$$\pi_i = e(\sigma_i, g^R) = e((u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)}), g)^{\epsilon_0 R} \tag{6}$$

The cloud server generates

$$\pi = \prod_{i \in D} \pi_i \tag{7}$$

Finally, the cloud server responds the TPA with proof information $Prf = \{\pi, \psi\}$.

Verify: Based on the proof information Prf, the TPA verifies the integrity of file as

$$e(\eta, \kappa_0^R) \cdot e(\psi, \kappa_0^R) \stackrel{?}{=} \pi \tag{8}$$

where $\eta = u^\omega$ and $\omega = \sum_{i \in D} B_i$. If Eq.8 holds, the TPA outputs the verification result $VerifyRst$ as $Accept$; otherwise, outputs $VerifyRst$ as $Reject$.

User Revocation: Whenever there is a user to be revoked, say $u_k, k \ne 0$, the master user u_0 will notify cloud servers, valid group users and the TPA that u_k is revoked. On receiving the user revocation request, the cloud server will update the authentication tags of blocks that were last modified by u_k as:

$$
\begin{aligned}
\sigma_i' &= e(\sigma_i, g^{\frac{\epsilon_0}{\epsilon_k}})^\rho \\
&= e((u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)})^{\epsilon_k}, g^{\frac{\epsilon_0}{\epsilon_k}})^\rho \\
&= e((u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)}), g)^{\epsilon_0 \rho}
\end{aligned} \tag{9}
$$

where ρ is a random number chosen by the cloud server. On receiving the user revocation notification, the TPA will discard the public information $g^{\frac{\epsilon_0}{\epsilon_k}}$ and will no longer use it in the $Challenge$ algorithm. Let Rev denote the set of challenging data blocks that were last modified by the revoked user u_k. When the cloud server runs the $Prove$ algorithm, it generates proof information π for data blocks in set $D \backslash Rev$ as:

$$\pi = \prod_{i \in D \backslash Rev} \pi_i \tag{10}$$

For those blocks in Rev, the cloud server computes

$$\vartheta = \prod_{i \in Rev} \sigma_i' = \prod_{i \in Rev} e((u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)}), g)^{\epsilon_0 \rho} \tag{11}$$

The cloud server also computes:

$$\psi = \prod_{j=0}^{s-1} (g^{\alpha^j})^{A_j} = g^{f_{\vec{A}}(\alpha)} \tag{12}$$

where $\vec{A} = (\sum_{i \in D \backslash Rev} m_{i,0} \cdot \sum_{i \in Rev} m_{i,0}\rho, \cdots, \sum_{i \in D \backslash Rev} m_{i,s-1} \cdot \sum_{i \in Rev} m_{i,s-1}\rho)$. The cloud server also computes $\chi = u^{\rho}$ and sends the proof information back to the TPA as:

$$Prf = \{\pi, \vartheta, \psi, \chi\}$$

On receiving the proof information, the TPA first computes $\omega' = \sum_{i \in Rev} B_i$ and $\omega'' = \sum_{i \in D \backslash Rev} B_i$. Then, the TPA generates $\eta = \chi^{\omega'} \cdot u^{\omega''}$ and verifies the integrity as

$$e(\eta, \kappa_0^R) \cdot e(\psi, \kappa_0^R) \stackrel{?}{=} \pi \cdot \vartheta^R \tag{13}$$

If Eq.13 holds, the TPA outputs the verification result $VerifyRst$ as $Accept$; otherwise, outputs $VerifyRst$ as $Reject$.

Correctness: We analyze the correctness of our construction based on Eq.8 and Eq.13 as
Eq.8:

$$\begin{aligned} & e(\eta, \kappa_0^R) \cdot e(\psi, \kappa_0^R) \\ &= e(u^{\sum_{i \in D} B_i}, g)^{R\epsilon_0} \cdot e(g^{f_{\vec{A}}(\alpha)}, g)^{R\epsilon_0} \\ &= \prod_{i \in D} e(u^{B_i}, g)^{R\epsilon_0} \cdot \prod_{i \in D} e(g^{f_{\vec{\beta_i}}(\alpha)}, g)^{R\epsilon_0} \\ &= \prod_{i \in D} e((u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)}), g)^{R\epsilon_0} \\ &= \pi \end{aligned} \tag{14}$$

