# Provably-Secure Public Auditing with Deduplication

**Dongmin Kim and Ik Rae Jeong**
CIST (Center for Information Security Technologies), Korea University
Anam-dong, Seongbuk-gu, Seoul 136-713, Korea
[e-mail: kkomang03@korea.ac.kr, irjeong@korea.ac.kr]
*Corresponding author: Ik Rae Jeong

*Received August 29, 2016; revised December 5, 2016; revised January 21, 2017; accepted February 14, 2017; published April 30, 2017*

## Abstract

With cloud storage services, users can handle an enormous amount of data in an efficient manner. However, due to the widespread popularization of cloud storage, users have raised concerns about the integrity of outsourced data, since they no longer possess the data locally. To address these concerns, many auditing schemes have been proposed that allow users to check the integrity of their outsourced data without retrieving it in full. Yuan and Yu proposed a public auditing scheme with a *deduplication* property where the cloud server does not store the duplicated data between users. In this paper, we analyze the weakness of the Yuan and Yu's scheme as well as present modifications which could improve the security of the scheme. We also define two types of adversaries and prove that our proposed scheme is secure against these adversaries under formal security models.

***Keywords:*** Public auditing, deduplication, data integrity, tag-unforgeability, proof-unforgebility

## 1. Introduction

Cloud storage service allows users to outsource their data to a cloud server and access it whenever necessary. Due to their widespread availability and convenience, cloud storage services have become common in recent years, where diverse commercial products have been released, such as Dropbox, Google Drive, and iCloud, which are used by numerous people and enterprises [1-3].

Cloud storage services are convenient for data management and they have reduced the burden of the management, but they also involve new security threats related to outsourced data in terms of the integrity of the data. Data are stored remotely in a cloud server, so users are unable to be certain that their data are stored in full without any modifications. Thus, after the users outsource their data to the server, the server may delete or not store the data in full in order to make a profit by saving on storage costs. In order to detect such misbehaviors by the storage service provider, many studies have proposed methods for integrity checking without downloading the data in full [4-11].

Ateniese et al. first proposed the concept of *public verifiability* and a public auditing scheme called provable data possession, which uses the RSA-based homomorphic verifiable tag to generate a tag for each block of data [4]. The tag allows users to check whether the server contains the data blocks, but this incurs high costs in terms of server computation and communication. Juels and Kaliski also proposed a new scheme called Proof of Retrievability (POR) which allows the integrity of the remotely stored data to be checked as well as the retrievability of the data by applying the error-correcting codes [5]. However, this method also incurs high computational overheads on the client side. Shacham and Waters proposed a more efficient and compact POR scheme [7] that uses the BLS short signature, they proved that this scheme is secure under the security model defined by Juels and Kaliski [5], but it does not ensure the confidentiality of the file blocks. Thus, the user data may be revealed to a public verifier; called a third party auditor (TPA); who checks the integrity of outsourced data in the cloud on behalf of the users. Wang et al. first proposed a *privacy-preserving* public auditing scheme, which ensures the confidentiality of the file blocks [11]. Subsequently, many solutions have been proposed with various properties, such as identity privacy, traceability, and key updates, in addition to public verifiability and privacy-preserving property [12-25].

Zheng and Xu proposed a public auditing scheme with deduplication, which makes the server store only a single copy of each file (or block) to save the storage costs in a legitimate manner [26]. However, Shin et al. showed that the scheme proposed by Zheng and Xu is not secure against a weak key attack and they modified it to make it secure against this type of attack [27]. Yuan and Yu also proposed a public auditing scheme with deduplication, which has constant communication and computation costs of TPA [28]. Li et al. proposed two secure systems called SecCloud and SecCloud+ to ensure data integrity and deduplication in the cloud [29]. SecCloud+ allows integrity auditing and data deduplication on the encrypted data because of the deterministic encryption property in convergent encryption. Alkhojandi and Miri also proposed privacy-preserving public auditing scheme with deduplication, which ensures that TPA cannot learn any information about the stored data [30]. Unfortunately, most schemes, except for the scheme proposed by Yuan and Yu, are not efficiently constructed since computation costs of TPA are affected by the number of challenged blocks. In other words, they have computation costs of $O(k)$ when performing the auditing process, where $k$ is the number of challenged blocks. Recently, He et al. introduced the concept of deduplicatable dynamic proof of storage and proposed an efficient construction that uses homomorphic authenticated tree, but their scheme does not ensure public verifiability [31]. For that reason, we mainly analyze Yuan and Yu's scheme which is the most efficient in terms of computation and communication costs of TPA.

In the paper, we show that Yuan and Yu's scheme is not secure against a malicious server and propose a new secure scheme with constant communication and computation costs. We also define two types of adversaries and prove that our scheme is secure against these adversaries under formal security models.

The rest of paper is organized as follows. In Section 2, we describe the formal security models and review some preliminaries. We demonstrate the insecurity of Yuan and Yu's scheme in Section 3. In Section 4, we construct our scheme and prove its security. We give our conclusion in Section 5.

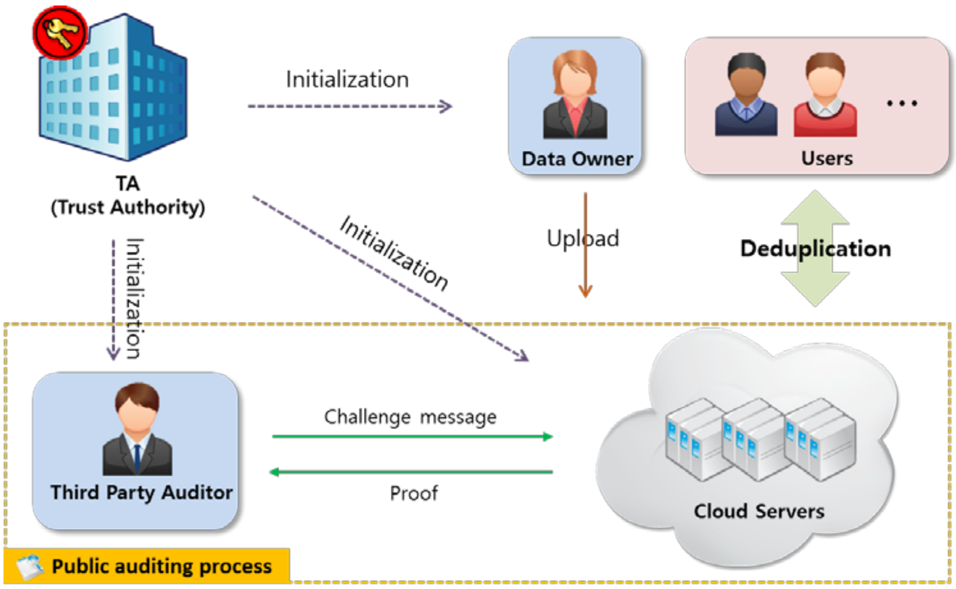

**Fig. 1.** System model for public auditing with deduplication

## 2. Models and Preliminaries

### 2.1 System Model

We consider a system model with four entities, i.e., trust authority (TA), data owner, cloud server, and the third party auditor (TPA) as shown in **Fig. 1**. TA generates a master key and a set of public keys for the system, but does not participate in any other process. A data owner has a collection of data and outsources them to the cloud server with authenticated tags corresponding to each block of data files. When the data owner wishes to check the integrity of their outsourced data, TPA performs the integrity checking on behalf of the data owner. To check the integrity of the data, TPA sends a challenge message to the cloud server. After receiving the challenge message, the cloud server generates a valid proof for the selected blocks and responds to TPA. Next, TPA verifies the validity of the received proof. We define the algorithms for our public auditing scheme as follows.

