Hindawi
Security and Communication Networks
Volume 2017, Article ID 6758618, 14 pages
https://doi.org/10.1155/2017/6758618

Research Article

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 ensures 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 $\mathbb{G}$ and $\mathbb{G}_T$ be multiplicative cyclic groups with prime order q. A function $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map, if it satisfies the following:

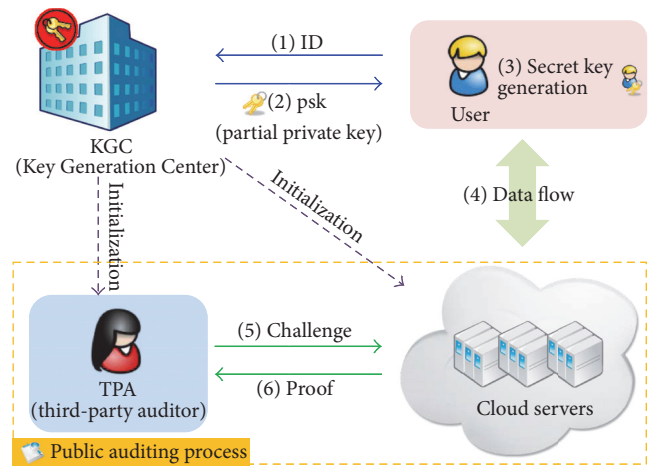


Figure 1: System model for our CL-PA protocol.

(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 $\mathbb{G}$, $e(g, g)$ is a generator of group $\mathbb{G}_T$.

(3) It is easy to compute $e(g^a, g^b)$ for all $g \in G$.

2.2. Complexity Assumptions

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

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

where the probability is over the random choice of $g \in \mathbb{G}$, 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 $g, g^a, g^b \in \mathbb{G}$, an algorithm $\mathcal{A}$ tries to compute $g^{a^{-1}b} \in \mathbb{G}$. We assume that there is no algorithm $\mathcal{A}$ with a nonnegligible probability ϵ such that

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

where the probability is over the random choice of $g \in \mathbb{G}$, 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 $\{t_i\}_{1 \le i \le n}$ 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 $\{m_i, t_i\}_{1 \le i \le n}$. 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 $\mathscr{A}_1$, Type II adversary $\mathscr{A}_2$, and Type III adversary $\mathscr{A}_3$, and define the adversaries as follows:

(i) *Type I adversary* $\mathscr{A}_1$ 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* $\mathscr{A}_2$, 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* $\mathscr{A}_3$, 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. If Type I adversary $\mathscr{A}_1$ 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 public key replacement attack. It is defined using the following game between a challenger $\mathscr{C}$ and Type I adversary $\mathscr{A}_1$:

(1) Type I adversary outputs an identity ID_{tg} where it wishes to be forged.

(2) The challenger $\mathscr{C}$ runs algorithm Setup to generate a public parameter (PP) and the master key (MK) and sends PP to Type I adversary $\mathscr{A}_1$.

(3) Type I adversary $\mathscr{A}_1$ is allowed to access the hash oracles, KeyGen, Partial-Private-Key-Extract, Public-Key-Replacement, Secret-Value-Extract, and TagGen oracle, adaptively. For KeyGen oracle, Type I adversary $\mathscr{A}_1$ sends ID_u to the challenger $\mathscr{C}$, and then the public key pk_u is returned. For Partial-Private-Key-Extract oracle, Type I adversary $\mathscr{A}_1$ sends ID_u to the challenger $\mathscr{C}$, and then the partial private key psk_u is returned. For Public-Key-Replacement oracle, Type I adversary $\mathscr{A}_1$ sends $(\mathrm{ID}_u, \mathrm{pk}'_u)$ to the challenger $\mathscr{C}$. For Secret-Value-Extract oracle, Type I adversary $\mathscr{A}_1$ sends ID_u to the challenger $\mathscr{C}$, and then the secret value $\mathrm{sv}_{\mathrm{ID}_u}$ is returned. For TagGen oracle, Type I adversary $\mathscr{A}_1$ sends $(\mathrm{ID}_u, \mathrm{id}_i, m_i)$, where id_i is the block identifier, to the challenger $\mathscr{C}$. The challenger $\mathscr{C}$ responds with a pair of message and tag (m_i, t_i), where $t_i = \mathrm{TagGen}(\mathrm{sk}_u, \mathrm{ID}_u, \mathrm{id}_i, m_i)$.

(4) Finally, Type I adversary $\mathscr{A}_1$ outputs a forged pair of message and tag $(\mathrm{id}_i, m^*, t^*)$ for an identity ID_{tg}.

If the pair $(\mathrm{id}_i, m^*, t^*)$ is valid pair of message and tag and m^* has not queried TagGen oracle, then Type I adversary $\mathscr{A}_1$ wins the game.

Definition 1. Type I adversary $\mathscr{A}_1$ $(t, q_h, q_k, q_{\mathrm{ppk}}, q_{\mathrm{pkr}}, q_{\mathrm{sv}}, q_t, \varepsilon)$-breaks a certificateless public auditing scheme, if $\mathscr{A}_1$ 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_{pkr} Public-Key-Replacement queries, q_{sv} Secret-Value-Extract queries, and q_t TagGen queries, and wins the game with the probability ε. A certificateless public auditing scheme is secure against a tag-forgery by the public key replacement attack, if there is no tag-forger which $(t, q_h, q_k, q_{\mathrm{ppk}}, q_{\mathrm{pkr}}, q_{\mathrm{sv}}, q_t, \varepsilon)$-breaks the scheme with a nonnegligible probability ε.

