Certificateless Public Auditing Protocol with Constant Verification Time

Dongmin Kim and Ik Rae Jeong

Center for Information Security Technologies (CIST), Korea University, Anam-dong, Seongbuk-gu, Seoul 136-713, Republic of Korea

Correspondence should be addressed to Ik Rae Jeong; irjeong@korea.ac.kr

Received 18 August 2016; Accepted 7 November 2016; Published 30 January 2017

Academic Editor: Muhammad Khurram Khan

Copyright © 2017 Dongmin Kim and Ik Rae Jeong. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

To provide the integrity of outsourced data in the cloud storage services, many public auditing schemes which allow a user to check the integrity of the outsourced data have been proposed. Since most of the schemes are constructed on Public Key Infrastructure (PKI), they suffer from several concerns like management of certificates. To resolve the problems, certificateless public auditing schemes also have been studied in recent years. In this paper, we propose a certificateless public auditing scheme which has the constant-time verification algorithm. Therefore, our scheme is more efficient than previous certificateless public auditing schemes. To prove the security of our certificateless public auditing scheme, we first define three formal security models and prove the security of our scheme under the three security models.

1. Introduction

Cloud storage is an essential service of cloud computing, which allows users to outsource their data to a cloud server. It is an easy and reliable way to handle a large amount of data due to the characteristics of cloud computing such as ubiquitous network access, location-independent resource pooling, and elasticity [1]. However, there exist challenging security threats with respect to the outsourced data. An untrusted server might discard the data to save a disk space or modify the data for personal gains. Therefore, the users need to be convinced that their data are stored in the cloud server without any loss or modification. That is, the cloud server should ensure the data integrity to the users.

To check the integrity of the outsourced files efficiently, many solutions have been proposed in recent years. When the user wishes to check the integrity, the third-party auditor (TPA) executes auditing algorithms in place of the user without downloading the whole files. If the TPA performs auditing process without any secret values delegated from the user, we called it public auditing. Most of the public auditing schemes are based on Public Key Infrastructure (PKI) [2–10]. In traditional PKI, Certificate Authority (CA) issues the private/public keys for the users and certificates to bind the users with their public keys. These certificates are generated and managed by the CA. Even though PKI has been widely used in public key cryptography (PKC), it has some security issues related to the management of the certificates, such as revocation, distribution, storage, and the computational cost of the certificates verification as described in [11, 12].

Identity-based public key cryptography was introduced to overcome the above issues of PKI by Shamir in [13]. In the identity-based public key cryptography, the users use identity as their public key (e.g., e-mail address) which is assumed to be publicly known, so there is no need to consider the management of the certificates [13–17]. A trusted third party called Key Generation Center (KGC) generates a master secret key and publishes a master public key corresponding to the master secret key. Then the KGC issues users’ private key corresponding to the identity (public keys) using the master secret key. Unfortunately, this approach causes a new inherent problem known as key escrow. Since the users’ private key is entirely generated by the KGC and it is not fully trusted, it can impersonate any valid user whenever it wants. The condition that the KGC is fully trusted has to be assumed to resolve the above inherent problem. The problem also should
be considered for the identity-based public auditing schemes [18–20].

Al-Riyami and Paterson proposed certificateless cryptosystem that preserves the advantages of the identity-based public key cryptography and eliminates the vulnerability of it [21]. In other words, certificateless public key cryptography does not require certificates to authenticate the users’ public key and the strong assumption. In the certificateless public cryptography, the KGC cannot know the user’s full secret key unlike the ID-PKC, because the user’s full secret key is generated by combining a partial private key generated by the KGC with a secret value chosen by the user [21–23]. As the related works, many certificateless signature schemes are also proposed [24–27].

In [28], the authors proposed the secure certificateless public auditing scheme based on a new certificateless signature scheme constructed by them. However, He et al. pointed out that the certificateless signature scheme cannot withstand the public key replacement attack, where the adversary can replace user’s public key with a manipulated public key [29]. They also proposed the certificateless public auditing scheme suitable for a cloud-assisted Wireless Body Area Networks (WBAN). Even though many authentication schemes are proposed, they are not suitable because they suffer from the certificate management and key escrow problem [30–35]. The certificateless public auditing scheme is constructed to withstand the attack and also ensures the integrity of stored data which is an important issue for the cloud-assisted WBAN, therefore the certificateless public auditing scheme is an essential technology to construct WBAN.

1.1. Our Contributions

(1) We first design a certificateless public auditing (CL-PA) scheme which has the constant computation cost in verification. The previous certificateless public auditing schemes require the linearly increasing verification time with respect to the number of the challenged message blocks.

(2) We define the three formal security models which ensure the security against the public key replacement attack, malicious-but-passive-KGC attack, and proof-forgery.

(3) We show that our certificateless scheme is secure against existential forgery by proving the security under the models.

The rest of paper is organized as follows. In Section 2, we describe the formal security models and we review some preliminaries in Section 3. We construct our scheme and prove the securities in Section 4. We conclude the paper in Section 5.

2. Preliminaries

2.1. Bilinear Group. Let groups $G$ and $G_T$ be multiplicative cyclic groups with prime order $q$. A function $e : G \times G \rightarrow G_T$ is a bilinear map, if it satisfies the following:

\begin{align}
(1) \quad & e(g^a, g^b) = e(g, g)^{ab} \quad \text{for all } g \in G, a, b \in \mathbb{Z}_q^*.
(2) \quad & e(g, g) = 1
(3) \quad & \text{It is easy to compute } e(g^a, g^b) \text{ for all } g \in G.
\end{align}

2.2. Complexity Assumptions

Computational Diffie-Hellman (CDH) Assumption. Given a tuple $g, g^a, g^b \in G$, an algorithm $\mathcal{A}$ tries to compute $g^{ab} \in G$. We assume that there is no algorithm $\mathcal{A}$ with a nonnegligible probability $\epsilon$ such that

$$
\Pr[\mathcal{A}(g, g^a, g^b) = g^{ab}] \geq \epsilon,
$$

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

Divisible Computational Diffie-Hellman (DCDH) Assumption [36]. Given a tuple $g, g^a, g^b \in G$, an algorithm $\mathcal{A}$ tries to compute $g^{a^{-1}b} \in G$. We assume that there is no algorithm $\mathcal{A}$ with a nonnegligible probability $\epsilon$ such that

$$
\Pr[\mathcal{A}(g, g^a, g^b) = g^{a^{-1}b}] \geq \epsilon,
$$

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

3. Models

3.1. System Model. We consider a system model that has four entities, KGC (Key Generation Center), a user, a cloud server, and a third-party auditor (TPA) as shown Figure 1. The KGC is responsible for the generation of a master key and a set of public keys for the system and issuing the partial private key for an identity (ID). Unlike the PKG in identity-based system, the KGC cannot issue the user’s full secret key that will be used in the system. The full secret key is generated by combining the partial private key with some
random secret values chosen from the user. With this key, the user generates authenticated tags for the file blocks and uploads them to the cloud server. When the user wishes to check the integrity of the stored files, TPA performs it on behalf of the user. For checking the integrity of the stored files, TPA sends a challenge message to the cloud server, and then the server responds with the proof corresponding to the challenge message. TPA is able to check the integrity of them by verifying the validity of the received proof.

We define the algorithms for our certificateless public auditing scheme as follows:

(i) **Setup** takes a security parameter and returns a master key (MK) and a public parameter (PP).

(ii) **Partial-Private-Key-Extract** takes an identity (ID), the master key (MK), and the public parameter (PP). It outputs a partial private key (ppk) for the identity (ID).

(iii) **KeyGen** takes an identity (ID), the public parameter (PP), and the partial private key (psk). It outputs a set of public keys (PK) and a secret key (sk) of the identity (ID).

(iv) **TagGen** takes as an input the public parameter (PP), a secret key (sk) of a user, and a file $F = (m_1, \ldots, m_n)$, where $n$ is the number of the file blocks. It outputs tags $\{t_i\}_{1 \leq i \leq n}$ for the file blocks.

(v) **Challenge** takes as an input the public parameters pp and returns a challenge message $C$.

(vi) **Prove** takes as an input the public parameter (PP) and the challenge message $C$ and the pairs of message and tag $\{m_i, t_i\}_{1 \leq i \leq n}$. It outputs a proof (Prf) for the challenge message $C$.

(vii) **Verify** takes as an input the public parameter (PP), the set of public keys (PK), the challenge message $C$, and the proof (Prf). It outputs *Accept* or *Reject*.

### 3.2. Security Models

We assume that three types of adversaries in our certificateless public auditing scheme, that is, Type I adversary $A_1$, Type II adversary $A_2$, and Type III adversary $A_3$, and define the adversaries as follows:

(i) **Type I adversary $A_1$** cannot access the master key, but he can replace user’s public key with a manipulated public key and generate a forged pair of message and tag.