- **KeyGen** takes a security parameter and returns the master key $MK$, a set of public keys $PK$ for the system and a secret key $SK$ for a user.
- **TagGen** takes as inputs the set of public keys $PK$, a secret key $SK$ of a user, and a file $F = (m_1, ..., m_n)$, where $n$ is the number of the file blocks. It outputs tags $\{\sigma_i\}_{1 \le i \le n}$ for the file blocks.
- **Challenge** takes the set of public keys $PK$ as an input and returns a challenge message $CM$.
- **Prove** takes as inputs the set of public keys $PK$, the challenge message $CM$, and the pairs of message and tag $\{m_i, \sigma_i\}_{1 \le i \le n}$. It outputs a proof $P$ for the challenge message $CM$.
- **Verify** takes as inputs the set of public keys $PK$, the challenge message $CM$, and the proof $P$. It outputs $Accept$ or $Reject$.
- **Deduplication** is an interactive protocol between a user and the cloud server for verifying that the user and the cloud have the same data.

## 2.2 Security Model

In this section, we first introduce our new security models in the public auditing scheme. We assume that there are two types of adversaries in our public auditing scheme, i.e., Type-I adversary $A_1$ and Type-II adversary $A_2$. $A_1$ can acquire valid pairs of message and tag, and generate a forged pair of message and tag without knowing the secret key of the user. $A_2$ can also acquire valid pairs of message and tag, and generate a forged proof for the challenged message without tag-forgery. We define new security models, tag-unforgeability and proof-unforgeability, using the game between a challenger and Type-I adversary $A_1$ and Type-II adversary $A_2$, respectively, as follows,.

**Tag-Unforgeability**. If a valid tag is not able to be generated by Type-I adversary $A_1$, then we consider that a public auditing scheme satisfies tag-unforgeability. It is defined using the following game between a challenger $C$ and Type-I adversary $A_1$.

1. The challenger $C$ runs KeyGen algorithm to generate the master key $MK$, a secret key $SK$, and a set of public keys $PK$, and then gives $PK$ to Type-I adversary $A_1$.

2. Type-I adversary $A_1$ receives a set of public keys $PK$ and is allowed to make hash queries and TagGen queries, adaptively. For TagGen queries, Type-I adversary $A_1$ sends $(i, m_i)$, where $i$ is the index of the message block to be signed in the file $F$, to the challenger $C$. The challenger $C$ gives $\sigma_i = TagGen(SK, m_i)$ to $A_1$.

3. Finally, Type-I adversary $A_1$ outputs a forged pair of message and tag $(i, m^*, \sigma^*)$.

If the pair $(i, m^*, \sigma^*)$ is a valid message and tag pair and $m^*$ has not been queried before, then Type-I adversary $A_1$ wins the game.

**Definition 1.** Type-I adversary $A_1(t, q_h, q_t, \varepsilon)$-breaks the public auditing scheme, if $A_1$ runs within a time of at most $t$, makes at most $q_h$ hash queries, and $q_t$ TagGen queries, and wins the game with the probability $\varepsilon$. A public auditing scheme is tag-unforgeable, if no tag-forger $(t, q_h, q_t, \varepsilon)$-breaks the scheme with a non-negligible probability $\varepsilon$.

**Proof-Unforgeability**. If Type-II adversary $A_2$ is not able to generate a valid proof for the blocks that are modified or deleted by Type-II adversary $A_2$, then we say that a public auditing scheme satisfies proof-unforgeability. It is defined using the following game between a challenger $C$ and Type-II adversary $A_2$.

1. The challenger $C$ runs KeyGen algorithm to generate the master key $MK$, a secret key $SK$, and a set of public keys $PK$.

2. Type-II adversary $A_2$ receives a set of public keys $PK$ and is allowed to make hash queries and TagGen queries, adaptively. For TagGen queries, Type-II adversary $A_2$ sends $(i, m_i)$, where $i$ is the index of the message block to the challenger $C$. The challenger $C$ responds with $\sigma_i = TagGen(SK, m_i)$.

3. The challenger $C$ generates the challenge message $CM = Challenge(PK)$, and sends it to Type-II adversary $A_2$.

4. Finally, Type-II adversary $A_2$ outputs a forged proof $P^*$.

If the equation $Verify(P^*) = Accept$ holds and the challenge message $CM$ includes the index that is not queried in TagGen queries, then Type-II adversary $A_2$ wins the game.

**Definition 2.** Type-II adversary $A_2(t, q_h, q_t, \varepsilon)$-breaks the public auditing scheme, if $A_2$ runs within a time of at most $t$, makes at most $q_h$ hash queries, and $q_t$ TagGen queries, and wins the game with the probability $\varepsilon$. A public auditing scheme is proof-unforgeable, if no proof-forger $(t, q_h, q_t, \varepsilon)$-breaks the scheme with a non-negligible probability $\varepsilon$.

## 2.3 Preliminaries

In this section, we describe some preliminaries which are widespread used in the field of cryptography.

**Bilinear Group.** Let groups $G$ and $G_1$ be multiplicative cyclic groups with prime order $q$. A function $e : G \times G \to G_1$ is a bilinear map, if it satisfies the followings.

1. We have $e(g^a, g^b) = e(g, g)^{ab}$ for all $g \in G$, $a, b \in \mathbb{Z}_q^*$.
2. If $g$ is a generator of group $G$, then $e(g, g)$ is a generator of group $G_1$.
3. It is easy to compute $e(g^a, g^b)$ for all $g \in G$.

**Table 1.** Notaions in our scheme

| Notations | Descriptions |
|---|---|
| $q$ | a prime order |
| $G, G_1$ | multiplicative cyclic groups with prime order $q$ |
| $e: G \times G \to G_1$ | a bilinear map |
| $(spk, ssk)$ | a key pair of the signing algorithm |
| $g, u$ | generators of the group $G$ |
| $F' = \{m_{i0}, \cdots, m_{i(s-1)}\}_{1 \le i \le n}$ | a file which is applied erasure code to a file $F$ |
| $\tau$ | a file tag $\tau = name \| n \| Sign_{ssk}(name \| n)$ |
| $name$ | a file name |
| $n$ | the number of blocks in the file $F'$ |
| $\sigma_i$ | an authenticated tag for the block $m_i$ |
| $f_{\vec{\beta}_i}$ | a polynomial with a coefficient vector $\vec{\beta}_i$ |
| $\vec{\omega} = (\omega_0, \omega_1, \ldots, \omega_s)$ | a coefficient vector of the resulting quotient polynomial |
| $[1, n]$ | the numbers in a range $1 \le i \le n$ |
| $H(\cdot), \widetilde{H}(\cdot)$ | collision resistance hash functions |

**Computational Diffie-Hellman (CDH) Assumption.** Given a tuple $g, g^a, g^b \in G$, an algorithm $A$ tries to compute $g^{ab} \in G$. We assume that no algorithm $A$ has a non-negligible probability $\varepsilon$ such that

$$Pr[A(g, g^a, g^b) = g^{ab}] \ge \varepsilon, \tag{1}$$

where the probability is selected over the random choice of $g \in G$, the random choice of $a, b \in \mathbb{Z}_q^*$, and the random bits of $A$.

