Public Integrity Auditing for Dynamic Data Sharing With Multiuser Modification

Jiawei Yuan and Shucheng Yu, *Member, IEEE*

***Abstract*— In past years, the rapid development of cloud storage services makes it easier than ever for cloud users to share data with each other. To ensure users' confidence of the integrity of their shared data on cloud, a number of techniques have been proposed for data integrity auditing with focuses on various practical features, e.g., the support of dynamic data, public integrity auditing, low communication/computational audit cost, and low storage overhead. However, most of these techniques consider that only the original data owner can modify the shared data, which limits these techniques to client read-only applications. Recently, a few attempts started considering more realistic scenarios by allowing multiple cloud users to modify data with integrity assurance. Nevertheless, these attempts are still far from practical due to the tremendous computational cost on cloud users, especially when high error detection probability is required by the system. In this paper, we propose a novel integrity auditing scheme for cloud data sharing services characterized by multiuser modification, public auditing, high error detection probability, efficient user revocation as well as practical computational/communication auditing performance. Our scheme can resist user impersonation attack, which is not considered in existing techniques that support multiuser modification. Batch auditing of multiple tasks is also efficiently supported in our scheme. Extensive experiments on Amazon EC2 cloud and different client devices (contemporary and mobile devices) show that our design allows the client to audit the integrity of a shared file with a constant computational cost of 340 ms on PC (4.6 s on mobile device) and a bounded communication cost of 77 kB for 99% error detection probability with data corruption rate of 1%.**

***Index Terms*— Integrity auditing, cloud storage, public verification, dynamic data, batch verification.**

I. Introduction

THE continuous development of cloud techniques has boosted a number of public cloud storage applications. In particular, more and more cloud storage applications are being used as collaboration platforms, in which data are not only persisted in cloud for storage but also subject to frequent modifications from multiple users. Real-world examples are cloud-based storage synchronization platforms such as Dropbox for Business [2] and Sugarsync [3], Version Control Systems (VCS) such as Subversion [4] and Concurrent Versions System [5], which enable multiple team members to work in sync, accessing and modifying same files on cloud servers anywhere anytime. For correct execution of this kind of 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 and update to date thereafter. This problem is important given the fact that cloud storage platforms, even well-known cloud platforms, may experience hardware/software failures, human errors and external malicious attacks [6], [7]. In addition, we observed that there have been large discrepancies between the numbers of data corruption events reported by users and those acknowledged by service providers [7], which also causes users to doubt whether or not their data on cloud are truly intact.

With the concern on data integrity of cloud storage services, users wish to have a way of auditing the cloud server to ensure that the server stores all their latest data without any corruption. To offer such a service, a series of schemes [8]–[22] have been proposed. However, for most of these existing schemes, only the data owner who holds secret keys can modify the data and all other users who share data with the data owner only have read permission. 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.

Considering practical scenarios wherein all users share data with each other in cloud have both read and write privileges, Wang et al. [18] proposed a public integrity auditing scheme using ring signature-based homomorphic authenticators. Nevertheless, the scalability of ref. [18] is limited by the group size and data size. Moreover, user revocation is not considered in this paper. In order to further enhance previous work, another attempt was made by Wang et al. [21]. However, ref. [21] still suffers from a non-trivial computational cost which is linear to the number of modifiers (especially for achieving high error detection probability with small data corruption rate, e.g., less then 0.1%) and the number of checking tasks, and thus is limited in scalability. In addition, user revocation in ref. [21] is based on the assumption that the cloud node responsible for updating signatures will not be compromised nor encounter internal errors, which has been proven not always true in practice. As a matter of fact, compromisation and internal errors of the cloud node

Manuscript received June 10, 2014; revised September 29, 2014 and February 16, 2015; accepted March 31, 2015. Date of publication April 15, 2015; date of current version June 25, 2015. This work was supported in part by the U.S. National Science Foundation under Award CNS-1338102 and in part by the Amazon AWS in Education Research Grant. A preliminary version of this paper was presented at IEEE INFOCOM 2014, Toronto, ON, Canada, April/May 2014. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Athanasios Vasilakos.

The authors are with the Department of Computer Science, University of Arkansas at Little Rock, Little Rock, AR 72204 USA (e-mail: jxyuan@ualr.edu; sxyu1@ualr.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2015.2423264

in ref. [21] will lead to the disclosure of users' secrets, and thus causing severe attacks such as user impersonation.

To design an efficient public integrity auditing scheme supporting multi-user modification and efficient user revocation simultaneously, we need to overcome the following major (not necessarily complete list of) challenges: 1) Aggregation of individually generated authentication tags. Specifically, any user with read and write privilege should be able to modify the data and generate new authentication tags without the help of the data owner. In this context, how to aggregate tags from different users becomes a challenge problem, because the tags are signed with individual users' secret keys which are different from each other. Without aggregation, a data integrity verifier has to process tags from different modifiers one by one for an auditing task, and thus limiting the scalability of scheme. A straightforward solution to this problem is to let all users share the same secret key, so all authentication tags are in the same format and can be easily aggregated. Nevertheless, this kind of solution is limited by another challenge: 2) efficient and secure user revocation. User revocation is a challenging issue in most security systems and usually involves disabling user secret keys. Upon user revocation, all authentication tags generated by the revoked user should be updated, and this heavy task is usually delegated to the cloud by disclosing partial secrets to it. This method, however, can lead to disclosure of secret keys of valid users once the cloud server node colludes with a revoked user. 3) public auditing. In practical systems, data integrity auditing can be performed not only by data owners or other group users but also by a third-party auditor or any general user who has public keys of the system.

In this work, we address above challenges and propose an efficient public integrity auditing scheme for cloud data sharing that supports multiple writers. Thanks to our novel design on polynomial-based authentication tags, we can empower the cloud to aggregate authentication tags from multiple writers into one when sending the integrity proof information to the verifier. As a result, just a constant size of integrity proof information and a constant number of computational operations are needed for the verifier, no matter how large the audited file is and how many writers are associated with the data blocks. Moreover, with the novel proxy authentication tag update technique, our scheme allows secure delegation of user revocation operations to the cloud and can defeat impersonation attacks from illegitimate users. By uniquely incorporating Shamir's Secret Sharing scheme [23], we further enhance the reliability of our design during the user revocation procedure. Last but not least, our proposed scheme allows aggregation of integrity auditing operations for multiple tasks (files) through our batch integrity auditing technique, which promote our scheme in terms of auditing efficiency and data corruption detection probability. Thorough analysis and extensive experimental results on Amazon EC2 cloud [24] and various client devices (PC and mobile devices) demonstrate the scalability and efficiency of our design.