(ii) **Type II adversary $A_2$**, which represents a malicious KGC, can access the master key and generate a forged pair of message and tag, but he cannot replace user’s public key.

(iii) **Type III adversary $A_3$**, which represents a malicious server, can forge a proof for the challenge message without knowing any secret values.

Type I adversary and Type II adversary aim to forge the pair of message and tag and are classed as the normal, strong, and super adversaries (ordered by the capability of the adversaries). If the scheme is secure against the super adversary, it is also secure against the normal and strong adversaries. Therefore, we just consider the super Type I adversary and the super Type II adversary in Game I and Game II, respectively. We define three security models using the following games between a challenger and the defined adversaries.

**Game I.** Type I adversary $A_1$ cannot forge a pair of message and tag, and then we consider that a certificateless public auditing scheme is secure against a tag-forgery by the public key replacement attack. It is defined using the following game between a challenger $C$ and Type I adversary $A_1$:

1. Type I adversary outputs an identity $ID_{tg}$ where it wishes to be forged.
2. The challenger $C$ runs algorithm Setup to generate a public parameter (PP) and the master key (MK) and sends PP to Type I adversary $A_1$.
3. Type I adversary $A_1$ is allowed to access the hash oracles, KeyGen, Partial-Private-Key-Extract, Public-Key-Replacement, Secret-Value-Extract, and TagGen oracle, adaptively. For the KeyGen oracle, Type I adversary $A_1$ sends $ID_o$ to the challenger $C$, and then the public key $pk_r$ is returned. For the Partial-Private-Key-Extract oracle, Type I adversary $A_1$ sends $ID_o$ to the challenger $C$, and then the partial private key $psk_r$ is returned. For the Public-Key-Replacement oracle, Type I adversary $A_1$ sends $\{ID_o, pk_r\}$ to the challenger $C$. For the Secret-Value-Extract oracle, Type I adversary $A_1$ sends $ID_o$ to the challenger $C$, and then the secret value $sv_{ID_o}$ is returned. For the TagGen oracle, Type I adversary $A_1$ sends $\{ID_a, id, m_i\}$, where $id$ is the block identifier, to the challenger $C$. The challenger $C$ responses with a pair of message and tag $(m_i, t_i)$, where $t_i = TagGen(sk_r, ID_{oa}, id, m_i)$.
4. Finally, Type I adversary $A_1$ outputs a forged pair of message and tag $(id_a, m^*, t^*)$ for an identity $ID_{tg}$.

If the pair $(id_a, m^*, t^*)$ is valid pair of message and tag and $m^*$ has not queried TagGen oracle, then Type I adversary $A_1$ wins the game.

**Definition 1.** Type I adversary $A_1(t, q_{Hp}, q_{qpk}, q_{ppk}, q_{pv}, q_\theta, \epsilon)$-breaks a certificateless public auditing scheme, if $A_1$ runs in time at most $t$, makes at most $q_{Hp}$ hash queries, $q_{ppk}$ KeyGen queries, $q_{qpk}$ Partial-Private-Key-Extract queries, $q_{pv}$ Public-Key-Replacement queries, $q_{sv}$ Secret-Value-Extract queries, and $q_\theta$ TagGen queries, and wins the game with the probability $\epsilon$. A certificateless public auditing scheme is secure against a tag-forgery by the public key replacement attack, if there is no tag-forgery which $(t, q_{Hp}, q_{qpk}, q_{ppk}, q_{pv}, q_\theta, \epsilon)$-breaks the scheme with a nonnegligible probability $\epsilon$.

**Game II.** If Type II adversary $A_2$ cannot forge a pair of message and tag, then we consider that a certificateless public auditing scheme is secure against a tag-forgery by the malicious-but-passive-KGC attack. It is defined using the
following game between a challenger $C$ and Type II adversary $\mathcal{A}_2$:

1. Type II adversary outputs an identity $ID_u$ where it wishes to be forged.
2. The challenger $C$ runs algorithm Setup to generate a public parameter (PP) and the master key (MK) and sends "PP and MK" to Type II adversary $\mathcal{A}_2$.
3. Type II adversary $\mathcal{A}_2$ is allowed to access the hash oracles, KeyGen, Partial-Private-Key-Extract, Secret-Value-Extract, and TagGen oracle, adaptively. For KeyGen oracle, Type II adversary $\mathcal{A}_2$ sends $ID_u$ to the challenger $C$, and then the public key $pk_u$ is returned. For Partial-Private-Key-Extract oracle, Type II adversary $\mathcal{A}_2$ sends $ID_u$ to the challenger $C$, and then the partial private key $psk_u$ is returned. For Secret-Value-Extract oracle, Type III adversary $\mathcal{A}_3$ sends $ID_u$ to the challenger $C$, and then the secret value $sv_{ID_u}$ is returned. For TagGen oracle, Type III adversary $\mathcal{A}_3$ sends $(ID_u, id, m)$, where $id$ is the block identifier, to the challenger $C$. The challenger $C$ responds with a pair of message and tag $(m, t)$, where $t = TagGen(sk_u, ID_u, id, m)$.
4. Finally, Type II adversary $\mathcal{A}_2$ outputs a forged pair of message and tag $(id, m^*, t^*)$ for an identity $ID_y$.

If the pair $(id, m^*, t^*)$ is valid pair of message and tag and $m^*$ has not queried TagGen oracle, then Type II adversary $\mathcal{A}_2$ wins the game.

Definition 2. Type II adversary $\mathcal{A}_2$: $(t, q_h, q_k, q_{ppk}, q_{sv}, q_{f}, \epsilon)$-breaks a certificateless public auditing scheme, if $\mathcal{A}_2$ runs in time at most $t$, makes at most $q_h$ hash queries, $q_k$ KeyGen queries, $q_{ppk}$ Partial-Private-Key-Extract queries, $q_{sv}$ Secret-Value-Extract queries, and $q_{f}$ TagGen queries, and wins the game with the probability $\epsilon$. A certificateless public auditing scheme is secure against a tag-forgery by the malicious-but-passive-KGC attack, if there is no tag-forgery which $(t, q_h, q_k, q_{ppk}, q_{sv}, q_{f}, \epsilon)$-breaks the scheme with a nonnegligible probability $\epsilon$.

Game III. Type III adversary $\mathcal{A}_3$ cannot generate a valid proof for the blocks modified or deleted by Type III adversary $\mathcal{A}_3$, and then we say that a public auditing scheme satisfies a proof-unforgeability (soundness). It is defined using the following game between a challenger $C$ and Type III adversary $\mathcal{A}_3$:

1. The challenger $C$ runs algorithm Setup to generate a public parameter (PP) and the master key (MK) and sends PP to Type III adversary $\mathcal{A}_3$.
2. Type III adversary $\mathcal{A}_3$ is allowed to access the hash oracles, KeyGen, Partial-Private-Key-Extract, Secret-Value-Extract, and TagGen oracle, adaptively. For KeyGen oracle, Type III adversary $\mathcal{A}_3$ sends $ID_u$ to the challenger $C$, and then the public key $pk_u$ is returned. For Partial-Private-Key-Extract oracle, Type III adversary $\mathcal{A}_3$ sends $ID_u$ to the challenger $C$, and then the partial private key $psk_u$ is returned. For Secret-Value-Extract oracle, Type III adversary $\mathcal{A}_3$ sends $(ID_u, id, m)$, where $id$ is the block identifier, to the challenger $C$. The challenger $C$ responds with a pair of message and tag $(m, t)$, where $t = TagGen(sk_u, ID_u, id, m)$.
3. The challenger $C$ generates the challenge message $C = \text{CHALLENGE}(PP)$ and sends it to Type III adversary $\mathcal{A}_3$.
4. Finally, Type III adversary $\mathcal{A}_3$ outputs a forged proof $Prf^*$.

If the equation $\text{VERIFY}(Prf^*) = \text{Accept}$ holds and the challenge message $C$ includes the index which is not queried in TagGen queries, then Type III adversary $\mathcal{A}_3$ wins the game.

Definition 3. Type III adversary $\mathcal{A}_3$: $(t, q_h, q_k, q_{ppk}, q_{sv}, q_{f}, \epsilon)$-breaks a certificateless public auditing scheme, if $\mathcal{A}_3$ runs in time at most $t$, makes at most $q_h$ hash queries, $q_k$ KeyGen queries, $q_{ppk}$ Partial-Private-Key-Extract queries, $q_{sv}$ Secret-Value-Extract queries, and $q_{f}$ TagGen queries, and wins the game with the probability $\epsilon$. A certificateless public auditing scheme is proof-unforgeable (sound), if there is no proof-forger which $(t, q_h, q_k, q_{ppk}, q_{sv}, q_{f}, \epsilon)$-breaks the scheme with a nonnegligible probability $\epsilon$.