**Divisible Computational Diffie-Hellman (DCDH) Assumption** [32]. Given a tuple $g, g^a, g^b \in G$, an algorithm $A$ tries to compute $g^{a^{-1}b} \in G$. We assume that no algorithm $A$ has a non-negligible probability $\varepsilon$ such that

$$Pr[A(g, g^a, g^b) = g^{a^{-1}b}] \ge \varepsilon, \tag{2}$$

where the probability is selected over the random choice of $g \in G$, the random choice of $a, b \in \mathbb{Z}_q^*$, and the random bits of $A$.

## 3. Review and Analysis of the Yuan and Yu's scheme

In this section, we briefly review the Yuan and Yu's scheme in [28]. The details of the algorithms are as follows.

**KeyGen**. Let $G$ and $G_1$ be multiplicative cyclic groups with prime order $q$, and $g$ and $u$ be generators of the group $G$. Let $e: G \times G \to G_1$ be a bilinear map and $\widetilde{H}(\cdot): \{0,1\}^* \to Z_q^*$ be a collision resistance hash function. TA randomly chooses $\alpha \leftarrow \mathbb{Z}_q^*$ and computes $\{g^{\alpha^j}\}_{0\le j\le s+1}$ as public keys for the system. The data owner generates a key pair $(spk, ssk)$ for the signing algorithm [33], where $(spk, ssk) \leftarrow Sign.KeyGen()$. The data owner also chooses $x \leftarrow \mathbb{Z}_q^*$, and calculates $v = g^{\alpha x}$ and $\kappa = g^x$. The master key is $MK = \alpha$, the set of public keys is $PK = (g, u, q, v, \{g^{\alpha^j}\}_{0\le j\le s+1}, \kappa)$, and the set of secret keys is $SK = (x, ssk)$.

**TagGen.** Let $F'$ be a file obtained by applying an erasure code to a file $F$, where $F' = \{m_{i0}, \cdots, m_{i(s-1)}\}_{1\le i\le n}$. The data owner computes a file tag $\tau$ and an authenticated tag $\sigma_i$ for each block $m_i$ as

$$\sigma_i = (u^{\widetilde{H}(name\|i)} \cdot \prod_{j=0}^{s-1} g^{m_{ij}\alpha^{j+2}})^x = (u^{\widetilde{H}(name\|i)} \cdot g^{f_{\vec{\beta}_i}(\alpha)})^x, \tag{3}$$

where $\tau = name \| n \| Sign_{ssk}(name \| n)$, $\vec{\beta}_i = (0, 0, m_{i0}, m_{i1}, ..., m_{i(s-1)})$ and $f_{\vec{\beta}_i}$ is a polynomial with a coefficient vector $\vec{\beta}_i$. Then the data owner uploads the file tag $\tau$ and the file $F' = \{m_i\}_{1\le i\le n}$ with the corresponding authenticated tags $\{\sigma_i\}_{1\le i\le n}$ to the server.

**Challenge.** To check the integrity of $F'$, TPA first retrieves the file tag $\tau$ and verifies its validity. If $\tau$ is valid, then TPA obtains $name \| n$ from $\tau$; otherwise, TPA sends $fail$ to the server. TPA randomly chooses a random number $r \leftarrow \mathbb{Z}_q^*$ and picks a $k$-elements subset $K$ of $[1, n]$. Then, TPA sends the challenge messages $(K, r)$ to the server.

**Prove.** Given the challenge message $(K, r)$, the server first computes $p_i = r^i \bmod q$ for all $i \in K$ and $y = f_{\vec{A}}(r) \bmod q$, where $\vec{A} = (0, 0, \sum_{i\in K} p_i m_{i0}, ..., \sum_{i\in K} p_i m_{i(s-1)})$. The server computes $f_{\vec{\omega}}(z) = \dfrac{f_{\vec{A}}(z) - f_{\vec{A}}(r)}{z - r}$, where $\vec{\omega} = (\omega_0, \omega_1, ..., \omega_s)$ is the coefficient vector of the resulting quotient polynomial, and $\psi = \prod_{j=0}^{s} (g^{\alpha^j})^{w_j} = g^{f_{\vec{\omega}}(\alpha)}$. The server also computes the aggregated proof $\sigma = \prod_{i\in K} \sigma_i^{p_i}$ and then sends the proof $(\sigma, \psi, y)$ to TPA.

**Verify.** After receiving the proof $(\sigma, \psi, y)$, TPA checks the integrity of the file. TPA computes $\eta = u^{\vartheta}$, where $\vartheta = \sum_{i\in K} p_i \widetilde{H}(name \| i)$, and checks the following equation

$$e(\eta, \kappa) \cdot e(\psi, v \cdot \kappa^{-r}) = e(\sigma, g) \cdot e(\kappa^{-y}, g). \tag{4}$$

If the equation holds, TPA outputs $Accept$; otherwise TPA outputs $Reject$.

**Deduplication**. Suppose that a user wants to store a file $F'$ in the server and $F'$ is stored in the server. To check that the user actually owns file $F'$, the server selects $k$ - elements the subset $K$ of $[1, n]$ and requests the corresponding file blocks. After receiving the requests $K$, the user responds with the corresponding file blocks $m_i$, for all $i \in K$. Then the server computes

$$\sigma' = \prod_{i\in K} \sigma_i, \ \eta' = \prod_{i\in K} u^{H(name\|i)}, \ \psi' = e(\prod_{j=2}^{s+1} (g^{\alpha^j})^{\beta_j}, \kappa) = e(g^{f_{\vec{B}}(\alpha)}, \kappa), \tag{5}$$

where $\vec{B} = (0, 0, \sum_{i\in K} m_{i0}, ..., \sum_{i\in K} m_{i(s-1)})$, and checks the following equation

$$e(\eta',\kappa)\cdot\psi' = e(\sigma',g). \tag{6}$$

If the equation holds, the server considers that the user owns the file $F'$.

## 3.1 Insecurity against Tag-Unforgeability

In this section, we show that the Yuan and Yu's scheme in [28] is not secure against a malicious server. Note that the schemes proposed in [17] and [28] use the same technique to generate the tags for the message, and the scheme in [17] is also not secure. Thus, we show that the malicious server can generate the valid tags for the modified messages using the valid pairs of message and tag. In their scheme, the data owner uses the same value $u$, although he generates the tag for different messages. Because of this vulnerability, the attacker can generate a valid tag for an arbitrary message without knowing the user's secret key. The detailed attack process as follows.

We assume that $\sigma_1$ and $\sigma_2$ are valid tags for the messages $m_1$ and $m_2$, respectively, where $\sigma_1 = (u^{\widetilde{H}(name\|1)}\cdot g^{f_{\vec{\beta}_1}(\alpha)})^x$ and $\sigma_2 = (u^{\widetilde{H}(name\|2)}\cdot g^{f_{\vec{\beta}_2}(\alpha)})^x$. Using the public key $PK$, public values $\widetilde{H}(name\|1)$, $\widetilde{H}(name\|2)$, and the stored original messages $m_1, m_2$, the server calculates

$$\frac{\sigma_1}{\sigma_2} = u^{(h_1-h_2)\cdot x}\cdot g^{(\sum_{j=0}^{s-1}\alpha^{j+2}(m_{1j}-m_{2j}))\cdot x}, \tag{7}$$

$$\left(\frac{\sigma_1}{\sigma_2}\right)^{\frac{1}{h_1-h_2}} = u^x\cdot g^{\frac{\sum_{j=0}^{s-1}\alpha^{j+2}(m_{1j}-m_{2j})}{h_1-h_2}\cdot x}, \tag{8}$$

$$\left(\frac{\sigma_1}{\sigma_2}\right)^{\frac{h_1}{h_1-h_2}} = u^{h_1\cdot x}\cdot g^{\frac{h_1\cdot\sum_{j=0}^{s-1}\alpha^{j+2}(m_{1j}-m_{2j})}{h_1-h_2}\cdot x}, \tag{9}$$

$$\sigma_1^* = (u^{h_1}\cdot g^{\alpha^2\cdot m_{10}^*+\cdots+\alpha^{s+1}\cdot m_{1(s-1)}^*})^x, \tag{10}$$

where $h_2 = \widetilde{H}(name\|2)$, and $m_{1j}^* = \dfrac{h_1(m_{1j}-m_{2j})}{h_1-h_2}$ for all $0\le j\le s-1$. Then the value $\sigma_1^*$ is a valid tag for the modified message $m_1^*$.