The rest of this paper is organized as follows: In Section 2, we introduce models of our scheme. We provide technique preliminaries for our design in Section 3. Section 4 provides our detailed construction, which is followed by the security analysis in Section 5. The performance evaluation of our scheme is presented in Section 6. Section 7 introduces practical applications, in which our proposed scheme is applicable for integrity assurance. In Section 8, we discuss existing works related to our design. We conclude this paper in Section 9.

II. Models

A. System Models

We consider a cloud system composed of three major entities: 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 consist of a number of general users and a *master user*, who is the owner of the shared data and manages the membership of other group users. All group users can access and modify data. The TPA refers to any party that checks the integrity of data being stored on the cloud. As our proposed scheme allows public integrity auditing, the TPA can actually be any cloud user as long as he/she has access to the public keys. Once the TPA detects a data corruption during the auditing process, he/she will report the error to group users. In our design, data can be uploaded/created by either the master user or other group users. We assume data are stored in form of files which are further divided into a number of blocks. For integrity auditing, 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. Threat Model

In this work, we assume the cloud server to be *curious-but-honest*, which is consistent with ref. [21]. Specifically, the cloud server will follow the protocol, but it may lie to the users about the corruption of their data stored on it in order to preserve the reputation of its services. This kind of situation occurs many times, being it internationally or not, with existing cloud storage platforms [7]: the data loss events claimed by users are much more than those acknowledged by service providers. In this context, 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 impersonate valid users. Since valid users are always allowed to modify data, we assume that they are always honest when they are in the group. We also assume that secure communication channels (e.g., SSL) exist between each pair of entities.

III. Technique Preliminaries

A. Shamir's Secret Sharing

A (k, n)-Shamir's Secret Sharing scheme [23] divides (based on polynomials) a secret S into n shares. With any k shares, one can recover the secret S. However, knowledge of any k-1 or fewer shares leaves the secret

TABLE I
NOTATION

$H(\cdot)$	One-way hash function [25]
G	A multiplicative cyclic group
q	The prime order of Group G
$e : G \times G \to G_1$	A bilinear map
g, u	Random generators of group G
λ	Security parameter
F	Data file that will be split into n blocks
m_i	A data block of file F and will be split into s elements
m_{ij}	A block element of data block m_i
σ_i	Authentication tag generated for data block m_i
$f_{\vec{\alpha}(x)}$	a polynomial with coefficient vector $\vec{\alpha} = (\alpha_0, \alpha_1, \cdots, \alpha_{s-1}), \alpha_j \in Z_q^*$
$\epsilon_k, \alpha, \mu, R$	Random numbers, where $0 \le k \le K-1$

Algorithm: Setup$(1^\lambda, F) \to (PK, MK, SK_k, \sigma_i)$

Key Generation: Select K random number $\epsilon_k \overset{R}{\leftarrow} Z_q^*$ and generate $\nu = g^{\alpha\epsilon_0}$, $\kappa_0 = g^{\epsilon_0}$, $\{\kappa_k = g^{\epsilon_k}, g^{\frac{\epsilon_0}{\epsilon_k}}\}_{1 \le k \le K-1}$. Randomly choose $\alpha \overset{R}{\leftarrow} Z_q^*$ and generate $\{g^{\alpha^j}\}, 0 \le j \le s+1$. The public keys (PK), master keys (MK) of the system and secret keys (SK) of users are:

$$PK = \{g, u, q, \nu, \{g^{\alpha^j}\}_{0 \le j \le s+1}, \kappa_0, \{\kappa_k, g^{\frac{\epsilon_0}{\epsilon_k}}\}_{1 \le k \le K-1}\}$$
$$MK = \{\epsilon_0, \alpha\} \quad SK_k = \{\epsilon_k\}_{1 \le k \le K-1}$$

File Processing: Split file F into n data blocks, and each block into s elements: $\{m_{ij}\}, 1 \le i \le n, 0 \le j \le s-1$. Compute authentication tag $\sigma_i, 1 \le i \le n$ for each data block as:

$$\sigma_i = (u^{B_i} \cdot \prod_{j=0}^{s-1} g^{m_{ij}\alpha^{j+2}})^{\epsilon_0} = (u^{B_i} \cdot g^{f_{\vec{\beta_i}}(\alpha)})^{\epsilon_0} \qquad (1)$$

where $\vec{\beta_i} = \{0, 0, \beta_{i,0}, \beta_{i,1}, \cdots, \beta_{i,s-1}\}$ and $\beta_{i,j} = m_{i,j}$. $B_i = H(\{fname||i||t_i||k\})$, $fname$ is the file name, i is the index of data block m_i, t_i is the time stamp and k is the index of user in the group.

Fig. 1. Detailed construction: system setup.

completely undetermined. Specifically, as any k points can uniquely define a polynomial of degree $k-1$, by choosing $k-1$ random positive integers $a_1, a_2, \cdots, a_{k-1}$ from a finite filed of size q and set $a_0 = S$, we can construct the polynomial: $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1}$, where $a_i < q$ and q is a prime number. When there are n participants to share the secret S, we can construct n points out of $f(x)$ as $(j, f(j))_{1 \le j \le n}$ and give each participant a point. After that, given any k out of these n points, one can compute the coefficients of $f(x)$ using polynomial interpolation and recover the constant term a_0, which is set as the secret S. For more details about Shamir's Secret Sharing scheme, please refer to ref. [23].

IV. Our Construction

A. Detailed Construction

We now give the detailed construction of our design. In our scheme, there are K users u_k, $0 \le k \le K-1$ in a group sharing data stored on cloud 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. For simplicity of expression, we assume that the TPA performs the data integrity auditing procedure, who in practice can be any user knowing the public key. Table I summarizes the notation to be used in our construction.

Setup: To setup the system, the master user u_0 first runs the **Key Generation** part of the **Setup** algorithm as shown in Fig.1 and generates public keys (PK), master keys (MK) of the system and secret keys (SK) of users. In our design, each user will have his/her own secret keys for data modification.

To outsource a file F, the master user u_0 runs the **File Processing** procedure of the **Setup** algorithm to generate data blocks m_i and the corresponding authentication tags σ_i. u_0 then uploads data blocks and tags to the cloud. u_0 also publishes and maintains a *Log* for the file, which contains $\{i||t_i||k\}$ information for each block. Note that the size of each log record is 16 bytes, which is $\frac{1}{256}$ of the data block size (typically 4KB). For instance, the owner only needs to maintain a 4MB log file for a 1GB file that is split into 4KB blocks.