4. Our Certificateless Public Auditing (CL-PA) Scheme

In this section, we propose a provably secure certificateless public auditing scheme. Our scheme consists of the seven algorithms, Setup, Partial-Private-Key-Extract, KeyGen, TagGen, Challenge, Prove, and Verify. The details of the algorithms are as follows:

SetUp: let $G$ and $G_T$ be multiplicative cyclic groups of prime order $q$ and $e : G \times G \rightarrow G_T$ be a bilinear map. Let $g, g_1$, and $g_2$ be generators of $G$. $H() : \{0, 1\}^* \rightarrow Z_q^*$, $\hat{H}() : \{0, 1\}^* \rightarrow Z_q^*$, and $h() : \{0, 1\}^* \rightarrow G$ are collision resistant hash functions. KGC randomly chooses $\alpha, x \leftarrow Z_q^*$ and computes $\{g_1^\alpha\}_{0 \leq j \leq s+1}$ and $X_1 = g^x$ as system parameters. A set of master keys is $MK = (\alpha, x)$ and the public parameters are $PP = (G, G_T, e, H, h, g, g_1, g_2, X_1, g^x, \{g_1^\alpha\}_{0 \leq j \leq s+1})$.

Partial-Private-Key-Extract: for each identity $ID_u \in \{0, 1\}^*$, KGC picks the random number $r \leftarrow Z_q^*$ and computes $R_1 = g^r, \tilde{\sigma} = r + xH(ID_u, R_1)$ mod $q$, and $\overline{R_2} = g^{\alpha \tilde{\sigma}}$. KGC sends $pk_u = (R_1, R_2, \overline{R_2})$ to the user $U$ as the partial private keys for the identity $ID_u$.

KeyGen: given the public parameters and the partial private keys, the user $U$ generates his/her secret keys and public keys. The user $U$ chooses $w, k \leftarrow Z_q^*$ and computes $\sigma = k + \tilde{\sigma} = k + r + xH(ID_u, R_1)$, $R_0 = g^k, R_2 = \overline{R_2}, g^{(g_1^k)k}, R_3 = (R_1 - R_2)^w, X_2 = X_1^w$. The user sets $sk_u = (\sigma, w)$ as a set of secret keys and $pk_u = (R_0, R_1, R_2, R_3, X_2)$ as a set of public keys.
The correctness of our scheme can be proved as follows:

\[ e(g_1^n, g^{\omega_T}) \cdot e(g_2^y, g^\sigma) \cdot e(\psi, R_2 \cdot g^{\sigma(-r_1)}) = e\left(\prod_{i \in D} g_1^{\tilde{h}_i \cdot i}, \prod_{i \in D} h_i \cdot g_2^{f_h(\pi)}\right) \cdot e(g_2^y, g^\sigma) \cdot e(\psi, g^{\omega_T} \cdot g^{\sigma(-r_1)}) \]

\[ = e\left(\prod_{i \in D} g_1^{\tilde{h}_i \cdot i}, \prod_{i \in D} h_i \cdot g_2^{f_h(\pi)}\right) \cdot e(g_2^y, g^\sigma) \cdot e(g_2^f(\tau_1), g^\sigma) \]

\[ = e\left(\prod_{i \in D} g_1^{\tilde{h}_i \cdot i}, \prod_{i \in D} h_i \cdot g_2^{f_h(\pi)}\right) \cdot e(g_2^f(\tau_1), g^\sigma) \]

\[ = e\left(\prod_{i \in D} g_1^{\tilde{h}_i \cdot i}, \prod_{i \in D} h_i \cdot g_2^{f_h(\pi)}\right) \cdot e(g_2^f(\tau_1), g^\sigma) \]

4.1. Security Analyses. In this section, we analyze the security of our certificateless public auditing scheme. If the scheme is secure against the tag-forgery by Type I adversary and Type II adversary and the proof-forgery by Type III adversary under adaptive chosen message attacks, then we consider that the scheme is secure against existential forgery.

Theorem 4. The proposed certificateless public auditing scheme is secure against the tag-forgery by Type I adversary under chosen message attacks in the random oracle model, if the CDH assumption holds in \( G \).

Proof of Theorem 4. Suppose Type I adversary \( \mathcal{A} \) is the forger that could break the scheme by generating a valid pair of message and tag under user's secret key with a probability \( \epsilon \). Then we can construct an algorithm \( \mathcal{B} \) that solves the CDH problem on \( G \).
Given an instance \((g, g^a, g^b) \in G\) of the CDH problem, the algorithm \(B\) simulates the challenger and interacts with the forger \(A_1\) in Game I.

1. The forger \(A_1\) outputs an identity ID\(_{tg}\) where it wishes to be forged.

2. The algorithm \(B\) chooses random numbers \(\alpha, z_1, z_2 \leftarrow Z_p^*\) and computes \(g^{\alpha z_1} = g^{z_2}\). The algorithm \(B\) sets \(X_1 = g^\alpha\) and \(g_1 = g^\beta\) and sends PP to the forger \(A_1\).

3. For KeyGen oracle, at any time the forger \(A_1\) can query the KeyGen oracle for an identity ID\(_u\). The algorithm \(B\) stores Tab1 = \((ID_u, e_u, r_u, \sigma_u, u_u, w_u, k_u, R_{u,2}, pk_u)\), where Tab1 is initially empty. When the forger \(A_1\) queries ID\(_u\), the algorithm \(B\) responds as follows:

   (a) If ID\(_u\) \neq ID\(_{tg}\), the algorithm \(B\) chooses random number \(e_u, \sigma_u, u_u, k_u \leftarrow Z_p^*\) and calculates \(\sigma_u = \sigma_u + k_u, u_u = g^{\alpha u} / (g^{\beta u})^{\alpha u}, R_{u,0} = g^{\alpha u}, R_{u,1} = g^{\alpha u}, R_{u,2} = g^{R_{u,2} + \alpha u}, R_{u,3} = (R_{u,1} \cdot R_{u,0}^{\alpha u}), R_{u,4} = (g^{\alpha u})^{u_u}, \) and \(X_{u,2} = (g^{\alpha u})^{u_u}.\) The algorithm \(B\) sends \(pk_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})\) to the forger \(A_1\) as the set of public keys for the identity ID\(_u\) and updates Tab1.

   (b) If ID\(_u\) = ID\(_{tg}\), the algorithm \(B\) chooses random number \(e_u, r_u, w_u, k_u \leftarrow Z_p^*\) and calculates \(R_{u,0} = g^{k_u}, R_{u,1} = g^{\alpha u}, R_{u,2} = (R_{u,1} \cdot (g^{\beta u})^{\alpha u})^{\alpha u}, R_{u,3} = (R_{u,1} \cdot R_{u,0}^{\alpha u})^{\alpha u}, R_{u,4} = (g^{\alpha u})^{u_u}, \) and \(X_{u,2} = (g^{\alpha u})^{u_u}.\) The algorithm \(B\) sends \(pk_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})\) to the forger \(A_1\) as the set of public keys for the identity ID\(_u\) and updates Tab1.

4. The algorithm \(B\) simulates the hash oracles as follows:

   For Hash Oracle \(H\). At any time the forger \(A_1\) can query the random oracle \(H\) for \((ID_u, R_{u,1})\). To respond to the hash queries, the algorithm \(B\) retrieves the tuple which includes the queried identity ID\(_u\) in Tab1. Then the algorithm \(B\) returns \(H(ID_u, R_{u,1}) = e_u\) in Tab1 to the forger \(A_1\).

   For Hash Oracle \(H\). At any time the forger \(A_1\) can query the random oracle \(H\) for \((ID_u, pk_u, id_u)\). To respond to the hash queries, the algorithm \(B\) maintains a table Tab2 = \((ID_u, pk_u, id_u, \xi_{u,j})\), where Tab2 is initially empty. When the forger \(A_1\) queries \((ID_u, pk_u, id_u)\), the algorithm \(B\) responds as follows:

   (a) If \((ID_u, pk_u, id_u)\) \notin Tab2, the algorithm \(B\) picks a random number \(\xi_{u,j} \leftarrow Z_p^*\) and sets the hash value \(\tilde{H}_{u,j} = \xi_{u,j}\). Then the algorithm \(B\) stores the tuple Tab2 = \((ID_u, pk_u, id_u, \xi_{u,j})\) and responds with \(H(pk_u, id_u) = \xi_{u,j}\) to the forger \(A_1\).

   (b) If \((ID_u, pk_u, id_u)\) \in Tab2, the algorithm \(B\) returns \(\tilde{H}(pk_u, id_u) = \xi_{u,j}\) in Tab2 to the forger \(A_1\).