Another simple method can be used to attack the scheme. The attacker computes

$$\sigma_1^{\frac{h_2}{h_1}} = u^{h_1\cdot\frac{h_2}{h_1}\cdot x}\cdot g^{\frac{h_2}{h_1}\cdot(\sum_{j=0}^{s-1}\alpha^{j+2}m_{1j})\cdot x}, \tag{11}$$

$$\sigma_2^* = (u^{h_2}\cdot g^{\alpha^2\cdot m_{20}^*+\cdots+\alpha^{s+1}\cdot m_{2(s-1)}^*})^x, \tag{12}$$

where $m_{2j}^* = \dfrac{h_2}{h_1}\cdot m_{2j}$ for all $0\le j\le s-1$. Then the value $\sigma_2^*$ is a valid tag for the modified message $m_2^*$.

According to the attacks above, the server can generate valid tags for the modified messages without knowing the user's secret keys, and thus the malicious server can cheat TPA into believing that the stored messages are intact without any modifications or deletions.

## 4. Our Construction

First, we propose the basic public auditing scheme without deduplication and prove its security in formal security models. Next, we extend the basic public auditing scheme to a public auditing scheme with deduplication.

### 4.1 Basic Scheme without Deduplication

In this section, we describe our basic public auditing scheme. The main difference with the previous works is a modification of the tag generation method. We modify TagGen algorithm to make it secure against the vulnerability by using $u^t \cdot h_i$ instead of $u^{h_i}$. In addition, we modify all parts of each algorithm which relates to the modification of TagGen algorithm, such as the set of public keys, the challenge messages, the proofs and the verification equation. These modifications enable the scheme to maintain constant computation costs and be well simulated in formal security proofs. Our basic scheme comprises the five algorithms; KeyGen, TagGen, Challenge, Prove, and Verify. The details of the algorithms are as follows.

**KeyGen**. Let $G$ and $G_1$ be multiplicative cyclic groups with prime order $q$, and $g$ and $u$ be generators of the group $G$. Let $e: G \times G \rightarrow G_1$ be a bilinear map and $H(\cdot): \{0,1\}^* \rightarrow G_1$ be a collision resistance hash function. TA randomly chooses $\alpha \leftarrow \mathbb{Z}_q^*$ and computes $\{g^{\alpha^j}\}_{0 \le j \le s+1}$ as public keys for the system. The data owner generates a key pair $(spk, ssk)$ for the signing algorithm [33], where $(spk, ssk) \leftarrow Sign.KeyGen()$. The data owner also chooses $x, t \leftarrow \mathbb{Z}_q^*$, and calculates $v = g^{\alpha x}$, $w = g^{xt}$, and $\kappa = g^x$. The master key is $MK = \alpha$, the set of public keys is $PK = (g, u, q, v, w, \{g^{\alpha^j}\}_{0 \le j \le s+1}, \kappa)$, and the set of secret keys is $SK = (x, ssk)$.

**TagGen**. Let $F'$ be a file obtained by applying an erasure code to the file $F$, where $F' = \{m_{i0}, \cdots, m_{i(s-1)}\}_{1 \le i \le n}$. The data owner computes a file tag $\tau$ and an authenticated tag $\sigma_i$ for each block $m_i$ as

$$\sigma_i = (u^t \cdot h_i \cdot \prod_{j=0}^{s-1} g^{m_{ij} \alpha^{j+2}})^x = (u^t \cdot h_i \cdot g^{f_{\vec{\beta_i}}(\alpha)})^x, \tag{13}$$

where $\tau = name \| n \| Sign_{ssk}(name \| n)$, $h_i = H(name \| i)$, $\vec{\beta_i} = (0, 0, m_{i0}, m_{i1}, ..., m_{i(s-1)})$ and $f_{\vec{\beta_i}}$ is a polynomial with a coefficient vector $\vec{\beta_i}$. Then the data owner uploads the file tag $\tau$ and the file $F' = \{m_i\}_{1 \le i \le n}$ to the server with the corresponding authenticated tags $\{\sigma_i\}_{1 \le i \le n}$.

**Challenge**. To check the integrity of $F'$, TPA first retrieves the file tag $\tau$ and verifies its validity. If $\tau$ is valid, then TPA obtains $name \| n$ from $\tau$; otherwise, TPA sends a *fail* message to the server. TPA randomly chooses random numbers $r, R \leftarrow \mathbb{Z}_q^*$ and a $k$-elements subset $K$ of $[1, n]$, and computes $g^R$. Then, TPA sends the challenge messages $(K, r, g^R)$ to the server.

**Prove.** Given the challenge message $(K, r, g^R)$, the server first computes $p_i = r^i \bmod q$ for all $i \in K$ and $y = f_{\vec{A}}(r) \bmod q$, where $\vec{A} = (0, 0, \sum_{i\in K} p_i m_{i0}, ..., \sum_{i\in K} p_i m_{i(s-1)})$. The server calculates the polynomial $f_{\vec{\omega}}(z) = \frac{f_{\vec{A}}(z) - f_{\vec{A}}(r)}{z - r}$, where $\vec{\omega} = (\omega_0, \omega_1, ..., \omega_s)$ is the coefficient vector of the resulting quotient polynomial, and $\psi = \prod_{j=0}^{s} (g^{\alpha^j})^{w_j} = g^{f_{\vec{\omega}}(\alpha)}$. To aggregate the tags for the challenged set $K$, the server finally computes $\rho = \prod_{i\in K} h_i^{p_i}$ and $\pi = e(\prod_{i\in K} \sigma_i^{p_i}, g^R)$. Then the server sends the proof $(\pi, \psi, y, \rho)$ to TPA.

**Verify.** After receiving the proof $(\pi, \psi, y, \rho)$, TPA checks the integrity of the file. TPA computes $\eta = u^{\vartheta R}$, where $\vartheta = \sum_{i\in K} p_i$, and checks the following equation

$$e(\eta, w) \cdot e(\rho, \kappa^R) \cdot e(\psi^R, v \cdot \kappa^{-r}) = \pi \cdot e(\kappa^{-y}, g^R). \tag{14}$$

If the equation holds, TPA outputs *Accept*; otherwise TPA outputs *Reject*.