Update: We now show how to allow group users to modify the shared data. Suppose a group user $u_k, k \neq 0$ modifies a data block m_i to m_i'. u_k computes the tag for m_i' with his own secret key ϵ_k as:

$$\sigma_i' = (u^{B_i'} \cdot \prod_{j=0}^{s-1} g^{m_{ij}'\alpha^{j+2}})^{\epsilon_k} = (u^{B_i'} \cdot g^{f_{\vec{\beta_i'}}(\alpha)})^{\epsilon_k} \qquad (3)$$

where $\vec{\beta_i'} = \{0, 0, \beta_{i,0}', \beta_{i,1}', \cdots, \beta_{i,s-1}'\}$ and $\beta_{i,j}' = m_{i,j}'$. $B_i' = H(\{fname||i||t_i'||k\})$. u_k then uploads updated data blocks m_i' and its corresponding tags σ_i' to the cloud. u_k will also updates $\{i||t_i'||k\}$ in the log file. Note that, the size of our authentication tag for each data block is only 128 bytes, which is $\frac{1}{32}$ of the block size and thus introducing slight communication overhead for users.

Challenge: To audit data integrity, the TPA generates the challenge message $CM = \{D, X, g^R, \mu\}$ by running the **Challenge** algorithm as shown in Fig.2. Note that, the TPA is aware of the set C used in the *Challenge* algorithm by looking at records $\{i||t_i||k\}_{i \in D}$ in the log file. The TPA then sends the challenging message CM to the cloud.

Prove: On receiving the challenging message $CM = \{D, X, g^R, \mu\}$, the cloud will run the **Prove** algorithm in Fig.2 to generate the proof information $Prf = \{\pi, \psi, y\}$, which shows that it actually store the challenged data file correctly. Note that, polynomials $f(x) \in Z[x]$ have the algebraic property that $(x-r)$ perfectly divides the polynomial $f(x) - f(r), r \overset{R}{\leftarrow} Z_p^*$. With this property, the cloud can efficiently divide the polynomial $f_{\vec{A}}(x) - f_{\vec{A}}(\mu)$ with $(x-\mu)$ using polynomial long division in *Step 2* of the **Prove** algorithm. The cloud responds the TPA with proof information $Prf = \{\pi, \psi, y\}$.

Verify: Based on the proof information Prf, the TPA verifies the integrity of file F by running the **Verify** algorithm as shown in Fig.2.

User Revocation: We now provide a basic *User Revocation* design of our scheme. We will also introduce an advanced version of user revocation with improved reliability in Section 4.3.1.

Algorithm: Challenge$(1^\lambda, PK) \rightarrow (CM = \{D, X, g^R, \mu\})$

1) Randomly choose d data blocks as a set D (The size of set D is discussed in Section 4.3.2).

2) Suppose the chosen d blocks are modified by a set of users, denoted by C, $0 \leq |C| \leq K-1$. Generate two random numbers R and μ and produce set $X = \{(g^{\frac{\epsilon_0}{\epsilon_k}})^R\}_{k \in C}$. If the set D contains blocks last modified by any revoked user, add $(g^{\frac{\epsilon_0}{\epsilon_0+\rho}})^R$ to set X, where $(g^{\frac{\epsilon_0}{\epsilon_0+\rho}})$ is generated by the master user during the user revocation procedure (see Fig.3 for details).

Algorithm: Prove$(CM, F, PK) \rightarrow (Prf = \{\pi, \psi, y\})$

1) Generate $\{p_i = \mu^i \ mod \ q\}$, $i \in D$ and compute $y = f_{\vec{A}}(\mu) \ mod \ q$, where $\vec{A} = \{0, 0, \sum_{i \in D} p_i m_{i,0}, \cdots, \sum_{i \in D} p_i m_{i,s-1}\}$.

2) Divide the polynomial $f_{\vec{A}}(x) - f_{\vec{A}}(\mu)$ with $(x - \mu)$ using polynomial long division, and denote the coefficients vector of the resulting quotient polynomial by $\vec{w} = (w_0, w_1, \cdots, w_s)$, i.e., $f_{\vec{w}}(x) \equiv \frac{f_{\vec{A}}(x) - f_{\vec{A}}(\mu)}{x - \mu}$.
With $\vec{w}$, compute $\psi = \prod_{j=0}^{s} (g^{\alpha^j})^{w_j} = g^{f_{\vec{w}}(\alpha)}$.

3) For data blocks in D that were last-modified by user $u_k, k \in C$, compute $\pi_i = e(\sigma_i, g^{\frac{\epsilon_0 R}{\epsilon_k}}) = e((u^{B_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)}), g)^{\epsilon_0 R}$.
For data blocks never modified by any group user or only modified by u_0, compute $\pi_i = e(\sigma_i, g^R) = e((u^{B_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)}), g)^{\epsilon_0 R}$.
For data blocks in D last modified by revoked users, compute $\pi_i = e(\sigma_i', (g^{\frac{\epsilon_0}{\epsilon_0+\rho}})^R) = e((u^{B_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)}), g)^{\epsilon_0 R}$, where σ_i' is the updated authentication tag of blocks that are last modified by revoked users (see Fig.3 for details).

Aggregate π_i as $\pi = \prod_{i \in D} \pi_i^{p_i}$.

Algorithm: Verify$(Prf, PK) \rightarrow$ (*VerifyRst*)

1) Compute $\eta = u^\omega$, where $\omega = \sum_{i \in D} B_i p_i$.
2) Verify the integrity of file as

$$e(\eta, \kappa_0^R) \cdot e(\psi^R, \nu \cdot \kappa_0^{-\mu}) \stackrel{?}{=} \pi \cdot e(\kappa_0^{-y}, g^R) \tag{2}$$

If Eq.2 holds, output the verification result $VerifyRst$ as $Accept$; otherwise, output $VerifyRst$ as $Reject$.

Fig. 2. Detailed construction: auditing process.

Algorithm: User Revocation – Basic

1) The master user u_0 computes $\chi = \frac{\epsilon_0+\rho}{\epsilon_k} \ mod \ q$ and sends it to the cloud, where $\rho \xleftarrow{R} Z_q^*$. u_0 also generates $g^{\frac{\epsilon_0}{\epsilon_0+\rho}}$ and sends it to valid group users and the TPA as part of the PK.
2) By receiving χ the cloud updates the authentication tags of blocks that are last modified by u_k as: $\sigma_i' = \sigma_i^\chi = (u^{B_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)})^{\epsilon_0+\rho}$.
3) The TPA and valid group users discard the public information $g^{\frac{\epsilon_0}{\epsilon_k}}$.

Fig. 3. Detailed construction: basic user revocation.

Whenever there is a user to be revoked, say $u_k, k \neq 0$, the master user u_0 and the cloud run the **User Revocation - Basic** algorithm as shown in Fig.3. In particular, all authentication tags generated by revoked users are updated so that the revoked users' secret keys are removed from the tags. Depending on how many the tags were modified by the revoked users, these update operations can be potentially communication/computation-intensive. To relieve the master user from this potential burden, our design securely offloads all tag update operations to the cloud, which is resource abundant and supports parallel processing. The master user only needs to compute two group elements in each user revocation event (see step 1 of Fig.3).