Eq.13:

$$\begin{aligned} & e(\eta, \kappa_0^R) \cdot e(\psi, \kappa_0^R) \\ &= \prod_{i \in D \backslash Rev} e(u^{B_i}, g)^{R\epsilon_0} \cdot \prod_{i \in Rev} e(u^{B_i}, g)^{R\rho\epsilon_0} \\ &\cdot \prod_{i \in D \backslash Rev} e(g^{f_{\vec{\beta_i}}(\alpha)}, g)^{R\epsilon_0} \cdot \prod_{i \in Rev} e(g^{f_{\vec{\beta_i}}(\alpha)}, g)^{R\rho\epsilon_0} \\ &= \prod_{i \in D \backslash Rev} e((u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)}), g)^{R\epsilon_0} \\ &\cdot \prod_{i \in Rev} e((u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)}), g)^{R\rho\epsilon_0} \\ &= \pi \cdot \vartheta^R \end{aligned} \tag{15}$$

From the above Eq.14 and Eq.15, it is easy to see that our scheme is correct.

C. Efficient Support for Multi-file Checking

For cloud systems in which data blocks are subject to frequent modifications by group users, the TPA may need to check data integrity often to assure data integrity. In large-scale systems, performing integrity checking file by file is inefficient in terms of both bandwidth consumption and computational cost. Specifically, given a set of T different files $F_t = \{m_{ti,j}\}$ $1 \le t \le T, 1 \le i \le n_t, 0 \le j \le s_t - 1$, it is desirable for the TPA to be able to aggregate integrity checking operations of T files into one challenge and one verification in order to reduce cost, where n_t is the number of data blocks in file F_t and s_t is the number of elements in each block. To this end, we design a batch verification algorithm based on our single file checking scenario, and enable the TPA to handle integrity checking of T files at the cost comparable to the single file scenario. In the batch verification algorithm, $KeyGen$, $Setup$ and $Update$ algorithms are same as those in the single file scenario respectively. Here we focus on introducing the $Batch - Challenge$, $Batch - Prove$ and $Batch - Verify$ as below.

Batch-Challenge: The challenge algorithm for checking T files is same as the single file scenario. Notably, the challenging message $CM = \{D, X, g^R\}$ now contains the information for data blocks of all these T files.

Batch-Prove: On receiving the challenging message $CM = \{D, X, g^R\}$, the cloud server first runs the $Prove$ algorithm for each single file and generates $\psi_t, \pi_t, 1 \le t \le T$. Then the cloud server aggregates the proof information into two elements as

$$\pi = \prod_{t=1}^{T} \pi_t \quad \psi = \prod_{t=1}^{T} \psi_t$$

Finally, the cloud server responds the TPA with proof information $Prf = \{\pi, \psi\}$.

Batch-Verify: On receiving the proof information Prf, the TPA first computes $\omega_t = \sum_{i \in D} B_{ti}$ for each file and generates $\eta = u^{\sum_{t=1}^{T} \omega_t}$. Then he verifies the integrity of the T files as

$$e(\eta, \kappa_0^R) \cdot e(\psi, \kappa_0^R) \stackrel{?}{=} \pi \tag{16}$$

If Eq.16 holds, the TPA outputs the verification result $VerifyRst$ as $Accept$ for these T files; otherwise, outputs $VerifyRst$ as $Reject$.

Based on above batch verification construction, we can see that the computational cost on TPA side for verification of T files is almost the same as that of one file.

Batch-Correctness: We analyze the correctness of our batch verification based on Eq.16 as follows:

$$\begin{aligned} & e(\eta, \kappa_0^R) \cdot e(\psi, \kappa_0^R) \\ &= e(u^{\sum_{i \in D, t=1}^{T} B_{ti}}, g)^{R\epsilon_0} \cdot e(g^{\sum_{t=1}^{T} f_{\vec{A_t}}(\alpha)}, g)^{R\epsilon_0} \\ &= \prod_{i \in D, t=1}^{T} e(u^{B_{ti}}, g)^{R\epsilon_0} \cdot \prod_{i \in D, t=1}^{T} e(g^{f_{\vec{\beta_{ti}}}(\alpha)}, g)^{R\epsilon_0} \\ &= \prod_{i \in D, t=1}^{T} e((u^{B_{ti}} \cdot g^{f_{\vec{\beta_{ti}}}(\alpha)}), g)^{R\epsilon_0} \end{aligned}$$

$$= \pi \tag{17}$$

From Eq.17, it is easy to see that our batch verification construction is correct.

D. Discussion

In this section, we discuss the error detection probability of our design and the choice of the size of set D in the $Challenge$ algorithm. As mentioned in Section.III-B, instead of choosing all data blocks of a file to check its integrity, we randomly choose d blocks as set D, in order to save communication and computational costs while remaining an acceptable level of error detection probability. Specifically, as shown in ref. [6], if there are 1% corrupted data blocks, 460 challenging data will result in 99% detection probability, and 95% detection probability only requires 300 challenging blocks, no matter the total number (greater than 460 and 300 respectively) of data blocks in the file. Therefore, the number d can be considered a fixed number in our scheme once the required error detection probability is determined.