The correctness of our scheme can be proved as follows.

$$\begin{aligned}
&\pi \cdot e(\kappa^{-y}, g^R) \\
&= e(\prod_{i\in K} \sigma_i^{p_i}, g^R) \cdot e(g^{x(-y)}, g^R) \\
&= \prod_{i\in K} e(u^t \cdot h_i \cdot g^{f_{\vec{\beta}_i}(\alpha)}, g^{xR})^{p_i} \cdot e(g^{x(-y)}, g^R) \\
&= \prod_{i\in K} e((u^{tp_i} \cdot h_i^{p_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)\cdot p_i}), g^{xR}) \cdot e(g^{-f_{\vec{A}}(r)}, g^{xR}) \\
&= e(\prod_{i\in K} u^{tp_i} \cdot h_i^{p_i} \cdot g^{\sum_{i\in K} f_{\vec{\beta}_i}(\alpha)\cdot p_i - f_{\vec{A}}(r)}, g^{xR}) \\
&= e(\prod_{i\in K} u^{tp_i} \cdot h_i^{p_i} \cdot g^{f_{\vec{A}}(\alpha) - f_{\vec{A}}(r)}, g^{xR}) \\
&= e(\prod_{i\in K} u^{tp_i} \cdot h_i^{p_i} \cdot g^{f_{\vec{\omega}}(\alpha)\cdot(\alpha - r)}, g^{xR}) \\
&= e(\prod_{i\in K} u^{tp_i} \cdot h_i^{p_i}, g^{xR}) \cdot e(g^{f_{\vec{\omega}}(\alpha)R}, g^{\alpha x} \cdot g^{x(-r)}) \\
&= e(\prod_{i\in K} u^{tp_i}, g^{xR}) \cdot e(\prod_{i\in K} h_i^{p_i}, g^{xR}) \cdot e(\psi^R, v \cdot \kappa^{-r}) \\
&= e(\eta, w) \cdot e(\rho, \kappa^R) \cdot e(\psi^R, v \cdot \kappa^{-r}).
\end{aligned} \tag{15}$$

### 4.2 Security Analyses

In this section, we show that our public auditing scheme is secure against existential forgery(by a Type-I adversary and Type-II adversary) under adaptive chosen message attacks. If the public auditing scheme satisfies tag-unforgeability and proof-unforgeability, then we can say that the scheme is secure against existential forgery.

**Theorem 4.1.** Our public auditing scheme is secure against the tag-forgery(by a Type-I adversary) under chosen message attacks in the random oracle model, if the CDH assumption holds in $G$.

*Proof of Theorem 4.1*. Suppose that Type-I adversary $A_1$ is the forger that $(t, q_h, q_t, \varepsilon)$ -breaks our public auditing scheme by generating a valid pair of message and tag under user's secret key. Then we can construct an algorithm $B$ that solves the CDH problem on $G$.

Given $(g, g^a, g^b) \in G$ as an input to the CDH problem, algorithm $B$ simulates the challenger and interacts with the forger $A_1$ in the tag-unforgeability game.

1. The algorithm $B$ randomly picks $\alpha, t, z \leftarrow \mathbb{Z}_q^*$ and computes $\{g^{\alpha^j}\}_{0 \le j \le \ell+1}$. The algorithm $B$ sets $\kappa = g^x = g^a$ and $u = g^z$, and computes $v = g^{x\alpha} = (g^a)^\alpha$ and $w = g^{xt} = (g^a)^t$. The algorithm $B$ gives $PK$ to the forger $A_1$.

2. The algorithm $B$ simulates the hash oracles and the *TagGen* oracle as follows.
   *(For hash oracle $H$ )* At any time the forger $A_1$ is able to query the random hash oracle $H$ for $name \| i$. After receiving the hash queries, the algorithm $B$ stores $Tab1 = (i, \delta_i, c_i, h_i)$ as generated below. $Tab1$ is initially empty. When the forger $A_1$ queries $name \| i$, the algorithm $B$ responds as follows.
   (a) If $i \notin Tab1$, the algorithm $B$ flips a random coin $c_i \in \{0,1\}$ with a probability $Pr[c_i = 0] = 1/(q_t + 1)$, where $q_t$ is the maximum number of TagGen queries. The algorithm $B$ picks a random number $\delta_i \in \mathbb{Z}_q^*$ and computes the hash value $h_i = (g^b)^{(1-c_i)} \cdot g^{\delta_i}$. Then the algorithm $B$ updates the tuple $Tab1 = (i, \delta_i, c_i, h_i)$ and responds with $H(name \| i) = h_i$ to the forger $A_1$.
   (b) If $i \in Tab1$, the algorithm $B$ responds with $H(name \| i) = h_i$ in $Tab1$.

   *(For TagGen oracle)* At any time the forger $A_1$ can request a tag for $(i, m_i)$ to the TagGen oracle. When the forger $A_1$ request $(i, m_i)$ to the TagGen oracle, the algorithm $B$ responds as follows.
   (a) If $i \notin Tab1$, the algorithm $B$ chooses a random number $\delta_i \in \mathbb{Z}_q^*$ and computes $\sigma_i = (g^a)^{\delta_i} \cdot (g^a)^{f_{\overline{\beta}_i}(\alpha)}$. The algorithm $B$ responds with $\sigma_i$ and stores the tuple $Tab2 = (i, \delta_i, *, h_i, m_i, \sigma_i)$.
   (b) If $i \in Tab1$, the algorithm $B$ retrieves the tuple $(i, \delta_i, c_i, h_i)$. If $c_i = 0$, then the algorithm $B$ reports a failure and terminates. If $c_i = 1$, the algorithm $B$ computes $\sigma_i = (g^a)^{\delta_i} \cdot (g^a)^{f_{\overline{\beta}_i}(\alpha)}$. The algorithm $B$ responds with $\sigma_i$ and updates the tuple $Tab2 = (i, \delta_i, c_i, h_i, m_i, \sigma_i)$.

3. Finally, the forger $A_1$ outputs the forged pair of message and tag $(i, m_i^*, \sigma_i^*)$ such that no TagGen query is issued for $m_i^*$. $B$ checks whether the pair $(m_i^*, \sigma_i^*)$ is valid under the given public keys $PK$. If $(m_i^*, \sigma_i^*)$ is not valid, the algorithm $B$ aborts. If $i \in Tab1$ and $c_i = 0$, $\sigma_i^*$ satisfies the following equation

$$\sigma_i^* = ((g^z)^t \cdot g^{b+\delta_i} \cdot g^{f_{\vec{\beta}_i}(\alpha)})^a = (g^a)^{zt} \cdot (g^a)^{b+\delta_i} (g^a)^{f_{\vec{\beta}_i}(\alpha)} = g^{ab} \cdot (g^a)^{zt+\delta_i} \cdot (g^a)^{f_{\vec{\beta}_i}(\alpha)} . \quad (16)$$

Then the algorithm $B$ outputs $g^{ab} = \sigma_i^* / ((g^a)^{zt+\delta_i} \cdot (g^a)^{f_{\vec{\beta}_i}(\alpha)})$.

First, we define three events to analyze $B$'s success probability as follows.

- $E_1$: During the TagGen queries made by the forger $A_1$, the algorithm $B$ does not abort.
- $E_2$: The forger $A_1$ outputs a valid pair of message and tag $(i, m_i^*, \sigma_i^*)$, which has not queried before.
- $E_3$: Event $E_2$ occurs and $c_i = 0$ for the tuple containing $i$ in $Tab1$.

If all events occur, the algorithm $B$ can solve the CDH problem. Thus, the probability of success is $Pr[E_1 \wedge E_3]$, and we can compute it as

$$Pr[E_1 \wedge E_3] = Pr[E_1] \cdot Pr[E_2 \mid E_1] \cdot Pr[E_3 \mid E_1 \wedge E_2]. \quad (17)$$

We analyze the lower bounds of $Pr[E_1]$, $Pr[E_2 \mid E_1]$, and $Pr[E_3 \mid E_1 \wedge E_2]$ in the following claims.

**Claim 1.** The probability $Pr[E_1]$ that the algorithm $B$ does not abort during the TagGen queries made by the forger $A_1$ is at least $1/e$.