During the auditing process (as shown in Fig.2), if the set D contains blocks last modified by any revoked users, the TPA, while executing the **Challenge** algorithm, adds $(g^{\frac{\epsilon_0}{\epsilon_0+\rho}})^R$ to set X as part of the challenge message $CM = \{D, X, g^R, \mu\}$; on receiving the challenge message, the cloud generates $\pi_i = e(\sigma_i', (g^{\frac{\epsilon_0}{\epsilon_0+\rho}})^R)$ for each block in set D that is modified by a revoked user (see step 3) of the **Prove** algorithm). Finally, the TPA can verify the integrity of the challenged file by running the **Verify** algorithm.

Correctness: We analyze the correctness of our construction based on Eq.2 as:
Integrity auditing without user revocation:

$$\begin{aligned}
&= \pi \cdot e(\kappa_0^{-y}, g^R) \\
&= \prod_{i \in D} e((u^{B_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)}), g)^{R\epsilon_0} \cdot e(g^{-y}, g)^{R\epsilon_0} \\
&= \prod_{i \in D} e(u^{B_i}, g)^{R\epsilon_0} \cdot \prod_{i \in D} e(g^{f_{\vec{\beta}_i}(\alpha)}, g)^{R\epsilon_0} \cdot e(g^{-y}, g)^{R\epsilon_0} \\
&= \prod_{i \in D} e(u^{B_i}, g)^{R\epsilon_0} \cdot e(g^{f_{\vec{A}}(\alpha)}, g)^{R\epsilon_0} \cdot e(g^{-y}, g)^{R\epsilon_0} \\
&= e(u^{\sum_{i \in D} B_i}, g)^{R\epsilon_0} \cdot e(g^{f_{\vec{w}}(\alpha)}, g^{(\alpha-\mu)})^{R\epsilon_0} \\
&= e(\eta, \kappa_0^R) \cdot e(\psi^R, \nu \cdot \kappa_0^{-\mu})
\end{aligned} \tag{4}$$

Integrity auditing after user revocation:

$$\begin{aligned}
&= \pi \cdot e(\kappa_0^{-y}, g^R) \\
&= e(g^{-y}, g)^{R\epsilon_0} \cdot \prod_{i\in D, i\notin Rev} e((u^{B_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)}), g)^{R\epsilon_0} \\
&\quad \cdot \prod_{i\in D, i\in Rev} e((u^{B_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)})^{\epsilon_0+\rho}, (g^{\frac{\epsilon_0}{\epsilon_0+\rho}})^R) \\
&= \prod_{i\in D} e(u^{B_i}, g)^{R\epsilon_0} \cdot \prod_{i\in D} e(g^{f_{\vec{\beta}_i}(\alpha)}, g)^{R\epsilon_0} \cdot e(g^{-y}, g)^{R\epsilon_0} \\
&= e(u^{\sum_{i\in D} B_i}, g)^{R\epsilon_0} \cdot e(g^{f_{\vec{\omega}}(\alpha)}, g^{(\alpha-\mu)})^{R\epsilon_0} \\
&= e(\eta, \kappa_0^R) \cdot e(\psi^R, \nu \cdot \kappa_0^{-\mu}) \qquad (5)
\end{aligned}$$

From Eq.4 and Eq.5, it is easy to see that our scheme is correct.

B. Efficient Support for Multi-File Auditing

For cloud systems in which data blocks are subject to frequent modifications by group users, the TPA may need to audit data integrity often to assure data integrity. In large-scale systems, performing integrity auditing 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 if the TPA can aggregate integrity auditing operations of T files into one challenge and one verification 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 auditing solution, and enable the TPA to handle integrity auditing of T files at the cost comparable to the single file scenario. In the batch verification algorithm, **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** algorithms.

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

Batch-Prove: On receiving the challenging message $CM = \{D, X, g^R, \mu\}$, the cloud first runs the **Prove** algorithm in Fig.2 for each single file and generates $\psi_t, \pi_t, 1 \le t \le T$. Then, the cloud aggregates the proof information into two elements as $\pi = \prod_{t=1}^{T} \pi_t$ and $\psi = \prod_{t=1}^{T} \psi_t$. The cloud also computes $y = f_{\vec{A}}(\mu)\ mod\ q$ and $\vec{A} = \{0, 0, \sum_{t=1}^{T}\sum_{i\in D} p_i * m_{ti,0}, \cdots, \sum_{t=1}^{T}\sum_{i\in D} p_i * m_{ti,s_t-1}\}$. Finally, the cloud responds the TPA with proof information $Prf = \{\pi, \psi, y\}$.

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 it verifies the integrity of these T files as

$$e(\eta, \kappa_0^R) \cdot e(\psi^R, \nu \cdot \kappa_0^{-\mu}) \overset{?}{=} \pi \cdot e(\kappa_0^{-y}, g^R) \qquad (6)$$

If Eq.6 holds, the TPA outputs the verification result $VerifyRst$ as $Accept$ for these T files; otherwise, outputs $VerifyRst$ as $Reject$.

Algorithm: User Revocation – Advanced

1) The master user u_0 runs a (U,N)-Shamir Secret Sharing scheme and generates N points $(j, f(j))$ of a $U-1$ degree polynomial $f(x) = \chi + a_1x + a_2x^2 + \cdots + a_{U-1}x^{U-1}$. N points will be sent to N nodes of a cloud server.
2) To update an authentication tag σ, any U cloud nodes with the point $(j, f(j))$ on $f(x)$ compute *Lagrange basis polynomials* $L_j(x) = \prod_{0\le m\le U, m\ne j} \frac{x-x_m}{x_j-x_m}$.
3) Each cloud node will update a piece of the tag as $\sigma'_j = \sigma^{f(j)L_j(0)}$.
4) Aggregate U updated tag pieces as $\sigma' = \prod_{1\le j\le U} \sigma'_j = \sigma^{\sum_{j=0}^{U} f(j)L_j(0)} = \sigma^{\chi}$.

Fig. 4. Advanced user revocation design.

Based on above batch verification construction, we can see that the computational cost on the TPA 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.6 as follows:

$$\begin{aligned}
&e(\eta, \kappa_0^R) \cdot e(\psi^R, \nu \cdot \kappa_0^{-\mu}) \\
&\quad = e(u^{\sum_{i\in D, t=1}^{T} B_{ti}}, g)^{R\epsilon_0} \cdot e(g^{f_{\vec{A}}(\alpha) - f_{\vec{A}}(\mu)}, g)^{R\epsilon_0} \\
&\quad = \prod_{i\in D, t=1}^{T} e(u^{B_{ti}}, g)^{R\epsilon_0} \cdot \prod_{i\in D, t=1}^{T} e(g^{f_{\vec{\beta}_{ti}}(\alpha)}, g)^{R\epsilon_0} \\
&\quad\quad \cdot e(g^{-f_{\vec{A}}(\mu)}, g)^{R\epsilon_0} \\
&\quad = \prod_{i\in D, t=1}^{T} e((u^{B_{ti}} \cdot g^{f_{\vec{\beta}_{ti}}(\alpha)}), g)^{R\epsilon_0} \cdot e(g^{-f_{\vec{A}}(\mu)}, g)^{R\epsilon_0} \\
&\quad = \pi \cdot e(\kappa_0^{-y}, g^R) \qquad (7)
\end{aligned}$$

Based on Eq.7, it is easy to validate the correctness of our batch verification construction.