IV. Security Analysis

A. Assumption

Our security proof is based on the following hard problems:

Definition IV.1. ***Computational Diffie-Hellman (CDH) Problem*** *[16]*

Let a, b be two random numbers. Given (g, g^a, g^b), it is computationally intractable to compute the value of g^{ab}, where G is a cyclic group of order q and g is a generator of G.

Definition IV.2. ***Bilinear Diffie-Hellman (BDH) Problem****[17]*

Let G, G_1 be two groups of prime order q, $e : G \times G \rightarrow G_1$ be an admissible bilinear map and g is the generator of G. Given $\{g, g^a, g^b, g^c\}$ for some $a, b, c \in Z_q^*$, the probability $Pr[Adv(g, g^a, g^b, g^c) = e(g, g)^{abc}]$ is negligible for any probabilistic polynomial time adversary (Adv).

Definition IV.3. ***t-Strong Diffie-Hellman (t-SDH) Problem*** *[18]*

Let $\alpha \overset{R}{\leftarrow} Z_q^*$. Given input as a $(t+1)-tuple$ $(g, g^\alpha, \cdots, g^{\alpha^t}) \in G^{t+1}$, where g is the generator of a cyclic group G of order q. For any probabilistic polynomial time adversary (Adv), the probability $Pr[Adv(g, g^\alpha, \cdots, g^{\alpha^t}) = (c, g^{\frac{1}{\alpha+c}})]$ is negligible for any value of $a \in Z_q^*/-\alpha$.

B. Security Proof

We first prove the following lemma which will be needed for proving unforgeability of the proof information of our scheme.

Lemma IV.4. *If $g^{f_{\vec{A}}(\alpha)}$ can be forged by an existed probabilistic polynomial time adversary Adv, we can construct an algorithm B to efficiently compute the solution to t-SDH problem based on Adv.*

By following the idea in Ref.[19], we prove Lemma.IV.4 as:

Proof: Suppose an Adv can generate $f_{\vec{A_1}}(\alpha)$ such that $g^{f_{\vec{A_1}}(\alpha)} = g^{f_{\vec{A}}(\alpha)}$, where $f_{\vec{A}}(x)$ and $f_{\vec{A_1}}(x)$ are known to the Adv. The Adv can construct another polynomial $f_{\vec{A_2}}(x) = f_{\vec{A}}(x) - f_{\vec{A_1}}(x)$. Therefore, $g^{f_{\vec{A_2}}(\alpha)} = g^{f_{\vec{A}}(\alpha)}/g^{f_{\vec{A_1}}(\alpha)} = g^{f_{\vec{A}}(\alpha) - f_{\vec{A_1}}(\alpha)} \in Z_q[x]$. As $f_{\vec{A_1}}(\alpha) = f_{\vec{A}}(\alpha)$ and $f_{\vec{A_2}}(\alpha) = 0$, α becomes a root of polynomial $f_{\vec{A_2}}(x)$. By factoring $f_{\vec{A_2}}(x)$ [20], B can easily find $SK = \alpha$. Based on $SK = \alpha$, B can easily find a number c and get $(c, g^{\frac{1}{\alpha+c}})$ as solution to the instance of the t-SDH problem given the system parameters. ■

Lemma IV.5. *If $e(\psi, \kappa_0^R)$ can be forged by an existed probabilistic polynomial time adversary Adv, we can construct an algorithm B to efficiently compute the solution to the BDH problem based on the Adv.*

Proof: Suppose an Adv can forge $e(\psi, \kappa_0^R)$, it can compute $e(g, g)^{f_{\vec{A}}(\alpha)\epsilon_0 R}$. With the given information, the Adv can get $g^\epsilon, g^{f_{\vec{A}}(\alpha)}$ and g^R, where ϵ, R and $f_{\vec{A}}(\alpha)$ are unknown (proved in Lemma IV.4), and thus solving the BDH problem. ■

Unforgeability of the proof information of our design can be proven as the follows:

Theorem IV.6. *If there exists a probabilistic polynomial time adversary Adv that successfully convinces the TPA to accept the forged proof information for a corrupted file with non-negligible advantage, we can use Adv to construct a polynomial time algorithm B that solves the BDH problem.*

Proof: To generate a proof information that can be accepted by the TPA, Adv either somehow generate $Prf = (\psi, \pi)$ which is the valid proof information as generated with our $Prove$ algorithm. According to Lemma IV.4, Adv is not able to forge ψ and hence cannot generate Prf.