*Proof of Claim 1.* We assume that the forger $A_1$ does not issue a query for the same index twice. In the TagGen oracle, the algorithm $B$ only aborts if $c_i = 0$, where $c_i$ is the random value with the probability $Pr[c_i = 0] = 1/(q_t + 1)$ corresponding to the TagGen query $i$ in $Tab1$. Based on the probability of $c_i$, the probability that the algorithm $B$ aborts is $1/(q_t + 1)$. Therefore, the probability $Pr[E_1]$ that the algorithm $B$ does not abort is $1 - 1/(q_t + 1)$ for each TagGen query. The TagGen queries are issued $q_t$ times at most, so the probability $Pr[E_1]$ that the algorithm $B$ does not abort during the forger $A_1$'s TagGen queries is at least $(1 - 1/(q_t + 1))^{q_t} \geq 1/e$.

**Claim 2.** The probability $Pr[E_2 \mid E_1]$ that the forger $A_1$ outputs a valid pair of message and tag under the condition that the event $E_1$ has occurred is at least $\varepsilon$.

*Proof of Claim 2.* When the event $E_1$ occurs, the probability $Pr[E_2 \mid E_1]$ depends on the information gathered by the forger $A_1$. The views of $A_1$ in the simulation and the real game are identical as follows.

- The distributions of $PK$ given to the forger $A_1$ in the simulation and the real game are identical.
- The distributions of the hash values of the hash function in the simulation and the real game are identical.
- The distributions of the responses to the TagGen oracle in the simulation and the real game are identical under the conditional probability.

Thus, the forger $A_1$ will generate a valid pair of message and tag with a probability at least $\varepsilon$.

**Claim 3.** The probability $Pr[E_3 \mid E_1 \wedge E_2]$ that the algorithm $B$ does not abort after the forger $A_1$ outputs a valid pair of message and tag is at least $1/(q_t+1)$.

*Proof of Claim 3.* When the events $E_1$ and $E_2$ have occurred, the algorithm $B$ will abort only in the case that $c_i = 0$ for the tuple containing $i$ corresponding to a forged pair of message and tag $(i, m_i^*, \sigma_i^*)$ in $Tab1$. Since the value $c_i$ is randomly chosen with the probability $Pr[c_i = 0] = 1/(q_t+1)$, the probability $Pr[E_3 \mid E_1 \wedge E_2]$ is at least $1/(q_t+1)$.

According to the claims above, the success probability of $B$ is $\varepsilon/(e\cdot(q_t+1))$. Thus we can conclude that if the CDH assumption holds in $G$, no algorithm exists that breaks the tag-unforgeability of our scheme with a non-negligible probability. □

**Theorem 4.2.** Our public auditing scheme is secure against proof-forgery(by Type-II adversary) under chosen message attacks in the random oracle model, if the DCDH assumption holds in $G$.

*Proof of Theorem 4.2.* Suppose that Type-II adversary $A_2$ is the proof forger that $(t, q_h, q_t, \varepsilon)$-breaks the public auditing scheme by generating a fake proof for an arbitrary message without tag-forgery, because we have already proven the tag-unforgeability in Theorem 4.1. Then we can construct an algorithm $B$ that solves the DCDH problem on $G$.

Given $(g, g^a, g^b)$ as an input for the DCDH problem, the algorithm $B$ simulates the challenger and interacts with the forger $A_2$ in the proof-unforgeability game.

1. The algorithm $B$ randomly selects $\alpha, z \leftarrow \mathbb{Z}_q^*$ and sets $\kappa = g^x = g^a$, $\{g^{\alpha^j}\}_{0\le j\le s+1}$, $v = g^{\alpha x} = (g^a)^\alpha$, $w = g^{at} = g^b$, and $u = g^z$. The algorithm $B$ sends $PK$ to the forger $A_2$.

2. The algorithm $B$ simulates the hash oracle and the $TagGen$ oracle as follows.
   *(For hash oracle $H$ )* At any time the forger $A_2$ is able to query the random hash oracle $h$ for $name \| i$. After receiving the hash queries, the algorithm $B$ responds as follows.
   (a) If $i \notin Tab1$, the algorithm $B$ chooses a random number $\delta_i \in \mathbb{Z}_q^*$ and sets $h_i = g^{\delta_i}$. Then the algorithm $B$ stores the tuple $Tab1 = (i, \delta_i, h_i)$ and responds with $H(name \| i) = h_i$ to the forger $A_2$.
   (b) If $i \in Tab1$, then the algorithm $B$ responds with $H(name \| i) = h_i$ in $Tab1$.

   *(For TagGen oracle)* At any time the forger $A_2$ can request a tag for $(i, m_i)$ to the TagGen oracle. When the forger $A_2$ asks $(i, m_i)$ to the TagGen oracle, the algorithm $B$ responds as follows.

(a) If $i \notin Tab1$, the algorithm $B$ chooses a random number $\delta_i \in \mathbb{Z}_q^*$ and computes $\sigma_i = (g^b)^z \cdot (g^a)^{\delta_i} \cdot (g^a)^{f_{\overline{\beta_i}}(\alpha)}$. The algorithm $B$ responds with $\sigma_i$ and stores the tuple $Tab2 = (i, \delta_i, h_i, m_i, \sigma_i)$.

(b) If $i \in Tab1$, the algorithm $B$ retrieves the tuple $(i, \delta_i, h_i)$ and computes $\sigma_i = (g^b)^z \cdot (g^a)^{\delta_i} \cdot (g^a)^{f_{\overline{\beta_i}}(\alpha)}$. The algorithm $B$ responds with $\sigma_i$ and stores the tuple $Tab2 = (i, \delta_i, h_i, m_i, \sigma_i)$.

3. The algorithm $B$ chooses $r, R \leftarrow \mathbb{Z}_q^*$ and a random $k$-elements subset $K \subset [1, n]$, and computes $g^R$. In this process, $B$ should choose a random subset that contains an index $i$ at least, where $i$ has not been queried before in the TagGen oracle. Let $K_d$ be a subset of the set $K$, which comprises indices that have not been queried in TagGen oracle. Then the algorithm $B$ sends $(K, r, g^R)$ as the challenge message.

4. Finally, if the forger $A_2$ outputs $(\pi, \psi, y, \rho^*)$ as the forged proof, the algorithm $B$ first checks the following equation

$$e(\eta, w) \cdot e(\rho^*, \kappa^R) \cdot e(\psi^R, v \cdot \kappa^{-r}) = \pi \cdot e(\kappa^{-y}, g^R). \tag{18}$$

If the above equation holds, $\rho^* = u^{-t\sum_{i \in K_d} p_i} \cdot \prod_{i \in K} h_i^{p_i}$, since the forger $A_2$ can not generate a valid tag. Then the algorithm $B$ computes $g^{a^{-1}b}$ as follows:

$$g^{a^{-1}b} = \left(\frac{\rho^*}{g^{\sum_{i \in K} \delta_i p_i}}\right)^{\frac{-1}{z \cdot \sum_{i \in K_d} p_i}}. \tag{19}$$

The success probability of $B$ is identical to the probability that the forger makes a valid proof forgery $(\pi, \psi, y, \rho^*)$. We analyze the lower bounds of the success probability in the following claim.

**Claim 4.** The success probability of $B$ is at least $\varepsilon$.

*Proof of Claim 4.* During the simulation, there is no case that the simulation is aborted. Therefore, the success probability of $B$ depends on the probability that the forger makes a valid proof forgery. Thus the success probability of $B$ is at least $\varepsilon$.