For Hash Oracle \(h\). At any time the forger \(A_1\) can query the random oracle \(h\) for \((fname, id)\). To respond to the hash queries, the algorithm \(B\) maintains a table Tab3 = \((fname, id, \delta, \epsilon, h_{i})\), where Tab3 is initially empty. When the forger \(A_1\) queries \((fname, id)\), the algorithm \(B\) responds as follows:

   (a) If \((fname, id)\) \notin Tab3, the algorithm \(B\) flips a random coin \(c_i \in \{0, 1\}\) with a probability \(Pr[c_i = 0] = 1 / (q_i + 1)\), where \(q_i\) is the maximum number of TagGen queries. The algorithm \(B\) picks a random number \(\delta_i \in Z^*_p\) and computes \(h_i = (g^{\delta_i}) \cdot g^b\). Then the algorithm \(B\) stores the tuple Tab3 = \((fname, id, \delta, \epsilon, h_{i})\) and returns \(h(fname, id) = h_i\) to the forger \(A_1\).

   (b) If \((fname, id)\) \in Tab3, the algorithm \(B\) responds with \(h(fname, id) = h_i\) in Tab3.

5. The algorithm \(B\) simulates Partial-Private-Key-Extract oracle, Public-Key-Replacement oracle, Secret-Value-Extract oracle, and TagGen oracle as follows.

   For Partial-Private-Key-Extract Oracle. At any time the forger \(A_1\) can request a partial private key for ID\(_u\) to the oracle. When the forger \(A_1\) queries ID\(_u\) to the oracle, the algorithm \(B\) responds as follows:

   (a) If ID\(_u\) \neq ID\(_{tg}\), the algorithm \(B\) retrieves the tuple which contains the queried ID\(_u\) in Tab1 and returns \(psk_u = (R_{u,1}, R_{u,2}, \sigma_u)\) to the forger \(A_1\).

   (b) If ID\(_u\) = ID\(_{tg}\), the algorithm \(B\) reports failure.

   For Public-Key-Replacement Oracle. At any time the forger \(A_1\) can replace a set of public keys for ID\(_u\) on receiving a public key replacement query \((ID_u, pk_u)\). \(A_1\) retrieves the tuple which contains ID\(_u\) in Tab1 and replaces \(pk_u\) with \(pk_u\).

   For Secret-Value-Extract Oracle. At any time the forger \(A_1\) can request a secret value for ID\(_u\) to the oracle. When the forger \(A_1\) queries ID\(_u\) to the oracle, \(A_1\) retrieves the tuple which contains ID\(_u\) in Tab1 and responds with \((w_u, k_u)\).

   For TagGen Oracle. At any time the forger \(A_1\) can query the oracle for the tuple \((ID_u, id_u, m_u)\). The algorithm \(B\) stores Tab4 = \((ID_u, id_u, m_u, \xi_u, \epsilon_u, w_u, k_u, \sigma_u, \xi_{u,j}, \delta, h_{i}, c_i, t_i)\), where Tab4 is initially empty. When the forger \(A_1\) queries ID\(_u\), the algorithm \(B\) responds as follows:

   (a) If ID\(_u\) \neq ID\(_{tg}\), the algorithm \(B\) retrieves the tuple which is corresponding to the queried tuple \((ID_u, id_u)\) and computes the tag \(t_i\) for the
quered message \( m_i \), where \( t_i = (g_i, h_i, g_{f_{\alpha}^{i}})^{u_i} \). The algorithm \( B \) responds with \( t_i \) and updates Tab4.

(b) If \( ID_u = ID_{2g^s} \) the algorithm \( B \) retrieves the tuple which is corresponding to the quer ed tuple \((ID_u, id_i)\). If \( c_i = 1 \), the algorithm \( B \) computes the tag \( t_i \) for the queried message \( m_i \), where \( t_i = (g_i, h_i, g_{f_{\alpha}^{i}})^{u_i} \). The algorithm \( B \) responds with \( t_i \) and updates Tab4. Otherwise, the algorithm \( B \) reports failure.

(6) Finally, the forger \( \mathcal{A}_f \) outputs the forged pair of message and tag \((ID^*, id_i, m_i, t_i)\) such that no \( TagGen \) query was issued for \( m_i \). If \( id_i \in \text{Tab3} \), \( c_i = 0 \), and \( ID^* = ID_{2g^s} \), \( t_i \) satisfies the following equation:

\[
e(t_i, g) = e \left( g_1^{f_{\alpha}^{i}}, R_{u,3} \cdot X_{H_{u,2}} \right) \cdot e \left( h_i \cdot g_{2}^{f_{\alpha}^{i}}, R_{u,1} \cdot X_{H_{u,2}} \right) .
\]

(7)

By using the oracle-replay technique [37] on hash function \( H \), the algorithm \( B \) can obtain another forged tag \( t_i' \) for the same message \( m_i \). The forged tag \( t_i' \) satisfies the following equation:

\[
e(t_i', g) = e \left( g_1^{f_{\alpha}^{i}}, R_{u,3} \cdot X_{H_{u,2}} \right) \cdot e \left( h_i \cdot g_{2}^{f_{\alpha}^{i}}, R_{u,1} \cdot X_{H_{u,2}} \right) .
\]

(8)

With the above equations, the algorithm \( B \) could compute

\[
e \left( \frac{t_i}{t_i'}, g \right) = e \left( g_1^{f_{\alpha}^{i}}, X_{H_{u,2}} \right) \cdot e \left( h_i \cdot g_{2}^{f_{\alpha}^{i}}, X_{H_{u,2}} \right) .
\]

(9)

From the right-hand-side of the above equation, the algorithm \( B \) could get

\[
e \left( g_1^{f_{\alpha}^{i}}, X_{H_{u,2}} \right) \cdot e \left( h_i \cdot g_{2}^{f_{\alpha}^{i}}, X_{H_{u,2}} \right) = e \left( (g^a)^{u_i} \cdot (g^a)^{u_i} \cdot (g^a)^{u_i} \cdot (g^a)^{u_i} \right) .
\]

(10)

Then the algorithm \( B \) outputs

\[
g^{f_{\alpha}^{i}} = \left( \frac{t_i}{t_i'} \right)^{1/\delta} \cdot (g^a)^{\delta} ,
\]

(11)

where \( C = (w_i z_i H_{u,2} + z_i f_{\alpha}^{i}(a))(H_u - H_{u}') \).

From now on, we define three events related to the success probability of \( B \) and analyze it as follows:

(i) \( E_1 \): the algorithm \( B \) does not abort in the \( \text{Partial-Private-Key-Extract} \) queries and \( TagGen \) queries.

(ii) \( E_2 \): the forger \( \mathcal{A}_f \) outputs a valid pair of message and tag \((ID^*, id_i, m_i, t_i)\), which has not been queried before.

(iii) \( E_3 \): event \( E_3 \) occurs and \( c_i = 0 \) for the tuple containing \( id_i \) on Tab3.

In the case that all events happen, the algorithm \( B \) can solve the CDH problem. Thus, the probability of success is

\[
Pr \left[ E_1 \land E_2 \right] = Pr \left[ E_1 \right] \cdot Pr \left[ E_2 \mid E_1 \right] .
\]

(12)

We analyze the lower bounds of \( Pr \left[ E_1 \right] \), \( Pr \left[ E_2 \mid E_1 \right] \), and \( Pr \left[ E_1 \mid E_2 \land E_3 \right] \) in the following claims.

Claim 1. The probability \( Pr \left[ E_1 \right] \) that the algorithm \( B \) does not abort in the \( \text{Partial-Private-Key-Extract} \) queries and \( TagGen \) queries by the forger \( \mathcal{A}_f \) is at least \( 1/e^2 \).

Proof of Claim 1. We assume that the forger \( \mathcal{A}_f \) does not query for the same index twice. At first, in the \( \text{Partial-Private-Key-Extract} \) oracle the algorithm \( B \) cannot respond to the forger \( \mathcal{A}_f \)'s query, if \( ID_u = ID_{2g^s} \). This probability is equal to \( 1/q_k \), where \( q_k \) is the number of \( KeyGen \) queries. Therefore, the probability that the algorithm \( B \) does not abort is \( 1 - 1/q_k \) for each \( \text{Partial-Private-Key-Extract} \) query. Since the \( \text{Partial-Private-Key-Extract} \) queries are issued at most \( q_k \) times, the probability that the algorithm \( B \) does not abort is at least \( 1 - 1/q_k^q \geq 1/e \). In the \( TagGen \) oracle, the algorithm \( B \) only aborts if \( c_i = 0 \), where \( c_i \) is the random value with the probability \( Pr[c_i = 0] = 1/(q_k + 1) \) corresponding to the \( TagGen \) query \( id_i \) in Tab3. By the probability of \( c_i \), the probability that the algorithm \( B \) does not abort is \( 1 - 1/(q_k + 1) \). Similarly, the probability that the algorithm \( B \) does not
abort in the TagGen oracle is at least \((1 - 1/(q_i + 1))^{\theta_i} \geq 1/e\). Thus, the probability \(Pr[E_1]\) that the algorithm \(\mathcal{B}\) does not abort in the forger \(\mathcal{A}_1\)'s PARTIAL-PRIVATE-KEY-EXTRACT queries and TagGen queries is at least \((1 - 1/q_i)^{\theta_i} \cdot (1 - 1/(q_i + 1))^{\theta_i} \geq 1/e^2\).