Another possible case is that the adversity Adv can generate a $Prf' = (\psi', \pi'), Prf' \neq Prf$ and pass the verification process in our proposed scheme. Therefore, we can get the following two equations:

$$e(\eta, \kappa_0^R) \cdot e(\psi, \kappa_0^R) \overset{?}{=} \pi \tag{18}$$

$$e(\eta, \kappa_0^R) \cdot e(\psi', \kappa_0^R) \overset{?}{=} \pi' \tag{19}$$

Dividing Eq.18 with Eq.19, we obtain:

$$\frac{e(\psi, \kappa_0^R)}{e(\psi', \kappa_0^R)} = \frac{\pi}{\pi'} \tag{20}$$

Now we do a case analysis on whether $\pi = \pi'$.
Case 1: $\pi \neq \pi'$. We rewrite Eq.20 as

$$e(\psi, \kappa_0^R) = e(\psi', \kappa_0^R) \cdot \frac{\pi}{\pi'} \tag{21}$$

Since $\frac{\pi}{\pi'}$ and $e(\psi', \kappa_0^R)$ are known values to the Adv, he can compute $e(\psi, \kappa_0^R)$. As we proved in Lemma IV.5, if $e(\psi, \kappa_0^R)$ can be forged, we can efficiently find solution to the BDH problem. Therefore, no Adv can find a valid response $Prf', Prf' \neq Prf$ and $\pi \neq \pi'$ with non-negligible probability.

	Single Task		Multiple Tasks		User Revocation
	Comp.Cost (User)	Comm.Cost	Comp.Cost (User)	Comm.Cost	Comp.Cost (Cloud)
Ref.[11]	$(d+C)$EXP +$(d+2C)$MUL +$(C+1)$Pairing	$d*\|index\|+2C\|G\|$	$(d+C)T$ EXP +$(d+2C)$T MUL +$(C+1)$T Pairing	$dT*\|index\|+2CT\|G\|$	Y Exp
Our Scheme	1 EXP + 2 Pairing	$d*\|index\|+(C+3)\|G\|$	2 EXP + 2 Pairing	$d*\|index\|+(C+3)\|G\|$	Y EXP+Y Pairing

Table.1 Complexity Summary: in this table, C is number of users that modify files, d is the number of of blocks selected for challenging, $|index|$ is the size of block index, $|G|$ is the size of a group element and T is the number of files for verification; EXP and MUL are one multiplication operation and one exponentiation operation on Group G respectively, Pairing is a bilinear pairing operation.

Case 2: $\pi = \pi'$. We rerite Eq.20 as

$$\frac{e(\psi, \kappa_0^R)}{e(\psi', \kappa_0^R)} = \frac{\pi}{\pi'} = 1 \tag{22}$$

From Eq.22, we can infer that $e(\psi, \kappa_0^R) = e(\psi', \kappa_0^R)$. Therefore, no Adv can find a valid $Prf' = (\psi', \pi'), Prf' \neq Prf$ with non-negligible probability.

■

Security against collusion attacks of our scheme can be proven as follows:

Theorem IV.7. *If there exists a probabilistic polynomial time adversary Adv that can collude with the cloud server and revoked users to generate authentication tags on behalf of valid users, we can construct a polynomial time algorithm B that solves the CDH problem.*

Proof: Suppose the a probabilistic polynomial time adversary Adv can generate the block tags impersonating a valid user u_k. Therefore, he will be able to compute $u^{B_i\epsilon_k} \cdot g^{\epsilon_k f_{\vec{\beta_i}}(\alpha)}$ as part of the authentication tags. Assuming $u = g^t$ for an unknown integer t, we have $u^{B_i\epsilon_k} \cdot g^{\epsilon_k f_{\vec{\beta_i}}(\alpha)} = g^{U\cdot\epsilon_k}$, where $U = tB_i + f_{\vec{\beta_i}}(\alpha)$. Note that, $g^U = u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)}$ can be computed by Adv if he knows the data block and $\kappa_k = g^{\epsilon_k}$ is public key; U and ϵ_k are integers in Z_q^* unknown to Adv. In this case Adv actually has solved CDH problem. ■

V. Performance Evaluation

A. Numerical Analysis

In this section, we numerically analyze our proposed scheme and compare it with Ref.[11] in terms of computational cost and communication cost. For simplicity, in the rest of this paper, we use EXP and MUL[1] to denote the complexity of one exponentiation operation and one multiplication operation on Group G respectively.