With the above claim, the algorithm $B$ solves the DCDH problem with the probability $\varepsilon$. Therefore, if the DCDH assumption holds, no algorithm exists that breaks the proof-unforgeability of our scheme with a non-negligible probability.

## 4.3 Extended Scheme with Deduplication

In this section, we describe the deduplication protocol for our public auditing scheme. When a user wishes to store a duplicated file, the server first checks the ownership of the file. If the user actually owns the file, the server gives access rights for the file to the user and deletes the duplicated files from the server. After the deduplication process, the users with valid rights to access the file is also able to perform the auditing process. The details of the deduplication process as follows.

Suppose that a user wants to store a file $F'$ in the server and $F'$ is stored in the server. To check that the user actually owns file $F'$, the server selects a $k$-elements subset $K$ of $[1,n]$ and requests the corresponding file blocks. After receiving the requests $K$, the user responds with the corresponding file blocks $m_i$, for all $i \in K$. Then the server computes

$$\sigma' = \prod_{i\in K}\sigma_i, \rho' = \prod_{i\in K} h_i, \ \ \psi' = e(\prod_{j=2}^{s+1}(g^{\alpha^j})^{\beta_j}, \kappa) = e(g^{f_{\vec{B}}(\alpha)}, \kappa), \tag{20}$$

where $\vec{B} = (0, 0, \sum_{i\in K} m_{i0}, \ldots, \sum_{i\in K} m_{i(s-1)})$, and checks the following equation

$$e(u,w)\cdot e(\rho',\kappa)\cdot\psi' = e(\sigma', g). \tag{21}$$

If the equation holds, the server considers that the user owns the file $F'$ and gives the user access rights to file $F'$, as follows. At first, let $user_0$ be the original user who uploaded file $F'$, $user_\ell$ be a new user who passes the ownership check defined above, and $S_{owner}$ be a set of users with access rights to the same file $F'$, where $1 \le \ell \le S_\ell$ and $S_\ell$ is the number of elements in the set $S_{owner}$. To eliminate the duplicated files, the server computes an aggregated tag $\sigma_{i'}$ and stores only the aggregated tag $\sigma_i'$ instead of storing all the tags generated by the different users for the same file $F'$, where

$$\sigma_{i'} = \prod_{\ell=0}^{S_\ell}\sigma_{\ell i} = u^{\sum_{\ell=0}^{S_\ell} t_\ell x_\ell}\cdot(h_i\cdot g^{f_{\vec{\beta_i}}(\alpha)})^{\sum_{\ell=0}^{S_\ell} x_\ell}. \tag{22}$$

When $user_c$ who belongs to the set $S_{owner}$ wants to audit the integrity of $F'$, he runs Challenge algorithm and then sends $(K, r, g^R)$ as a challenge message to the server. On receiving the challenge message, the server computes $\kappa' = \prod \kappa_\ell$, $w' = \prod w_\ell$, and $v' = \prod v_\ell$, where $\ell \in S_{owner} / c$, and runs the Prove algorithm to generate $(\pi, \psi, y, \rho)$. The server sends $(\pi, \psi, y, \rho, \kappa', v', w')$ to $user_c$. Then $user_c$ can verify the proof for the challenged message by the following equation:

$$e(\eta, w)\cdot e(\rho, \kappa^R)\cdot e(\psi^R, v'\cdot v_c\cdot\kappa^{-r}) = \pi\cdot e(\kappa^{-y}, g^R), \tag{23}$$

where $\kappa = \kappa'\cdot\kappa_c$ and $w = w'\cdot w_c$. If the equation holds, TPA outputs *Accept*; otherwise TPA outputs *Reject*. The correctness of the Equation (23) can be verified easily by extending the correctness of the Equation (15), so we omit the details.

**Table 2.** Security and Complexity Comparison: Let *Exp* and *Pair* be the exponentiation operation and the bilinear pairing operation, respectively. $s$ is the number of sectors for the file block, $d$ is the number of challenged blocks, and $L$ is the number of users' subgroup.

| | **Computation Cost of TPA** | **Communication Cost** | **Tag-Unforgeability** |
|---|---|---|---|
| [27] | $O(s)\cdot Exp + O(1)\cdot Pair$ | $O(s+d)$ | secure |
| [28] | $O(1)\cdot Exp + O(1)\cdot Pair$ | $O(1)$ | insecure |
| [29] | $O(d)\cdot Exp + O(1)\cdot Pair$ | $O(s+d)$ | secure |
| [30] | $O(d)\cdot Exp + O(L)\cdot Pair$ | $O(s+d)$ | secure |
| **Ours** | $O(1)\cdot Exp + O(1)\cdot Pair$ | $O(1)$ | secure |

### 4.4 Discussions

**Complexity Analysis.** We compare the computation and communication costs of our scheme with those of related works in **Table 2** [26,28,29,30]. As shown in **Table 2**, the scheme proposed in [28] has constant computation and communication costs, but the scheme is not secure against tag-forgery. By contrast, our scheme not only has constant computation and communication costs, but also is constructed secure against the tag-forgery.

**Batch-Auditing.** Occasionally, TPA may handle many auditing tasks during a short period of time for multiple users. If TPA processes the auditing tasks one by one, long period of time is needed for their computation. To reduce the communication and computation costs, we can extend our scheme by using a technique similar to that proposed in [28] to support batch auditing, thereby processing many auditing tasks at the same time.

**Error Detection Probability.** In our public auditing scheme, TPA requests a proof for a $k$ -elements subset of the stored data blocks instead of all the data blocks in order to improve the efficiency of our scheme. Note that even though TPA uses the $k$ sampled blocks for the auditing, TPA can still detect the corrupted blocks with a high probability [4]. Suppose that $1\%$ of all the blocks are corrupted. In such case, TPA can detect the modification of stored data blocks with a probability of $99\%$ or $95\%$ by choosing 460 or 360 challenge data blocks, respectively. If the user wishes to increase the detection probability for the same corrupted data blocks, TPA simply increases the number of the challenge data blocks, although this decreases the efficiency of the scheme.

## 5. Conclusion

In the paper, we pointed out that the Yuan and Yu’s public auditing scheme with deduplication is not secure against a tag-forgery by a malicious server, and we proposed an improved scheme which is secure and has the constant communication and computation costs. We also proved that our improved scheme satisfies the tag-unforgeability and the proof-unforgeability under the formal security models.

To the best of our knowledge, no ID-based public auditing scheme which has constant computation cost and is proved under formal security model has been proposed. Therefore, it may be interesting to construct a provably-secure ID-based public auditing scheme which has constant computation costs.

## References

[1] Dropbox for Business. [Online]. Available: https://www.dropbox.com/business, accessed Jan. 14, 2016.

[2] Google Drive. [Online]. Available: https://drive.google.com/, accessed Jan. 14, 2016.

[3] iCloud. [Online]. Available: https://www.icloud.com/, accessed Jan. 14, 2016.