Claim 2. The probability \(Pr[E_2 \mid E_1]\) that the forger \(\mathcal{A}_2\) 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 \(E_1\) has occurred, the probability \(Pr[E_2 \mid E_1]\) depends on the information gathered by the forger \(\mathcal{A}_1\). The views of \(\mathcal{A}_1\) in the simulation and the real game are identical as follows:

(i) The distribution of PK given to the forger \(\mathcal{A}_1\) in the simulation is identical with the distribution of PK given to the forger \(\mathcal{A}_1\) in the real game.

(ii) The distribution of the hash values of the hash function in the simulation is identical with the distributions of the hash values of the hash function in the real game.

(iii) The distribution of the responses to the TagGen oracle in the simulation is identical with the distribution of PK given to the forger \(\mathcal{A}_1\) in the real game under the conditional probability.

Thus, the forger \(\mathcal{A}_1\) will generate a valid pair of message and tag with a probability at least \(\varepsilon\).

Claim 3. The probability \(Pr[E_2 \mid E_1 \wedge E_2]\) that the algorithm \(\mathcal{B}\) does not abort after the forger \(\mathcal{A}_2\) outputs a valid pair of message and tag is at least \(1/(q_i + 1)\).

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

With the above claims, the success probability of \(\mathcal{B}\) is \(e/(e^2 \cdot (q_i + 1))\). Thus, we can conclude that if the CDH assumption holds in \(G\), there exists no algorithm which breaks the tag-unforgeability of our scheme by the public key replacement attack with a nonnegligible probability.

Theorem 5. The proposed certificateless public auditing scheme is secure against the tag-forgery by Type II adversary under chosen message attacks in the random oracle model, if the CDH assumption holds in \(G\).

Proof of Theorem 5. Suppose Type II adversary \(\mathcal{A}_2\) is the tag-forgery that could break the scheme by generating a valid pair of message and tag under user’s secret key with a probability \(\varepsilon\). Then we can construct an algorithm \(\mathcal{B}\) that solves the CDH problem on \(G\).

Given an instance \((g, g^a, g^b)\) in \(G\) of the CDH problem, the algorithm \(\mathcal{B}\) simulates the challenger and interacts with the forger \(\mathcal{A}_2\) in Game II.

1. The forger \(\mathcal{A}_2\) outputs an identity \(ID_{tg}\) where it wishes to be forged.

2. The algorithm \(\mathcal{B}\) chooses random numbers \(\alpha, x, z_1, z_2 \leftarrow Z_q^*\) and computes \(\{g_2^\alpha\}_{0 \leq j \leq s+1}\), where \(g_2 = g^{z_2}\). The algorithm \(\mathcal{B}\) sets \(X_i = g^\alpha\) and \(g_1 = g^{z_1}\) and sends “PP and MK” to the forger \(\mathcal{A}_2\).

3. For KeyGen oracle, at any time the forger \(\mathcal{A}_2\) can query the KeyGen oracle for an identity \(ID\). The algorithm \(\mathcal{B}\) stores Tab1 = \((ID_u, \epsilon_u, r_u, \sigma_u, u, w_u, k_u, R_u, pk_u)\), where Tab1 is initially empty. When the forger \(\mathcal{A}_2\) queries \(ID_u\), the algorithm \(\mathcal{B}\) responds as follows:

   a. If \(ID_u \neq ID_{tg}\), the algorithm \(\mathcal{B}\) chooses random number \(e_u, r_u, w_u, k_u \leftarrow Z_q^*\) and calculates \(R_{u,0} = g^{k_u}, R_{u,1} = g^e, \sigma_u = r_u + xe_u, \sigma_u = k_u + r_u + e_u, R_{u,2} = g^{\sigma_u}, R_{u,3} = (R_{u,1} \cdot R_{u,3})^{w_u}\), and \(X_{u,2} = X_{u,1}^{w_u}\). The algorithm \(\mathcal{B}\) sends \(pk_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})\) to the forger \(\mathcal{A}_2\) as the set of public keys for the identity \(ID_u\) and updates Tab1.

   b. If \(ID_u = ID_{tg}\), the algorithm \(\mathcal{B}\) chooses random number \(e_u, r_u, w_u, k_u \leftarrow Z_q^*\) and calculates \(\sigma_u = r_u + xe_u, R_{u,0} = g^e, R_{u,1} = g^e, R_{u,2} = g^{\sigma_u}, R_{u,3} = (R_{u,1} \cdot R_{u,3})^{w_u}\), and \(X_{u,2} = X_{u,1}^{w_u}\). The algorithm \(\mathcal{B}\) sends \(pk_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})\) to the forger \(\mathcal{A}_2\) as the set of public keys for the identity \(ID_u\) and updates Tab1.

4. The algorithm \(\mathcal{B}\) simulates the hash oracles as follows.

For Hash Oracle \(H\). At any time the forger \(\mathcal{A}_2\) can query the random oracle \(H\) for \((ID_u, R_{u,1})\). To respond to the hash queries, the algorithm \(\mathcal{B}\) retrieves the tuple which includes the queried identity \(ID_u\) in Tab1. Then the algorithm \(\mathcal{B}\) returns \(H(ID_u, R_{u,1}) = \epsilon_u\) in Tab to the forger \(\mathcal{A}_2\).

For Hash Oracle \(\overline{H}\). At any time the forger \(\mathcal{A}_2\) can query the random oracle \(\overline{H}\) for \((ID_u, pk_u, id_u)\). To respond to the hash queries, the algorithm \(\mathcal{B}\) maintains a table Tab2 = \((ID_u, pk_u, id_u, \xi_{u,2})\), where Tab2 is initially empty. When the forger \(\mathcal{A}_2\) queries \((ID_u, pk_u, id_u)\), the algorithm \(\mathcal{B}\) responds as follows:

   a. If \((ID_u, pk_u, id_u) \notin \text{Tab2}\), the algorithm \(\mathcal{B}\) picks a random number \(\xi_{u,1} \leftarrow Z_q^*\) and sets the hash value \(\overline{H} = \xi_{u,1}\). Then the algorithm \(\mathcal{B}\) stores the tuple \(T_{u,2} = (ID_u, pk_u, id_u, \xi_{u,2})\) and responds with \(H(pk_u, id_u) = \xi_{u,1}\) to the forger \(\mathcal{A}_2\).

   b. If \((ID_u, pk_u, id_u) \in \text{Tab2}\), the algorithm \(\mathcal{B}\) returns \(\overline{H}(pk_u, id_u) = \xi_{u,2}\) in Tab2 to the forger \(\mathcal{A}_2\).

For Hash Oracle \(h\). At any time the forger \(\mathcal{A}_2\) can query the random oracle \(h\) for \((fname, id_u)\). To respond to the hash queries, the algorithm \(\mathcal{B}\) maintains a table Tab3 = \((fname, id_u, \delta_u, c_u, h_u)\), where
Tab3 is initially empty. When the forger $A_2$ queries $(fname, id_i)$, the algorithm $B$ responds as follows:

(a) If $(fname, id_i) \notin Tab3$, the algorithm $B$ flips a random coin $c_i \in \{0, 1\}$ with a probability $Pr[c_i = 0] = 1/(q_i + 1)$, where $q_i$ is the maximum number of $TagGen$ queries. The algorithm $B$ picks a random number $\delta_i \in \mathbb{Z}_n^*$ and computes $h_i = (g_1^{(1-c)i} \cdot g_i)^{\delta_i}$. Then the algorithm $B$ stores the tuple $Tab3 = (fname, id_i, \delta_i, c_i, h_i)$ and returns $H(fname, id_i) = h_i$ to the forger $A_2$.

(b) If $(fname, id_i) \in Tab3$, the algorithm $B$ responds with $h(fname, id_i) = h_i$ in Tab3.

(5) The algorithm $B$ simulates the Partial-Private-Key-Extract oracle, Secret-Value-Extract oracle, and TagGen oracle as follows.