1) Computational Cost: As shown in Section.III, there are 7 algorithms in our scheme: $KeyGen$, $Setup$, $Challenge$, $Update$, $Prove$, $Verify$ and $User\ Revocation$. $KeyGen$ and $Setup$ are pre-processing procedures, which can be performed by group users off-line and will not influence the real-time verification performance. In the $KeyGen$ algorithm, the master user needs to perform $(s + K + 1)$ EXP operations to generate public keys and master keys for the system, where s is the number of elements in block and K is the number of group users. To setup the system, the master user conducts $(s + 2)n$ EXP and sn MUL operations for each file. When a user needs to modify or add data blocks, he executes the $Update$ algorithm with $(s+2)$EXP+$(s+1)$MUL operations to generate the corresponding tags. In order to check the integrity, the TPA first performs the $Challenge$ algorithm to generate the challenging message CM by choosing a constant number of random numbers at a negligible cost. The cloud server then runs the $Prove$ algorithm with s EXP, $(s + d)$ MUL and d Pairing operations, where d is a constant representing the number of blocks selected for challenging. To finally verify the integrity of the file, the TPA only needs 1 EXP and 2 Pairing operations. This property is interesting because such a cost can even be affordable to less powerful devices such as mobile phones. If there are some blocks last modified by revoked users, the TPA only needs to perform one more EXP and one more MUL operations compare to the scenarios without user revocation. In case multiple files shall be verified at the same time, our design can batch the verification operations of these files and aggregate them into one round operation. Consequentially, the TPA only needs 1 EXP and 2 Pairing operations to perform multiple file checking, the cost of which is the same as the single file scenario. To revoke a group user, no computational cost are required for group users; the cloud server needs Y pairing and Y EXP operations to update the corresponding tags, where Y is the number of data blocks last modified by the revoked user.

We now compare our proposed scheme with the existing scheme [11] and summarize the result in Table.1. To verify the integrity of a single file, ref.[11] costs the TPA $(d + C)$EXP+$(d+2C)$MUL+$(C+1)$Pairing operations, where C is number of users who modify this file. Our design can migrate most computational cost to cloud servers and makes the cost of the TPA independent to the number of challenging blocks and group users. In order to process multiple verification tasks simultaneously, the cost of the TPA in ref.[11] is linear to the task number. In contrast, our scheme can aggregate operations brought by multiple tasks and make the cost similar to the single file scenario, and thus outperforms ref.[11].

2) Communication Cost: In our scheme, the communication cost for data integrity checking mainly comes from the challenging message CM and the proof information Prf. The size of the $CM = \{D, X, g^R\}$ is $d*|index|+(C+1)|G|$ bits, where $|index|$ is the size of block index and $|G|$ is the size of a group element. For the proof information $Prf = \{\pi, \psi\}$, there are two group element with size $2|G|$ bits. If there are some blocks last modified by revoked users, $Prf = \{\pi, \vartheta, \psi, \chi\}$ and the size is $4|G|$ bits. Therefore, the total communication cost of an integrity verification task is $d*|index|+(C+3)|G|$ bits $(d*|index|+(C+5)|G|$ bits if there are user revocations). Considering the simultaneous checking of multiple files, our batch verification design allows users to aggregate these tasks into one challenge and one proof, and thus achieving the same communication cost as the single file scenario. When a user revocation occurs, our schemes only requires the master user

[1] When the operation is on the elliptic curve, EXP means scalar multiplication operation and MUL means one point addition operation.

to notify this revocation event to cloud server, the TPA and other users, which introduces negligible communication cost.

Now, we compare our proposed scheme with the existing scheme [11] and summerize the result in Table.1. In ref.[11] the communication cost for single verification task is $d * |index| + 2C|G|$ bits which is attribute to the challenge and proof generation processes and comparable to our scheme. However, when processing multiple tasks together, the size of proof information in ref.[11] will increase linearly to the number of tasks as shown in Table.1, where T is the number of files for verification. Differently, our batch verification design omits this kind of increase with information aggregation. In order to revoke a user, although ref.[11] has a similar cost to our scheme, their user revocation solution assumes non-collusion occurs between cloud servers and users.

B. Experimental Results

To evaluate the performance of our proposed scheme, we fully implemented our proposed scheme on Aamazon EC2 cloud using JAVA with JAVA Pairing-Based Cryptography Library (jPBC) [21]. On the cloud server, we deploy nodes running Linux with 8-core CPU and 32GB memory. The TPA is a desktop running Linux with 3.4GHz Intel i7-3770 CPU and 16GB memory. Machines for group users are laptops running Linux with 2.50GHz Intel i5-2520M CPU and 8GB memory. We set the security parameter $\lambda = 160$ bits, which achieve 1024-bits RSA equivalent security since our implementation is based on ECC. The shared files used for testing are open source codes Apache OpenOffice [22] which has 66,359 files. We set size of each data block as $4KB$, which is consistent with the ref.[6], [13]. To measure the influence of group size, we conduct experiments from small group with 5 users to large group with 150 users. In our experiments, we mainly evaluate the computational and communication performance of our scheme. All experimental results represent the mean of 50 trials. Our implementation is not optimized (e.g. it is a single process/thread program in some parts). Therefore, further performance improvements of our scheme is possible.