[4] G. Ateniese, R. Burns, R. Curtmola, H. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable Data Possession at Untrusted Stores,” in *Proc. of the 14th ACM Conf. on Computer and Communications Security, CCS 2007*, pp.598-609, 2007. Article(CrossRef Link)

[5] A. Juels and B.S. Kaliski, “PORs: Proofs of retrievability for large files,” in *Proc. of the 14th ACM Conf. on Computer and Communications Security, CCS 2007*, pp.584-597, 2007. Article(CrossRef Link)

[6] G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik, “Scalable and efficient provable data possession,” in *Proc. of the 4th Int. Conf. on Security and privacy in Communication networks, SecureComm 2008*, pp.1-10, 2008. Article(CrossRef Link)

[7] H. Shacham and B. Waters, “Compact proofs of retrievability,” in *Proc. of the 14th annual Int. Conf. on the theory and application of cryptology* & *information security, ASIACRYPT 2008*, pp.90-107, 2008. Article(CrossRef Link)

[8] C. Wang, Q. Wang, K. Ren, and W. Lou, “Ensuring data storage security in cloud computing,” in *Proc. of the 17th IEEE Int. Workshop on Quality of Services, IWQoS 2009*, pp.1-9, 2009. Article(CrossRef Link)

[9] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, “Enabling Public Verifiability and Data Dynamic for Storage Security in Cloud Computing,” in *Proc. of the 14th European Symposium on Research in Computer Security, ESORICS 2009*, pp.355-370, 2009. Article(CrossRef Link)

[10] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, “Dynamic Provable Data Possession,” in *Proc. of the 16th ACM Conf. on Computer and Communications Security, CCS 2009*, pp.213-222, 2009. Article(CrossRef Link)

[11] C. Wang, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for data storage security in cloud computing,” in *Proc. of the 29th IEEE Conf. on Computer Communications, INFOCOM 2010*, pp.525-533, 2010. Article(CrossRef Link)

[12] C. Wang, Q. Wang, K. Ren, and L. Lou, “Towards Secure and Dependable Storage Services in Cloud Computing,” *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp.220-232, 2012. Article(CrossRef Link)

[13] B. Wang, B. Li, and H. Li, “Oruta: Privacy-preserving public auditing for shared data in the cloud,” in *Proc. of the 5th IEEE Int. Conf. on Cloud Computing, CLOUD 2012*, pp.295-302, 2012. Article(CrossRef Link)

[14] C. Wang, S.S.M. Chow, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for secure cloud storage,” *IEEE Transactions on Computers*, vol. 62, no.2, pp.362-375, 2013. Article(CrossRef Link)

[15] H. Wang, “Proxy provable data possession in public clouds,” *IEEE Transactions on Services Comput*ing, vol. 6, no. 4, pp.551-559, 2013. Article(CrossRef Link)

[16] B. Wang, B. Li, and H. Li, “Public auditing for shared data with efficient user revocation in the cloud,” in *Proc. of the 32th IEEE Int. Conf. on Computer Communications, INFOCOM 2013*, pp.2904-2912, 2013. Article(CrossRef Link)

[17] J. Yuan and S. Yu, “Proofs of Retrievability with Public Verifiability and Constant Communication Cost in Cloud,” in *Proc. of the 2013 Int. workshop on Security in cloud computing , AISACCS-SCC 2013*, pp.19-26, 2013. Article(CrossRef Link)

[18] S.G. Worku, C. Xu, J. Zhao, and X. He, “Secure and efficient privacy-preserving public auditing scheme for cloud storage,” *Computers & Electrical Engineering*, vol. 40, no. 5, pp.1703-1713, 2014. Article(CrossRef Link)

[19] Y. Yu, J. Ni, M.H. Au, Y. Mu, B. Wang, and H. Li, “On the Security of a Public Auditing Mechanism for Shared Cloud Data Service,” *IEEE Transactions on Services Computing*, vol. 8, no. 6, pp.998-999, 2014. Article(CrossRef Link)

[20] F. Armknecht, JM. Bohli, GO. Karame, Z. Liu, and CA. Reuter, “Outsourced Proofs of Retrievability,” in *Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security, CCS 2014*, pp.831-843, 2014. Article(CrossRef Link)

[21] T. Jiang, X. Chen, and J. Ma, “Public Integrity Auditing for Shared Dynamic Cloud Data with Group User Revocation,” *IEEE Transactions on Computers*, vol. PP, no. 99, pp.1-12, 2015. Article(CrossRef Link)

[22] A. F. Barsoum and M. A. Hasan, “Provable multicopy dynamic data possession in cloud computing systems,” *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 3, pp.485-497, 2015. Article(CrossRef Link)

[23] G. Yang, J. Yu, W. Shen, Q. Su, Z. Fu, and R. Hao, “Enabling public auditing for shared data in cloud storage supporting identity privacy and traceability,” *The Journal of Systems and Software*, vol. 113, pp.130-139, 2016. Article(CrossRef Link)

[24] J. Yu, K. Ren, and C. Wang, “Enabling Cloud Storage Auditing with Verifiable Outsourcing of Key Updates,” *IEEE Transactions on Information Forensics and Security,* vol. 11, no. 6, pp.1362-1375, 2016. Article(CrossRef Link)
[25] Y. Li, Y. Yu, B. Yang, G. Min, and H. Wu, “Privacy preserving cloud data auditing with efficient key update,” *Future Generation Computer Systems*, available online, 2016. Article(CrossRef Link)
[26] Q. Zheng, and S. Xu, “Secure and efficient proof of storage with deduplication,” in *Proc. of the 2nd ACM Conf. on Data and Application Security and Privacy, CODASPY 2012*, pp.1-12, 2012. Article(CrossRef Link)
[27] Y. Shin, D. Koo, J. Hur, and J. Yun, “Secure proof of storage with deduplication for cloud storage systems,” *Multimedia Tools and Application*, pp.1-16, 2015. Article(CrossRef Link)
[28] J. Yuan and S. Yu, “Secure and Constant Cost Public Cloud Storage Auditing with Deduplication,” in *Proc. of the IEEE Conf. on Communications and Network Security, CNS 2013*, pp.145-153, 2013. Article(CrossRef Link)
[29] J. Li, J. Li, D. Xie, and Z. Cai, “Secure Auditing and Deduplicating Data in Cloud,” *IEEE Transactions on Computers*, vol. PP, no. 99, pp.1-11, 2015. Article(CrossRef Link)
[30] N. Alkhojandi and A. Miri, “Privacy-Preserving Public Auditing in Cloud Computing with Data Deduplication,” in *Proc. of the 8th Int. Symp. on Foundations & Practice of Security, FPS 2015,* pp.35-48, 2015. Article(CrossRef Link)
[31] K. He, J. Chen, R. Du, Q. Wu, G. Xue, and X. Zhang, “DeyPoS: Duplicatable Dynamic Proof of Storage for Multi-User Environments,” *IEEE Transactions on Computers*, Vol. 65, no. 12, 2016. Article(CrossRef Link)
[32] F. Bao, R.H. Deng, and H. Zhu, “Variations of Diffie-Hellman Problem,” in *Proc. of the 5th Int. Conf. of Information and Communications Security*, ICICS 2003, pp.301-312, 2003. Article(CrossRef Link)
[33] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the Weil pairing,” in *Proc. of the 7th annual Int. Conf. on the theory and application of cryptology & information security, ASIACRYPT 2001*, pp.514-532, 2001. Article(CrossRef Link)

**Dongmin Kim** received his B.S. degree in mathematics from University of Seoul, Korea, in 2009. He received his M.S. degree in Information Security from Korea University in 2011. Currently, he is a Ph.D. candidate in the Graduate School of Information Security, Korea University, Seoul, Korea. His current research areas include cryptographic protocols, and privacy-preserving technologies.

**Ik Rae Jeong** received his B.S. and M.S. degrees in Computer Science from Korea University, Korea, in 1998 and 2000, respectively. He received his Ph.D. degree in Information Security from Korea University in 2004. From June 2006 to February 2008, he was a senior engineer at the Electronics and Telecommunications Research Institute(ETRI) in Korea. Currently, he is a member of the faculty in the Graduate School of Information Security, Korea University, Seoul, Korea. His current research areas include cryptography and theoretical computer science.