For Partial-Private-Key-Extract Oracle. At any time the forger $A_2$ can request a partial private key for $ID_u$ to the oracle. When the forger $A_2$ queries $ID_u$ to the oracle, $B$ retrieves the tuple which contains $ID_u$ in Tab1 and responds with $(R_{u,0}, R_{u,1}, \overline{H}_u)$.

For Secret-Value-Extract Oracle. At any time the forger $A_2$ is able to request a secret value for $ID_u$ to the oracle. When the forger $A_2$ queries $ID_u$ to the oracle, $B$ responds as follows:

(a) If $ID_u \neq ID_{tgr}$, the algorithm $B$ retrieves the tuple which contains the queried $ID_u$ in Tab1 and returns $(w_u, k_u)$ to the forger $A_2$.

(b) If $ID_u = ID_{tgr}$, the algorithm $B$ reports failure.

For TagGen Oracle. At any time the forger $A_2$ can request the tag for the tuple $(ID_u, id_i, m_i)$. The algorithm $B$ stores Tab4 = $(ID_u, id_i, m_i, \epsilon_u, r_u, w_u, k_u, \delta_u, \xi_u, \delta_i, h_i, c_i, t_i)$, where Tab4 is initially empty. When the forger $A_2$ queries $ID_u$, the algorithm $B$ responds as follows:

(a) If $ID_u \neq ID_{tgr}$, the algorithm $B$ retrieves the tuple which is corresponding to the queried tuple $(ID_u, id_i)$ and computes the tag $t_i$ for the queried message $m_i$, where $t_i = (g_1^{-\overline{w}_hu} \cdot h_i^\gamma, g_2^{f_{\xi}^\delta(a)})$. The algorithm $B$ responds with $t_i$ and updates Tab4.

(b) If $ID_u = ID_{tgr}$, the algorithm $B$ retrieves the tuple which is corresponding to the queried tuple $(ID_u, id_i)$. If $c_i = 1$, the algorithm $B$ computes the tag $t_i$ for the queried message $m_i$, where $t_i = (g_1^{-\overline{w}_hu} \cdot h_i^\gamma, g_2^{f_{\xi}^\delta(a)}, g_3^{w_{\overline{H}_u} + \delta_i + z_{f_{\xi}^\delta(a)}^\delta_i})$. The algorithm $B$ responds with $t_i$ and updates Tab4. Otherwise, the algorithm $B$ reports failure.

(6) Finally, the forger $A_2$ outputs the forged pair of message and tag $(ID^*, id^*_i, m^*_i, t^*_i)$ such that no TagGen query was issued for $m^*_i$. If id, $\in Tab3$, $c_i = 0$, and $ID^* = ID_{tgr}$, $t^*_i$ satisfies the equation

$$e(t^*_i, g) = e(g_1^{\overline{H}_u}, R_{u,3} \cdot X_{u2}^H) \cdot e(h_i^\gamma, R_{u,0} \cdot R_{u,1} \cdot X_{u1}^H). \tag{13}$$

From the right-hand-side of the above equation, the algorithm $B$ could compute

$$e(g_1^{\overline{H}_u}, R_{u,3} \cdot X_{u2}^H) \cdot e(h_i^\gamma, R_{u,0} \cdot R_{u,1} \cdot X_{u1}^H) = e(g_1^{\overline{w}_hu}, g_2^{f_{\xi}^\delta(a)}) \cdot e(g_3^{w_{\overline{H}_u} + \delta_i + z_{f_{\xi}^\delta(a)}^\delta_i}, g_4^{f_{\xi}^\delta(a)}). \tag{14}$$

Then the algorithm $B$ outputs

$$g^{ab} = \frac{t^*_i}{(g_1^{\overline{w}_hu})^{\theta_1}} \cdot (g_1^{\overline{w}_hu})^{\theta_2} \cdot (g_1^{\overline{w}_hu})^{\theta_3}, \tag{15}$$

where $\theta_1 = w_u z_1 H_{u2} + \delta + z_{f_{\xi}^\delta(a)}^\delta, \theta_2 = r_u + x H_u$, and $\theta_3 = r_u + x H_u (w_u z_1 H_{u2} + \delta + z_{f_{\xi}^\delta(a)}^\delta)$. From now on, we define three events related to the success probability of $B$ and analyze it as follows:

(i) $E_4$: the algorithm $B$ does not abort in the Secret-Value-Extract queries and TagGen queries.

(ii) $E_5$: the forger $A_2$ outputs a valid pair of message and tag $(ID^*, id^*_i, m^*_i, t^*_i)$, which has not been queried before.

(iii) $E_6$: event $E_5$ occurs and $c_i = 0$ for the tuple containing id, $\in Tab3$. 
In the case that the all events happen, the algorithm \( \mathcal{B} \) can solve the CDH problem. Thus, the probability of success is 
\[
\Pr[E_4 \land E_6] = \Pr[E_4] \cdot \Pr[E_3 \mid E_4] 
\]
\[
\cdot \Pr[E_6 \mid E_4 \land E_5].
\]

We analyze the lower bounds of \( \Pr[E_1], \Pr[E_3 \mid E_4] \), and \( \Pr[E_6 \mid E_4 \land E_5] \) in the following claims.

**Claim 4.** The probability \( \Pr[E_1] \) that the algorithm \( \mathcal{B} \) does not abort in the Secret-Value-Extract queries and the TagGen queries by the forger \( \mathcal{A}_2 \) is at least \( 1/e^2 \).

**Claim 5.** The probability \( \Pr[E_3 \mid E_4] \) that the forger \( \mathcal{A}_2 \) outputs a valid pair of message and tag under the condition that the event \( E_4 \) has occurred is at least \( \epsilon \).

**Claim 6.** The probability \( \Pr[E_6 \mid E_4 \land E_5] \) that the algorithm \( \mathcal{B} \) does not abort after the forger \( \mathcal{A}_3 \) outputs a valid pair of message and tag is at least \( 1/(q_0 + 1) \).

By the similar analysis method in Theorem 4, we can easily compute \( \mathcal{B} \)'s success probability, thus we omit the detailed probability analyses. With the above claims, the success probability of \( \mathcal{B} \) is \( \epsilon/(e^2 \cdot (q_0 + 1)) \). Thus we can conclude that if the CDH assumption holds in \( G \), there exists no algorithm which breaks the tag-unforgeability of our scheme by the malicious-but-passive-KGC attack with a nonnegligible probability. \( \square \)

**Theorem 6.** The proposed certificateless public auditing scheme is secure against the proof-forgery by Type III adversary under chosen message attacks in the random oracle model, if the DCDH assumption holds in \( G \).

**Proof of Theorem 6.** Suppose Type III adversary \( \mathcal{A}_3 \) is the proof-forgery that could break the scheme by generating a fake proof for an arbitrary challenge message with a probability \( \epsilon \). We do not consider the case that the fake proof was generated by the forged tags, since we proved it in the above theorems (even though the theorems ensure the security against the tag-forgery by Type I adversary \( \mathcal{A}_1 \) and Type II adversary \( \mathcal{A}_2 \), it can be easily applied to Type III adversary \( \mathcal{A}_3 \) by eliminating the oracles related to the key extraction in the proof of Theorems 4 and 5). Using the proof-forgery, we can construct an algorithm \( \mathcal{B} \) that solves the DCDH problem on \( G \).

Given an instance \((g, g^a, g^b) \in G \) of the DCDH problem, the algorithm \( \mathcal{B} \) simulates the challenger and interacts with the forger \( \mathcal{A}_3 \) in Game III.

1. The forger \( \mathcal{A}_1 \) outputs an identity \( ID_{tg} \) where it wishes to be forged.
2. The algorithm \( \mathcal{B} \) chooses random numbers \( a, x, z_1, z_2 \leftarrow Z_q^* \) and computes \( g_2 = g^{z_1} \). The algorithm \( \mathcal{B} \) sets \( X_1 = g^x \) and \( g_1 = g^{z_2} \) and sends PP to the forger \( \mathcal{A}_3 \).
3. For KeyGen oracle, at any time the forger \( \mathcal{A}_3 \) can query the KeyGen oracle for an identity \( ID_a \). The algorithm \( \mathcal{B} \) stores \( Tab_l = (ID_a, \epsilon_a, r_a, \sigma_a, \sigma_u, w_u, k_u, Y_u, \overline{R}_{u,2}, pk_u) \), where Tab1 is initially empty. When the forger \( \mathcal{A}_3 \) queries \( ID_u \), the algorithm \( \mathcal{B} \) responds as follows:

   (a) If \( ID_u \neq ID_{tg} \), the algorithm \( \mathcal{B} \) chooses random number \( \epsilon_u, r_u, w_u, k_u \leftarrow Z_q^* \) and calculates 
   \[
   R_{u,0} = g^{\epsilon_u}, \quad R_{u,1} = g^{r_u}, \quad \overline{R}_{u,2} = r_u + xe_u, \quad \sigma_u = k_u + r_u + e_v, \quad R_{u,2} = g^{\sigma_u}, \quad R_{u,3} = (R_{u,1} \cdot R_{u,0})^{x_u}, \quad \text{and} \quad X_{u,2} = X_1. 
   \]
   The algorithm \( \mathcal{B} \) sends \( pk_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2}) \) to the forger \( \mathcal{A}_3 \) as the set of public keys for the identity \( ID_u \) and updates Tab1.