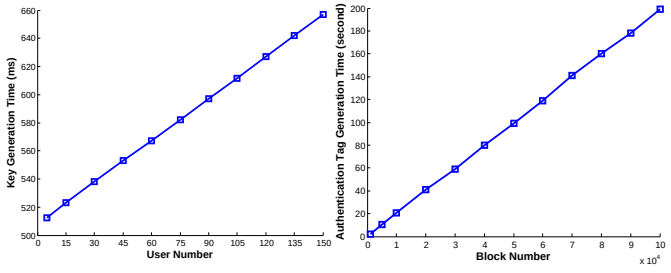

Fig. 2. (a) Key Generation Time (b) Authentication Tag Generation Time

1) System Setup: We first evaluate the generation of public keys, master keys and secrets for the system. Our result in Fig.2(a) indicates that the key generation cost is proportional to the group size, which is consist with our analysis in Section.V-A1. To show the performance of authentication tag generation, we vary the number of blocks in the file is from 1000 to 100,000. As shown in Fig.2(b), the tag generation time increases proportionally to the number of blocks, from 2.11s to 198.23s. Note that, the system setup cost is one-time, which can be conducted off-line and will not influence the realtime performance of integrity checking.

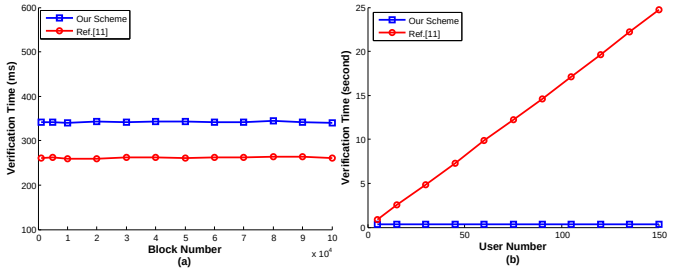

Fig. 3. (a) Verification Time On Different Block Number (b) Verification Time On Different User Number

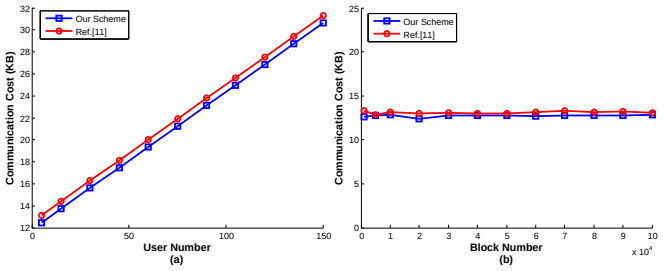

Fig. 4. (a) Communication Cost On Different User Number (b) Communication Cost On Different Block Number

2) Realtime Checking - Single File: As we discussed in Section.III-D, we can set the number of challenging block as 460 in our experiments to achieve 99% detection probability. We first measure the performance of the checking of single file. Fig.3(a) and Fig.3(b) shows that the verification performance on the TPA side is constant, i.e., the user number and file size have no influence on the TPA computational cost, which is consistent with our previous analysis in Section.V-A1. Although we do not conduct experiments on larger groups and files, it is easy to validate that the computational cost on the TPA side is constant regardless of the number of group users. However, as shown in Fig.3(b), the computation cost of the TPA in ref.[11] is proportional to the number of user and is thus much more expensive than our scheme as group size increases. Considering the communication cost, when we change the number of users that modify blocks in the file from 5 to 150, Fig.4(a) shows that our scheme allow a TPA to verify the integrity of a file at the communication cost from 12.78KB to 30.87KB. Fig.4(b) indicates that file size does not have influence on the communication cost of our scheme. Note that, the communication cost in our design is only 30.87KB even 150 users modify the blocks of the file at the same time, which can be efficiently handled in today's networks. Compared with ref.[11], we achieved comparable communication cost as shown in Fig.4(a) and Fig.4(b).