C. Discussion and Extension

In this section, we discuss the error detection probability of our design and the choice of set D's size in the **Challenge** algorithm. We also provide extensions to our proposed scheme in terms of reliability and error detection probability.

1) Advanced User Revocation Design: In our basic *User Revocation* algorithm, we utilize a single cloud node to update the authentication tag last updated by the revoked users. In this scenario, if the cloud node responsible for tag update is compromised due to internal errors or outside attacks, the revoked user will be able to generate valid authentication tags again. The main issue that causes such compromisation attack is the attacker can access χ that is used to update authentication tags when it compromises the cloud node. Therefore, to prevent this kind of compromisation and enhance the reliability of our design, we uniquely incorporate a (U,N)-Shamir Secret Sharing [23] technique into our design and distribute χ and the authentication tag update process to multiple cloud nodes.

Specifically, instead of sending χ to single cloud node as our basic *User Revocation* algorithm, the master user u_0 runs the (U,N)-Shamir Secret Sharing on χ and shares it to N cloud nodes as shown in the *Step 1* of Fig.4. Then, any U out of N nodes with the shared χ will be chosen to compute their own pieces of the updated authentication tag as shown in *Step 2-3* of Fig.4. Note that U cloud nodes can update their own authentication tag pieces in parallel without interaction with each other, thus it can achieve comparable realtime update performance as the update on single cloud node. Finally, the updated tag pieces from U cloud nodes will be aggregated to generate the final updated tag. In our tag aggregation process, the shared χ of each node is built into their corresponding authentication tag piece σ'_j. Therefore, even the attack can compromise the cloud node that aggregates authentication tag pieces, it cannot access the shared pieces of χ and recover it.

With our extended authentication tag update design, attackers have to compromise at least U cloud nodes with shared secrets at the same time to recover the secret χ, because χ is shared to U cloud users using Shamir Secret Sharing [23]. Particularly, the knowledge of any $U-1$ or fewer pieces leaves the secret completely undetermined as shown in Section 3.1. Compared with our basic *User Revocation* algorithm, which depends on one cloud node for revoked tag updating, our advanced algorithm significantly enhances the reliability of the scheme.

2) Error Detection Probability: As mentioned in Section 4.1, instead of choosing all the data blocks of a file to audit 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. [9], the error detection probability is $P = 1-(1-E)^d$, where E is the error rate. Therefore, if there are 1% corrupted data blocks, 460 challenge data blocks will result in 99% detection probability, and 95% detection probability only requires 300 challenge blocks, despite 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.

To achieve a high error detection probability for small data corruption rate, our design can increase the size of set D. For example, if the system requires 99% detection confidence for 0.1% data corruption rate, the size of set D can be set as 4603. In Section 6.1.1, we will show that increasing set D's size to achieve better error detection confidence has slight influence on our auditing performance, which makes our design significantly outperform ref. [21] in terms of efficiency.

V. Security Analysis

Theorem 1: If there exists a probabilistic polynomial time adversary Adv that successfully convinces the TPA to accept the fake proof information for a corrupted file with non-negligible probability, we can use Adv to construct a polynomial time algorithm B that solves the BDH problem or the t-SDH problem with non-negligible probability.

Theorem 2: If there exists a probabilistic polynomial time adversary Adv that can impersonate valid users and generate authentication tags on behalf of them, we can construct a polynomial time algorithm B that solves the CDH problem.

Due to the space limitation, we provide detailed proof of Theorem 1 and Theorem 2 in the full version of this work [26].

VI. Performance Evaluation

A. Numerical Analysis

In this section, we numerically analyze our proposed scheme and compare it with ref. [21] 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. We ignore hash operations in our evaluation, since its cost is negligible compared to EXP, MUL and Pairing operations. For example, the cost of one EXP operation on a PC can be over 100,000 times more expensive than the cost of one SHA-1 hash operation required in our design.

1) Computational Cost: As shown in Section 4, there are 6 algorithms in our scheme: $Setup$, $Challenge$, $Update$, $Prove$, $Verify$ and $User\ Revocation$. $Setup$ is a pre-processing procedure, which can be performed by group users off-line and will not influence the real-time verification performance. In the $Setup$ algorithm, the master user first needs to perform $(s+K+4)$ EXP operations to generate public keys, master keys of the system and secret keys of users, where s is the number of elements in block and K is the number of group users. To process a data file, the master user conducts $(s+2)n$ EXP and sn MUL operations for each file, where n is the number of blocks in a 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 of a file, the TPA performs the $Challenge$ algorithm to generate the challenging message CM. In CM, the selection of a constant number of random numbers with given system requirement is at a negligible cost. To generate set X in CM, $|C|$ EXP operations are required by the TPA, where $|C|$ is number of users who modify latest file blocks. Note that the set of X can be precomputed and stored by the TPA without influencing the realtile auditing performance. For instance, a 1,000 users group only requires the TPA to store 128KB for any possible elements of set X in one round integrity auditing (the same size for the auditing of multiple files). The cloud server then runs the $Prove$ algorithm with s EXP, $(s+d)$ MUL and d Pairing operations, where d is a constant number of blocks selected for challenging as discussed in Section 4.3.2. To verify the integrity of the file, the TPA only needs 6 EXP, 3 MUL and 3 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 operation compared 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 operation. Consequentially, the TPA only needs 6 EXP, 3 MUL and 3 Pairing operations to perform multiple file checking, the cost of which is the same as the single file scenario. To revoke a group user, only one EXP operation is required for the master user; the cloud server needs Y EXP operations to update the

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

TABLE II

REALTIME AUDITING 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, λ IS THE SIZE OF SECURITY PARAMETER 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

	Single Task		Multiple Tasks		User Revocation
	Comp.Cost (User)	Comm.Cost	Comp.Cost (User)	Comm.Cost	Comp.Cost (Cloud)
Ref.[21]	$(d+\|C\|)$EXP $+(d+2\|C\|)$MUL $+(\|C\|+1)$Pairing	$d*index+2\|C\|\|G\|$	$(d+\|C\|)T$ EXP $+(d+2\|C\|)$T MUL $+(\|C\|+1)$T Pairing	$dT*index+2\|C\|T\|G\|$	Y Exp
Our Scheme	6 EXP + 3 MUL + 3 Pairing	$d*(index+\|log\|)$ $+(\|C\|+3)\|G\|+2\lambda$	6 EXP + 3 MUL + 3 Pairing	$d*(index+T*\|log\|)$ $+(\|C\|+3)\|G\|+2\lambda$	Y EXP+Y Pairing

corresponding tags, where Y is the number of data blocks last modified by the revoked user. Considering our *Advanced User Revocation* design, U cloud nodes will perform UY EXP and UY MUL operations in parallel to update authentication tags last modified by revoked users.