(b) If \( ID_u = ID_{tg} \), the algorithm \( \mathcal{B} \) chooses random number \( \epsilon_u, r_u, w_u \leftarrow Z_q^* \) and calculates \( \overline{\sigma}_u = g^{\epsilon_u} \). The algorithm \( \mathcal{B} \) chooses \( \overline{\sigma}_u \) as \( (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2}) \) to the forger \( \mathcal{A}_3 \) as the set of public keys for the identity \( ID_u \) and updates Tab1.

(4) The algorithm \( \mathcal{B} \) simulates the hash oracles as follows.

**For Hash Oracle \( H \).** At any time the forger \( \mathcal{A}_3 \) can query the random oracle \( H \) for \((ID_u, R_{u,1})\). The algorithm \( \mathcal{B} \) retrieves the tuple which includes the queried identity \( ID_u \) in Tab1. Then the algorithm \( \mathcal{B} \) returns \( H(ID_u, R_{u,1}) = \epsilon_u \) in Tab1 to the forger \( \mathcal{A}_3 \).

**For Hash Oracle \( \overline{H} \).** At any time the forger \( \mathcal{A}_3 \) can query the random oracle \( \overline{H} \) for \((ID_u, pk_u, id_u)\). To respond to the hash queries, the algorithm \( \mathcal{B} \) maintains a table \( Tab2 = (ID_u, pk_u, id_u, \xi_{u,j}) \), where \( Tab2 \) is initially empty. When the forger \( \mathcal{A}_3 \) queries \( ID_u, pk_u, id_u \), the algorithm \( \mathcal{B} \) responds as follows:

   (a) If \((ID_u, pk_u, id_u) \notin Tab2 \), the algorithm \( \mathcal{B} \) picks a random number \( \xi_{u,j} \in Z_q^* \) and sets \( H(ID_u, R_{u,1}) = \xi_{u,j} \). Then the algorithm \( \mathcal{B} \) stores the tuple \( Tab2 = (ID_u, pk_u, id_u, \xi_{u,j}) \) and responds with \( H(pk_u, id_u) = \xi_{u,j} \) to the forger \( \mathcal{A}_3 \).

   (b) If \((ID_u, pk_u, id_u) \in Tab2 \), the algorithm \( \mathcal{B} \) returns \( H(pk_u, id_u) = \xi_{u,j} \) in Tab2 to the forger \( \mathcal{A}_3 \).

**For Hash Oracle \( h \).** At any time the forger \( \mathcal{A}_3 \) can query the random oracle \( h \) for \((fname, id)\). To respond to the hash queries, the algorithm \( \mathcal{B} \) maintains a table \( Tab3 = (fname, id, \xi_{u,j}) \), where \( Tab3 \) is initially empty. When the forger \( \mathcal{A}_3 \) queries \( fname, id_u \), the algorithm \( \mathcal{B} \) responds as follows:

   (a) If \((fname, id_u) \notin Tab3 \), the algorithm \( \mathcal{B} \) picks a random number \( \xi_{u,j} \in Z_q^* \) and computes \( h_j = g^{\xi_{u,j}} \). Then the algorithm \( \mathcal{B} \) stores the tuple \( Tab3 = (fname, id_u, \xi_{u,j}) \) and returns \( h(fname, id_u) = h_j \) to the forger \( \mathcal{A}_3 \).
Finally, the forger contains at least an index id to the oracle, For Partial-Private-Key-Extract algorithm, the forger to the oracle, $\mathcal{B}$ responds as follows:

(a) If $ID_u \neq ID_{\ell^*}$, the algorithm $\mathcal{B}$ retrieves the tuple which contains the queried $ID_u$ in Tab 1 and returns $(\bar{R}_{u,1}, \bar{R}_{u,2}, \bar{\sigma}_u)$ to the forger $\mathcal{A}_3$.

(b) If $ID_u = ID_{\ell^*}$, the algorithm $\mathcal{B}$ reports failure.

For Secret-Value-Extract Oracle. At any time the forger $\mathcal{A}_3$ is able to request a secret value for $ID_u$ to the oracle. When the forger $\mathcal{A}_3$ queries $ID_u$ to the oracle, $\mathcal{B}$ responds as follows:

(a) If $ID_u \neq ID_{\ell^*}$, the algorithm $\mathcal{B}$ retrieves the tuple which contains the queried $ID_u$ in Tab 1 and returns $(w_u, k_u)$ to the forger $\mathcal{A}_3$.

(b) If $ID_u = ID_{\ell^*}$, the algorithm $\mathcal{B}$ reports failure.

For TagGen Oracle. At any time the forger $\mathcal{A}_3$ can query the oracle for the tuple $(ID_u, id, m_i)$. The algorithm $\mathcal{B}$ stores $\mathcal{T}_4 = (ID_u, id, m_i, e_u, r_u, w_u, k_u, t_i, \sigma_u, \xi_u, \delta_i, h_i, t_i)$, where Tab 4 is initially empty. When the forger $\mathcal{A}_3$ queries $ID_u$, the algorithm $\mathcal{B}$ responds as follows:

(a) If $ID_u \neq ID_{\ell^*}$, the algorithm $\mathcal{B}$ retrieves the tuple which is corresponding to the queried tuple $(ID_u, id)$ and computes the tag $t_i$ for the queried message $m_i$, where $t_i = (g_i^{w_u}, h_i, g_i^{f(\alpha)})^\alpha$. The algorithm $\mathcal{B}$ responds with $t_i$ and updates Tab 4.

(b) If $ID_u = ID_{\ell^*}$, the algorithm $\mathcal{B}$ retrieves the tuple which is corresponding to the queried tuple $(ID_u, id)$. The algorithm $\mathcal{B}$ computes the tag $t_i$ for the queried message $m_i$, where $t_i = (g_i^{w_u}, h_i, g_i^{f(\alpha)})^\alpha$.

The algorithm $\mathcal{B}$ chooses random number $r_c \leftarrow Z_q^*$ and a random $d$-elements subset $D \subset [1, n]$. In this process, $\mathcal{B}$ should choose a random subset which contains at least an index $id$, where $id$ is not queried before in TagGen oracle. Let $D'$ be a subset of $D$, and it is composed of indices which are not queried in TagGen oracle. Then the algorithm $\mathcal{B}$ sends $(r_c, D')$ as the challenge message.

Finally, the forger $\mathcal{A}_3$ outputs a forged proof for the challenge message $(T, \rho^*, y^*, \psi^*)$. We assume that $T$ is not able to be forged by the forger $\mathcal{A}_3$, since we proved the tag-unforgeability in the previous theorems. Then the algorithm checks the validity of the forged proof by the following equation:

$$
\begin{align*}
e(T, g) &= e\left(g_1^{\pi}, g^b\right) \cdot e\left(\rho^*, g_2^{y^*}, g^a\right) \\
&\quad \cdot e\left(\psi^*, R_{u,2} \cdot g^{a(\tau_3)}\right).
\end{align*}
$$

(17)

The algorithm $\mathcal{B}$ could compute the right-hand-side of the above equation as

$$
\begin{align*}
e\left(g_1^{\pi}, g^b\right) \cdot e\left(\rho^*, g_2^{y^*}, g^a\right) &= e\left(g^{z_{\pi}}, g^b\right) \cdot e\left(\rho^* \cdot g^{z_{y^*}}, g^a\right) \cdot e\left(\psi^*, g^{a(\tau_3)}\right) \\
&= e\left(g^{z_{\pi}}, g^b\right) \cdot e\left(\rho^* \cdot g^{z_{y^*}}, g^a\right) \cdot e\left(\psi^{a(\tau_3)}, g^a\right) \\
&= e\left(g^{z_{\pi}}, g^b\right) \cdot e\left(\rho^* \cdot g^{z_{y^*}}, \psi^{a(\tau_3)}, g^a\right).
\end{align*}
$$

(18)

To satisfy (18), the value $(\rho^* \cdot g^{z_{y^*}}) \cdot \psi^{a(\tau_3)}$ should contain $(g^{\alpha b})^{\pi'}$, where $\pi' = \sum_{i \in D'} \xi_i + p_i$. Thus, we can compute $g^{\alpha b}$ as

$$
g^{\alpha b} = \left(g^\rho \cdot g^{z_{y^*}} \cdot \psi^{a(\tau_3)}\right)^{\pi'} / g^a.
$$

(19)

where $\mathcal{C} = \sum_{i \in D'} \xi_i + p_i + z_{f(\alpha)}$.