3) Multiple Files Checking: To show the benefits of our batch verification design for multiple files scenario, we change number of simultaneous verification tasks from 250 to 2000. Since the group size does not influence the checking performance of the TPA, we set group size as 50 here. As shown in Fig.5(a), the computational cost of the TPA side is around 400ms with the increasing of simultaneous task

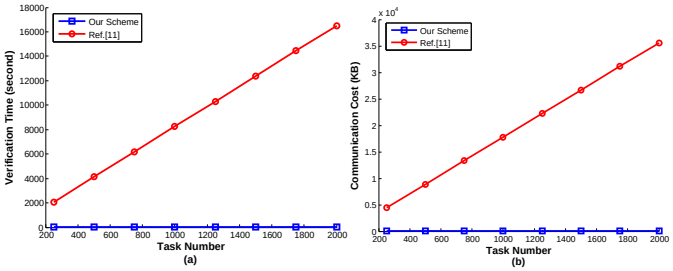

Fig. 5. (a) Verification Time On Different Task Number (b) Communication Cost On Different Task Number

number. In addition, Fig.5(b) shows that the communication cost of our scheme does not change with the number of tasks/files. In contrast, Fig.5(a) and Fig.5(b) indicate that in ref.[11] both computational cost and communication cost are linear to the number of simultaneous integrity checking tasks, which significantly limit its scalability in terms of the number of tasks.

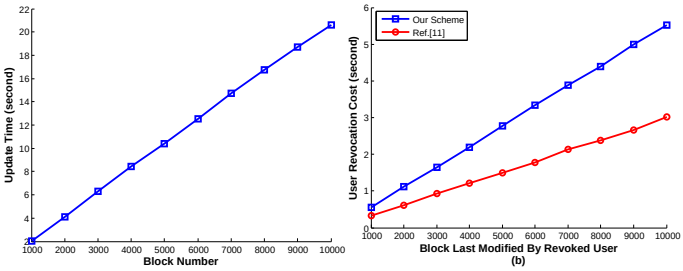

Fig. 6. (a) Block Update Cost (b) User Revocation Cost

4) Update and User Revocation: The update procedure of data blocks in our scheme is similar to the setup procedure. As shown in Fig.6(a), the update cost is proportional to the number of blocks for modification. To revoke a user, no computational cost and communication cost are required for group users and the TPA in our scheme. The cloud server can help the group users to temperately update the corresponding authentication tags when a user revocation occurs. Fig.6(b) shows that our tag update performance is comparable with ref.[11]. However, it is worth to note that the user revocation efficiency in ref.[11] is based on the non-collusion assumption, which does not exist any longer in our design.

From the above experimental results, we clearly see that the file size only influences the pre-processing procedures of our proposed scheme, which is one-time cost and does not have any effect on real-time performance of integrity checking. While most existing work such as ref.[11] is limited by the group size and task number due to the performance, our scheme can efficiently handle large size of groups and/or a large number of tasks. Therefore, our proposed scheme has high scalability in terms of the sizes of files/groups as well as the number of tasks.

VI. Applications

In this section, we briefly discuss the applicability of our proposed scheme in real-world application scenarios. In particular, we consider two contrasting applications.

One potential application is cloud-based open source software development, wherein source code and corresponding documents are stored on public storage platform such as cloud servers. In open source software, the number of contributors can be very large, and the member join and leave is very frequent. In this case, each contributor should be able to modify codes with integrity checking, and the performance should not be influenced by the number of contributors. In addition, the join and leave of contributors should be handled efficiently without affecting other contributors and the integrity checking performance. Our proposed solution is suitable for this kind of application because our realtime performance is not affected by group size as shown in above analysis. At the same time, the join and leave of a contributor can be efficiently delegated to cloud servers without help of any other contributors.

Another application scenario is cloud-based collaboration of organizational IT department, which may have a relative small number of group users. But the documents (e.g., software code, system bug report, daily work reports, operational data) of an organizational IT department may be subject to frequently modification by different employees due to the daily operational needs. This kind of high frequency modifications require fast integrity checking for each modification and efficient processing for a large number files. As shown in our previous analysis and experimental results, our proposed scheme can aggregate the integrity checking of multiple documents that underwent modifications by multiple employees into a single operation. Therefore, our scheme is appropriate for this kind of applications.

VII. Related Work

The problem of data integrity checking in cloud have been extensively studied in past years by a number of Proof of Retrievability (POR) and Proof of Data Possession (PDP) schemes [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. In ref.[5], [6], concepts of POR and PDP were first proposed separately using RSA-based homomorphic authentication tags. The efficiency and security of the POR scheme were later enhanced by Shacham et al. [7]. To support dynamic operations in public verification, Wang et al. [9] introduced a new mechanism by utilizing Merkle Hash Tree. In ref. [10] Wang et al. further improved the public integrity checking with simple batch verification and thus substantially reduce the cost of the TPA. Yuan et al. [12] recently proposed a public POR scheme with constant communication cost by using a nice algebraic property of polynomial. However, these schemes only consider single data owner who has the system secret key and is the only party allowed to modify the shared data on cloud. Trivial extension of these schemes to support the modification from multiple group users will introduce tremendous performance issues. Without giving the secret key to each group user, the data owner has to serve as the central point to perform all tags update tasks and thus suffers from heavy computational cost and bandwidth consumption.