We now compare our proposed scheme with the existing scheme [21] and summarize the result in Table II. To verify the integrity of a single file, ref. [21] costs the TPA $(d+|C|)$EXP$+(d+2|C|)$MUL$+(|C|+1)$Pairing operations. Note that, the value of d will increase significantly when users want to increase the error detection probability, and thus limiting the auditing performance of the TPA. For example, in order to achieve 99% confidence to detect error of a file with 0.1% data error rate, ref. [21] needs to challenge d data blocks and d should satisfy $1-(1-0.001)^d \geq 0.99$ as we discussed in Section 4.3.2. In this case, d will be larger than 4603. Differently, our design makes the cost of the TPA independent to the number of challenge blocks (d) and group users ($|C|$). Considering the integrity auditing of multiple files, the cost of the TPA in ref. [21] is linear to the number of files. In contrast, our scheme can aggregate operations brought by multiple files and make the cost similar to the single file scenario, and thus outperforms ref. [21].

2) Communication Cost: In our scheme, the communication cost for data integrity auditing mainly comes from the log records, challenging message CM and the proof information Prf. To generate the challenging message, the TPA will require d log records with size $d*|log|$ (Typically, the size of a log record is 16 bytes). The size of the $CM=\{D,X,g^R,\mu\}$ is $d*index+(|C|+1)|G|+\lambda$ bits, where $index$ is the size of block index and $|G|$ is the size of a group element. If there are some blocks last modified by revoked users, one more group element of size $|G|$ will be included in CM. For the proof information $Prf=\{\pi,\psi,y\}$, there are two group elements and the result of a polynomial with $2|G|+\lambda$ bits. Therefore, the total communication cost of an integrity verification task is $d*index+(|C|+3)|G|+2\lambda$ bits. 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 scheme only requires the master user to send one group element to the cloud and add one group element to the public keys.

Now, we compare our proposed scheme with the existing scheme [21] and summarize the result in Table II. In ref. [21] the communication cost for single verification task is $d*(index+|log|)+2|C||G|$ bits which is attributed to the challenge and proof generation processes and is comparable to our scheme. However, when processing multiple files together, the size of proof information in ref. [21] will increase linearly to the number of files as shown in Table II, where T is the number of files for verification. Differently, our batch verification design omits this kind of linear growth for group element with information aggregation. As the size of a log record is only $\frac{1}{8}$ of a group element, our scheme can save more than 80% communication cost compared with ref. [21]. In order to revoke a user, although ref. [21] has a similar cost to our scheme, their user revocation solution may allow the revoked users to impersonate valid users.

B. Experimental Results

To evaluate the performance of our proposed scheme, we fully implemented our proposed scheme on Amazon EC2 cloud using JAVA with JAVA Pairing-Based Cryptography Library (jPBC) [27]. On the cloud server, we deploy nodes running Linux with 8-core CPU and 32GB memory. The verifier is a desktop running Linux with 3.4GHz Intel i7-3770 CPU and 16GB memory, or a ASUS Android Iconia A200 Tablet with 1-GHz Nvidia Tegra 2 Dual Core Mobile CPU and with 1GB 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. We set size of each data block as $4KB$, which is consistent with the ref. [9], [28]. In order to verify the file size's influence on the auditing performance, we change it from 4MB to 400MB in our experiments, which contains 1,000 blocks to 100,000 blocks respectively. In order to compare our scheme with ref. [21], we implement the scheme proposed in ref. [21] with the same experiment environment. 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.

1) System Setup: We first evaluate the generation of public keys, master keys and secret keys for the system. Our result in Fig.5(a) indicates that the key generation cost is proportional to the group size, since the master user needs to generate secret keys for each group user separately. To show the performance

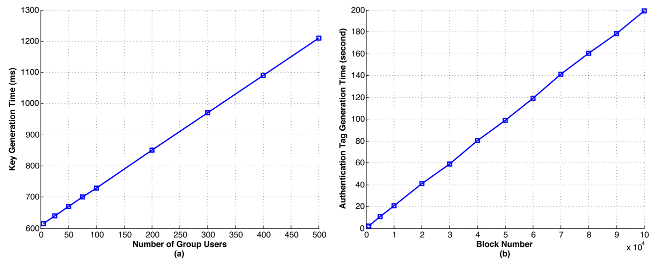

Fig. 5. (a) Key generation time. (b) Authentication tag generation time.

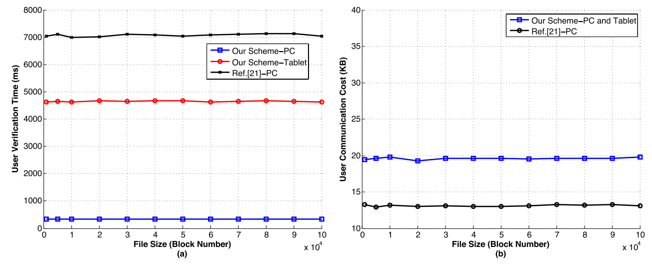

Fig. 7. (a) User verification time on different file size. (b) Communication cost on different file size.

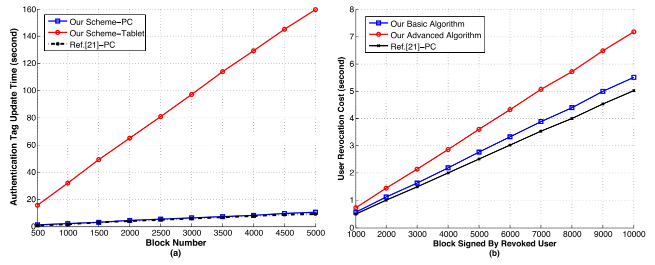

Fig. 6. (a) Block update cost. (b) User revocation cost on cloud.

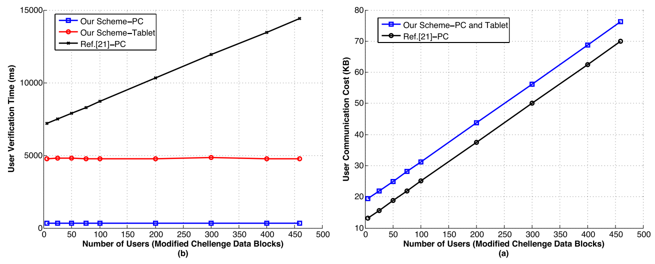

Fig. 8. (a) User verification time on different number of users. (b) Communication cost on different number of users.

of authentication tag generation, we vary the number of blocks in the file is from 1000 to 100,000. As shown in Fig.5(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 auditing.