From now on, we define two events related to the success probability of $\mathcal{B}$ and analyze it as follows:

(i) $E_5$: the algorithm $\mathcal{B}$ does not abort in the Partial-Private-Key-Extract queries and Secret-Value-Extract queries.

(ii) $E_6$: the forger $\mathcal{A}_3$ outputs a valid proof $(T, \rho^*, y^*, \psi^*)$, which has not been queried before.

In the case that both events happen, the algorithm $\mathcal{B}$ can solve the DCDH problem. Thus, the probability of success is $\Pr[E_5 \wedge E_6] = \Pr[E_5] \cdot \Pr[E_6]$. We analyze the lower bounds of $\Pr[E_5]$ in the following claim.

Claim 7. The probability $\Pr[E_5]$ that the algorithm $\mathcal{B}$ does not abort in the Partial-Private-Key-Extract queries and Secret-Value-Extract queries by the forger $\mathcal{A}_1$ is at least $1/|e|^2$.

Proof of Claim 7. We assume that the forger $\mathcal{A}_1$ does not query for the same index twice. In the Partial-Private-Key-Extract oracle, the algorithm $\mathcal{B}$ aborts only if $ID_u = ID_{\ell^*}$. This probability is equal to $1/q_k$, where $q_k$ is the number of Partial-Private-Key-Extract queries. Therefore, the probability that the algorithm $\mathcal{B}$ does not abort is $1 - 1/q_k$ for each Partial-Private-Key-Extract query. Since the Partial-Private-Key-Extract queries are issued at most $q_k$ times, the probability that the algorithm $\mathcal{B}$ does not abort is at least $(1 - 1/q_k)^{q_k} \geq 1/e$. In the Secret-Value-Extract oracle, the algorithm $\mathcal{B}$ only aborts if $ID_u = ID_{\ell^*}$. The probability that the algorithm $\mathcal{B}$ does not abort is $1 - 1/q_k$. Similarly, the probability that the algorithm $\mathcal{B}$ does not abort
in the Secret-Value-Extract oracle is at least \((1 - 1/q_i)\varepsilon^2 \geq 1/e\). Thus, the probability \(\Pr[E_\mathcal{B}]\) that the algorithm \(\mathcal{B}\) does not abort in the forger \(\mathcal{A}_2\)'s Partial-Private-Key-Extract queries and Secret-Value-Extract queries is at least \((1 - 1/q_i)^2 \varepsilon^2 \geq 1/e^2\).

With the above claim, the success probability of \(\mathcal{B}\) is \(\varepsilon/e^2\), since the probability \(\Pr[E_\mathcal{B}] = e\). Thus we can conclude that if the DCDH assumption holds in \(G\), there exists no algorithm which breaks the proof-unforgeability of our scheme with a nonnegligible probability.

According to Theorems 4, 5, and 6, we can get Theorem 7 for the security of our proposed scheme. The comparison of the security levels for the certificateless public auditing schemes are explained in Table 1.

**Theorem 7.** The proposed certificateless public auditing scheme is secure against the existential forgery under chosen message attack in the random oracle model, if the CDH and DCDH assumption hold in \(G\).

**Proof of Theorem 7.** By proving Theorems 4, 5, and 6, we have proved that our certificateless public auditing scheme is secure against Type I, Type II, and Type III adversaries, respectively. Thus, our certificateless public auditing scheme is secure against the existential forgery attacks defined in Section 3.2.

### 4.2. Efficiency Analysis

In this section, we compare the computational cost of our scheme with two certificateless public auditing schemes in [28, 29], when TPA performs the auditing in the verification algorithm. Since we aim to construct the scheme which can efficiently verify the proof for the challenge message, we just compare the verification time of the proof in Table 3 (in reality, the computing power of the server is sufficiently larger than that of the TPA (user), so we made an effort to reduce the TPA’s computational cost).

Let \(\text{Pair}\) be the bilinear pairing computation and \(\text{Mul}_G\) be the multiplication in group \(G\). Let \(\text{Hash}_G\) and \(\text{Exp}_G\) be the hash-to-point computation and the exponentiation in group \(G\), respectively. Scott et al. released implementation results for the operations with 3 GHz-Pentium IV using the MIRACL multiprecision library (ver.5.01) [38]. Their results are shown in Table 2.

### Table 2: Computational cost

<table>
<thead>
<tr>
<th>Computations</th>
<th>(\text{Pair})</th>
<th>(\text{Mul}_G)</th>
<th>(\text{Hash}_G)</th>
<th>(\text{Exp}_G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times (in ms)</td>
<td>3.16</td>
<td>1.17</td>
<td>&lt;1</td>
<td>0.62</td>
</tr>
</tbody>
</table>

### Table 3: Comparison between certificateless public auditing schemes. Let \(\text{Pair}\) be the bilinear pairing computation and \(\text{Mul}_G\) be the multiplication in group \(G\). Let \(\text{Hash}_G\) and \(\text{Exp}_G\) be the hash-to-point computation and the exponentiation in group \(G\), respectively. Let \(d\) be the number of the challenged blocks.

<table>
<thead>
<tr>
<th>Computational cost of TPA</th>
<th>(d = 300)</th>
<th>(d = 460)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[28] (3\cdot\text{Pair} + (2d - 1)\cdot\text{Mul}_G + (d + 1)\cdot\text{Hash}_G + (d + 1)\cdot\text{Exp}_G \approx (4.58 \times d + 9.93)) ms</td>
<td>1383.93 ms</td>
<td>2116.73 ms</td>
</tr>
<tr>
<td>[29] (2\cdot\text{Pair} + (d + 2)\cdot\text{Mul}_G + (d + 1)\cdot\text{Hash}_G + (d + 3)\cdot\text{Exp}_G \approx (2.79 \times d + 11.52)) ms</td>
<td>848.52 ms</td>
<td>1294.92 ms</td>
</tr>
<tr>
<td>Ours (4\cdot\text{Pair} + 7\cdot\text{Mul}_G + 6\cdot\text{Exp}_G \approx 23.93) ms</td>
<td>23.93 ms</td>
<td>23.93 ms</td>
</tr>
</tbody>
</table>

In [2], Atteniese et al. proved that TPA is able to detect any modification of the outsourced data with 95% or 99% probability by selecting 300 or 460 challenge blocks, respectively. In Table 3, we computed the execution time of the verification algorithm in the case of \(d = 300\) and \(d = 460\). Note that the computational cost of TPA in [28, 29] is linearly increasing by the number of challenged blocks \(d\). On the other hand, the computational cost of TPA in our scheme is constant with respect to the number of challenged blocks \(d\). In the case of \(c = 300\), our scheme is approximately 57.8 times and 35.4 times faster than the schemes in [28] and [29], respectively. In the case of \(c = 460\), our scheme is approximately 88.5 times and 54.1 times faster than the schemes in [28] and [29], respectively.

### 5. Conclusions

In the paper, we proposed a provably secure certificateless public auditing scheme with the constant verification time. We also defined the three security models and proved the security of our scheme under the security models.

To the best of our knowledge, no certificateless public auditing scheme for shared data with the constant verification time has been proposed. The shared data system means that the authorized users can share their data in the cloud storage systems. Thus, it would be interesting to construct a certificateless public auditing scheme with the constant verification time for shared data.

### Notations in Our Scheme

- \(q\): A prime order
- \(G, G_T\): Multiplicative cyclic groups of prime order \(q\)
$e$: A bilinear map $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$

$g_1, g_2$: Generators of group $\mathbb{G}$

$ID_u$: A user identity

$\text{psk}_u$: A set of partial private keys for the identity $ID_u$

$\text{sk}_u$: A set of secret keys for the identity $ID_u$

$\text{pk}_u$: A set of public keys for the identity $ID_u$

$F$: A file $F = \{m_1, \ldots, m_n\}$

$m_i$: An $i$th block of the file $F$

$id_i$: A block identifier of the block $m_i$

$f_{\text{name}}$: A file name

$f_{x}$: A polynomial with a coefficient vector $\beta_i$

$t_i$: An authenticated tag for the block $m_i$

$[1, n]$: The numbers in a range $1 \leq x \leq n$

$D$: $d$-element subset of $[1, n]$

$r_i$: A random number.

**Competing Interests**

The authors declare that there is no conflict of interests regarding the publication of this paper.

**Acknowledgments**

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2013R1A2A2A01068200).

**References**


Submit your manuscripts at https://www.hindawi.com