In order to improve the previous works to support multiple writers, Wang et al. [14] first proposed a public integrity auditing scheme for shared data on cloud based on ring signature-based homomorphic authenticators. In their scheme, user revocation is not considered and the auditing cost grows

with group size and data size. Recently, Wang et al. [11] enhanced their previous public integrity verification scheme with the support of user revocation. Nevertheless, their design assume that there is non collusion among cloud servers and revoked users. However, once cloud servers and the revoked users work together during their proxy tag update process, they can discover the secret keys of all other valid users. What is more, verification cost of the TPA (can also be users) in ref. [11] is linear to the size of group. Batch verification is not supported in their design. Therefore, this scheme is limited in its scalability.

VIII. Conclusion

In this paper, we propose a novel data integrity checking scheme that support multiple writers to the share data. Our proposed scheme is featured by salient properties of public integrity checking and constant computational cost on user side. We achieve this through our novel design on polynomial-based authentication tags which allows aggregation of tags of different data blocks. For system scalability, we further empower the cloud with the ability to aggregation authentication tags from multiple writers into one when sending the integrity proof information to the verifier (who may be general cloud users). As a result, just a constant size of integrity proof information needs to be transmitted to the verifier no matter how many data blocks are being checked and how many writers are associated with the data blocks. Moreover, our novel design allows secure delegation of user revocation operations to the cloud even if cloud servers collude with malicious users. Last but not least, our proposed scheme allows aggregation of integrity checking operations for multiple tasks (files) through our batch integrity checking technique. As compared to existing schemes, our solution has obvious advantage in terms of efficiency, scalability and security. Extensive numerical analysis and real-world experiments validate the performance of our scheme. Rigorous formal security analysis shows that our solution is provably secure.

References

[1] "Cloud9," https://c9.io.

[2] "Codeanywhere," https://codeanywhere.net/.

[3] "Amazon web service. summary of the amazon ec2 and amazon rds service disruption in the us east region," http://aws.amazon.com/message/65648/.

[4] "Dropbox. dropbox forums on data loss topic," http://forums.dropbox.com/tags.php?tag=data-loss.

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

[6] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song, "Provable data possession at untrusted stores," in *ACM Conference on Computer and Communications Security*, 2007, pp. 598–609.

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

[8] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: theory and implementation," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, ser. CCSW '09. New York, NY, USA: ACM, 2009, pp. 43–54.

[9] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proceedings of the 14th European conference on Research in computer security*, ser. ESORICS'09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 355–370.

[10] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proceedings of the 29th conference on Information communications*, ser. INFOCOM'10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 525–533.

[11] B. Wang, L. Baochun, and L. Hui, "Public auditing for shared data with efcient user revocation in the cloud," in *2013 Proceedings IEEE INFOCOM (INFOCOM'2013)*, Turin, Italy, Apr. 2013, pp. 2750–2758.

[12] J. Yuan and S. Yu, "Proofs of retrievability with public verifiability and constant communication cost in cloud," in *Proceedings of the 2013 international workshop on Security in cloud computing*, ser. Cloud Computing '13. Hangzhou, China: ACM, 2013, pp. 19–26.

[13] J. Yuan and S. Y, "Secure and constant cost public cloud storage auditing with deduplication," in *IEEE Conference on Communications and Network Security 2013 (IEEE CNS 2013)*, Washington, USA, Oct. 2013.

[14] B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," in *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, ser. CLOUD '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 295–302.

[15] D. E. Eastlake and P. E. Jones, "US Secure Hash Algorithm 1 (SHA1)," http://www.ietf.org/rfc/rfc3174.txt?number=3174.

[16] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, pp. 644–654, Sep. 1976.

[17] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '01. London, UK, UK: Springer-Verlag, 2001, pp. 213–229.

[18] D. Boneh and X. Boyen, "Short signatures without random oracles," in *EUROCRYPT*, 2004, pp. 56–73.

[19] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT*, 2010, pp. 177–194.

[20] V. Shoup, *A computational introduction to number theory and algebra*. New York, NY, USA: Cambridge University Press, 2005.

[21] jPBC, "http://gas.dia.unisa.it/projects/jpbc/."

[22] Apache, "http://www.openoffice.org/."