2) 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 modified blocks, since it has to generate a new authentication for each block.

To revoke a user, the computational cost and communication cost are required for group users and the TPA are negligible in our scheme as discussed in Section 6.1.1 and Section 6.1.2. The cloud server updates the corresponding authentication tags when a user revocation occurs. Fig.6(b) shows that our tag update performances of the *Basic* User Revocation and the *Advanced* User Revocation are comparable on cloud server. This is because the cloud nodes involved for tag update procedure in our advanced algorithm perform tasks in parallel, and the only additional cost compared with our basic algorithm is the final tag aggregation process. Although our advanced user revocation algorithm requires more cost and cloud node resources, it achieves better reliability for the system as we discussed in Section 4.3.1. In practical deployment, the parameters can be selected to trade one for the other (i.e., cost versus reliability) based on the system's actual requirement.

Fig.6(a)-(b) show that our scheme is comparable to ref. [21] in terms of performance on block update and user revocation.

3) Real-Time Auditing – Single File: As discussed in Section 4.3.2, we set the number of challenge blocks as 460 in our experiments to achieve 99% error detection probability. We first measure the auditing performance of our design in terms of file size. Fig.7(a) and Fig.7(b) show that the computational cost and communication cost on the TPA side is constant versus file size, i.e., the file size has no influence on the TPA's cost, which is consistent with our previous analysis in Section 6.1.1. Fig.7(a) and (b) also indicate that, although ref. [21] has comparable communication cost to our scheme, its computational cost on the TPA is tens that of our scheme. This is because ref. [21] requires number of exponentiation operations and multiplication operations on Group linear to the number of challenge blocks (460 in our setting), which are constant and only 6 and 3 in our scheme respectively.

We now evaluate the influence of the number of users on the performance of our design. As shown in Fig.2, the computational/communication costs of our design is related to the number of users who last modified the challenge data blocks in set D. In the best case, none of the group users ever modified these data blocks; in the worst case, however, every data block in D is modified by a different group user. In our experiment, we vary the number of such group users from 0 to 460 (recall that we set the number of challenge blocks as 460 in our experiments) and evaluate its impact on system performance. Fig.8(a) shows that the computational cost of the TPA is constant and independent to the number of users who last modified the challenge blocks. This is because our design can transform authentication tags under different users' secrets to the same format during the auditing process, which enables the aggregation of computational tasks into a constant number. However, the communication cost increases proportionally to the number of users who last modified the challenge blocks as shown in Fig.8(b). This is because the TPA needs to add a group element in the challenge message for each user who last modified the challenge blocks. Note that in the worst case (i.e., 460 users modified the 460 challenge data blocks in D), the communication cost of our design is only 76.71KB, which can be efficiently transmitted via most of today's communication networks.

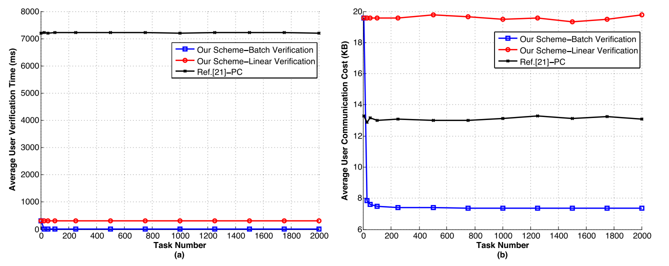

Fig. 9. (a) Average user verification time on different task number. (b) Communication cost on different task number.

Ref. [21] has a similar communication cost to ours. However, its computational cost on the TPA is proportional to the number of users who last modified the challenge blocks, which can be several magnitudes higher than our scheme as shown in Fig.8(a). These experimental results are also consistent with our analysis in Section 6.1.1. Considering the limited computational/communication costs, we believe that our design allows efficient auditing via resource-less-abundant devices such as smartphones, which is not necessarily true for ref. [21].

4) Real-Time Auditing – Multiple Files: To show the benefits of our batch auditing design for multiple-file scenario, we change the number of simultaneous auditing tasks from 16 to 2000. Here, we measure the average integrity auditing cost per file and compare our batch auditing design with straightforward auditing (i.e., processing the tasks one by one). As shown in Fig.9(a), compared with straightforward auditing, our batch auditing achieves the average computational cost per file on the TPA is nearly inversely proportional to the task number. This is because our design allows the most expensive auditing operations to be aggregated into one. Each additional auditing task only introduces a constant number of hash operations, which are extremely efficient. Fig.9(b) shows that our batch auditing design also reduces the average communication cost by about 60%. Therefore, our batch auditing design significantly enhances scalability of our scheme in terms of the number of tasks. Since ref. [21] does not support batch auditing, its computational performance is similar to the straightforward verification as shown in Fig.9(a) and its communicational cost almost doubles that of our batch verification. Thus, our batch verification design significantly outperforms ref. [21] in terms of multiple tasks processing.

VII. Applications

A potential application of our design is Version Control Systems (VCS) [4], [5], which are widely utilized in automating the management of source code, documentation and configuration files for software development. In a VCS, a number of developers work as a team and develop on shared source codes. To record all changes of development files from version to version, a VCS will store the first version of a file entirety and its subsequent versions of stored with the difference from the immediate previous version. In order to utilize our proposed scheme for the integrity auditing for a VCS, we treat the developers as a group and let the developer who creates the initial repository as the master user u_0. u_0 will setup the auditing system and generate keys with our $Setup$ algorithm. As the a file in VCS changes from one version to another by appending subsequent difference to the original file, we can treat them as a whole file F and the commit operation as update operation of adding blocks. When a developer makes changes to files and commits them to the server, he/she (i.e., his/her computer) will also generate authentication tags for modified blocks with our $Update$ algorithm. When developers want to audit the integrity of files and their changes on the version control server, they can first perform our $Challenge$ algorithm to enforce the server to generate corresponding proof information using our $Prove$ algorithm. Then, by running our $Verify$ algorithm with the proof information and public keys, the integrity can be checked with high error detection probability. As our design efficiently supports batch auditing, we can audit all development files at the same time to save cost. Thus, our scheme can be easily applied to existing VCSs to efficient support integrity assurance without changing their original design.

VIII. Related Work

The problem of data integrity auditing in cloud have been extensively studied in past years by a number of Proof of Retrievability (POR) and Proof of Data Possession (PDP) schemes [8]–[22]. In ref. [8], [9], concepts of POR and PDP were first proposed separately using RSA-based homomorphic authentication tags. The efficiency of POR scheme was later enhanced by Shacham and Waters [11] based on the BLS (Boneh-Lynn-Shacham) signature [29]. To further enhance the efficiency of data integrity auditing, batch integrity auditing was introduced by Wang et al. [14]. Recently, Xu and Chang [16] and Yuan and Yu [20] proposed private and public POR schemes respectively with constant communication cost by using a nice algebraic property of polynomial.

To support dynamic operations in verification, Ateniese et al. [10] proposed another private PDP scheme with symmetric encryption. A Public integrity auditing with dynamic operations is introduced by Wang et al. [13] based on the Merkle Hash Tree. Based on the rank information, Erway et al. also achieve the dynamic PDP. Zhu et al. [15] later utilized the fragment structure to save storage overhead of authentication tags with the support of dynamic data. A private POR scheme with the support of dynamic data is recently proposed by Cash et al. [22] by utilizing Oblivious RAM.

Although many efforts have been made to guarantee the integrity of data on remote server, most of them only consider single data owner who has the system secret key and is the only party allowed to modify the shared data on cloud. In order to improve the previous works to support multiple writers, Wang et al. [18] 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. [21] enhanced their previous public integrity verification scheme with the support of user revocation. However, if the cloud node responsible for tag update is compromised during user revocation process, attackers can discover the secret keys of all other valid users. What is more, verification cost of the TPA (can also be users) in ref. [21] is significantly influenced by the error detection probability requirement and is also linear to the number of data modifier. Batch verification is not supported in their design. Therefore, this scheme is limited in its scalability.

IX. Conclusion

In this paper, we propose a novel data integrity auditing scheme that supports multiple writers for cloud-based data sharing services. Our proposed scheme is featured by salient properties of public integrity auditing and constant computational cost on the user side. We achieve this through our innovative 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 aggregate 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 with an efficient basic design and an advanced design with enhanced reliability. Last but not least, our proposed scheme allows aggregation of integrity auditing operations for multiple tasks (files) through our batch integrity auditing technique. We provide practical application scenarios of proposed scheme. Extensive numerical analysis and real-world experiments validate the performance of our scheme.

References

[1] J. Yuan and S. Yu, "Efficient public integrity checking for cloud data sharing with multi-user modification," in *Proc. 33rd Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Toronto, ON, Canada, Apr./May 2014, pp. 2121–2129.

[2] *Dropbox for Business*. [Online]. Available: https://www.dropbox.com/business, accessed Apr. 24, 2015.

[3] SugarSync, Inc. *Business—SugarSync*. [Online]. Available: https://www.sugarsync.com/business/, accessed Apr. 24, 2015.

[4] *Apache Subversion*. [Online]. Available: http://subversion.apache.org/, accessed Apr. 24, 2015.

[5] *Concurrent Versions System*. [Online]. Available: http://cvs.nongnu.org, accessed Apr. 24, 2015.

[6] *Amazon EC2 and Amazon RDS Service Disruption*. [Online]. Available: http://aws.amazon.com/message/65648/, accessed Apr. 24, 2015.

[7] *Dropbox Forums on Data Loss Topic*. [Online]. Available: https://www.dropboxforum.com/hc/en-us/search?utf8=%E2%9C%93&query=data+loss&commit=Search

[8] A. Juels and B. S. Kaliski, Jr., "PORs: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 584–597.

[9] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 598–609.

[10] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. 4th Int. Conf. Secur. Privacy Commun. Netw. (SecureComm)*, 2008, Art. ID 9.

[11] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. 14th Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, Melbourne, Vic., Australia, 2008, pp. 90–107.

[12] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," in *Proc. 17th IEEE Int. Workshop Quality Service (IWQoS)*, Charleston, SC, USA, Jul. 2009, pp. 1–9.

[13] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proc. 14th Eur. Conf. Res. Comput. Secur.*, Saint-Malo, France, 2009, pp. 355–370.

[14] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proc. 29th IEEE Int. Conf. Comput. Commun. (INFOCOM)*, San Diego, CA, USA, Mar. 2010, pp. 1–9.

[15] 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 *Proc. ACM Symp. Appl. Comput. (SAC)*, 2011, pp. 1550–1557.

[16] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *Proc. 7th ACM Symp. Inf., Comput., Commun. Secur. (ASIACCS)*, Seoul, Korea, 2012, pp. 79–80.

[17] H. Wang, "Proxy provable data possession in public clouds," *IEEE Trans. Services Comput.*, vol. 6, no. 4, pp. 551–559, Oct./Dec. 2013.

[18] B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput. (CLOUD)*, Washington, DC, USA, Jun. 2012, pp. 295–302.

[19] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 362–375, Feb. 2013.

[20] J. Yuan and S. Yu, "Proofs of retrievability with public verifiability and constant communication cost in cloud," in *Proc. Int. Workshop Secur. Cloud Comput. (Cloud Computing)*, 2013, pp. 19–26.

[21] B. Wang, B. Li, and H. Li, "Public auditing for shared data with efficient user revocation in the cloud," in *Proc. 32nd IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Turin, Italy, Apr. 2013, pp. 2904–2912.

[22] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious RAM," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 7881, T. Johansson and P. Q. Nguyen, Eds. Berlin, Germany: Springer-Verlag, 2013, pp. 279–295.

[23] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[24] *Amazon EC2 Cloud*. [Online]. Available: http://aws.amazon.com/ec2/, accessed Apr. 24, 2015.

[25] D. Eastlake and P. Jones. *US Secure Hash Algorithm 1 (SHA1)*. [Online]. Available: http://www.ietf.org/rfc/rfc3174.txt?number=3174, accessed Apr. 24, 2015.

[26] *Public Integrity Auditing for Dynamic Data Sharing With Multi-User Modification*. [Online]. Available: http://ualr.edu/jxyuan/papers/IEEE-TIFS-15.pdf, accessed Apr. 24, 2015.

[27] A. De Caro and V. Iovino, "jPBC: Java pairing based cryptography," in *Proc. 16th IEEE Symp. Comput. Commun. (ISCC)*, Corfu, Greece, Jun./Jul. 2011, pp. 850–855.

[28] J. Yuan and S. Yu, "Secure and constant cost public cloud storage auditing with deduplication," in *Proc. 1st IEEE Conf. Commun. Netw. Secur. (CNS)*, Washington, DC, USA, Oct. 2013, pp. 145–153.

[29] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Proc. 7th Int. Conf. Theory Appl. Cryptol. Inf. Secur. (ASIACRYPT)*, 2001, pp. 514–532.

Jiawei Yuan received the B.S. degree from the University of Electronic Science and Technology of China, in 2011. He is currently pursuing the Ph.D. degree with the University of Arkansas at Little Rock. His research interests are in the areas of cloud computing and network security, with a current focus on securing data and computation outsourced into the cloud.

Shucheng Yu (S'07–M'10) received the Ph.D. degree in electrical and computer engineering from the Worcester Polytechnic Institute. He joined the Computer Science Department, University of Arkansas at Little Rock, in 2010, as an Assistant Professor. His current research interests include network security, applied cryptography, secure data services in cloud computing, attribute based cryptography, and security and privacy protection in cyber physical systems. He is a member of the Association for Computing Machinery.