Game II. If Type II adversary $\mathscr{A}_2$ 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 $\mathscr{C}$ and Type II adversary $\mathscr{A}_2$.

(1) Type II adversary outputs an identity ID_{tg} where it wishes to be forged.

(2) The challenger $\mathscr{C}$ runs algorithm Setup to generate a public parameter (PP) and the master key (MK) and sends "PP and MK" to Type II adversary $\mathscr{A}_2$.

(3) Type II adversary $\mathscr{A}_2$ 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 $\mathscr{A}_2$ sends ID_u to the challenger $\mathscr{C}$, and then the public key pk_u is returned. For Partial-Private-Key-Extract oracle, Type II adversary $\mathscr{A}_2$ sends ID_u to the challenger $\mathscr{C}$, and then the partial private key psk_u is returned. For Secret-Value-Extract oracle, Type II adversary $\mathscr{A}_2$ sends ID_u to the challenger $\mathscr{C}$, and then the secret value $\mathrm{sv}_{\mathrm{ID}_u}$ is returned. For TagGen oracle, Type II adversary $\mathscr{A}_2$ sends $(\mathrm{ID}_u, \mathrm{id}_i, m_i)$, where id_i is the block identifier, to the challenger $\mathscr{C}$. The challenger $\mathscr{C}$ responds with a pair of message and tag (m_i, t_i), where $t_i = \mathrm{TagGen}(\mathrm{sk}_u, \mathrm{ID}_u, \mathrm{id}_i, m_i)$.

(4) Finally, Type II adversary $\mathscr{A}_2$ outputs a forged pair of message and tag $(\mathrm{id}_i, m^*, t^*)$ for an identity ID_{tg}.

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

Definition 2. Type II adversary $\mathscr{A}_2$ $(t, q_h, q_k, q_{\mathrm{ppk}}, q_{\mathrm{sv}}, q_t, \varepsilon)$-breaks a certificateless public auditing scheme, if $\mathscr{A}_2$ 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_t TagGen queries, and wins the game with the probability ε. A certificateless public auditing scheme is secure against a tag-forgery by the malicious-but-passive-KGC attack, if there is no tag-forger which $(t, q_h, q_k, q_{\mathrm{ppk}}, q_{\mathrm{sv}}, q_t, \varepsilon)$-breaks the scheme with a nonnegligible probability ε.

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

(1) The challenger $\mathscr{C}$ runs algorithm Setup to generate a public parameter (PP) and the master key (MK) and sends PP to Type III adversary $\mathscr{A}_3$.

(2) Type III adversary $\mathscr{A}_3$ 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 $\mathscr{A}_3$ sends ID_u to the challenger $\mathscr{C}$, and then the public key pk_u is returned. For Partial-Private-Key-Extract oracle, Type III adversary $\mathscr{A}_3$ sends ID_u to the challenger $\mathscr{C}$, and then the partial private key psk_u is returned. For Secret-Value-Extract oracle, Type III adversary $\mathscr{A}_3$ sends ID_u to the challenger $\mathscr{C}$, and then the secret value $\mathrm{sv}_{\mathrm{ID}_u}$ is returned. For TagGen oracle, Type III adversary $\mathscr{A}_3$ sends $(\mathrm{ID}_u, \mathrm{id}_i, m_i)$, where id_i is the block identifier, to the challenger $\mathscr{C}$. The challenger $\mathscr{C}$ responds with a pair of message and tag (m_i, t_i), where $t_i = \mathrm{TagGen}(\mathrm{sk}_u, \mathrm{ID}_u, \mathrm{id}_i, m_i)$.

(3) The challenger $\mathscr{C}$ generates the challenge message $C = \mathrm{Challenge}(\mathrm{PP})$ and sends it to Type III adversary $\mathscr{A}_3$.

(4) Finally, Type III adversary $\mathscr{A}_3$ outputs a forged proof Prf^*.

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

Definition 3. Type III adversary $\mathscr{A}_3$ $(t, q_h, q_k, q_{\mathrm{ppk}}, q_{\mathrm{sv}}, q_t, \varepsilon)$-breaks a certificateless public auditing scheme, if $\mathscr{A}_3$ 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_t TagGen queries, and wins the game with the probability ε. A certificateless public auditing scheme is proof-unforgeable (sound), if there is no proof-forger which $(t, q_h, q_k, q_{\mathrm{ppk}}, q_{\mathrm{sv}}, q_t, \varepsilon)$-breaks the scheme with a nonnegligible probability ε.

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

Partial-Private-Key-Extract: for each identity $\mathrm{ID}_u \in \{0,1\}^*$, KGC picks the random number $r \leftarrow \mathbb{Z}_q^*$ and computes $R_1 = g^r, \widetilde{\sigma} = r + xH(\mathrm{ID}_u, R_1) \bmod q$, and $\widetilde{R_2} = g^{\alpha\widetilde{\sigma}}$. KGC sends $\mathrm{psk}_u = (R_1, \widetilde{R_2}, \widetilde{\sigma})$ 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 $w, k \leftarrow \mathbb{Z}_q^*$ and computes $\sigma = k + \widetilde{\sigma} = k + r + xH(\mathrm{ID}_u, R_1)$, $R_0 = g^k$, $R_2 = \widetilde{R_2} \cdot (g^\alpha)^k$, $R_3 = (R_1 \cdot R_0)^w$, and $X_2 = X_1^w$. The user sets $\mathrm{sk}_u = (\sigma, w)$ as a set of secret keys and $\mathrm{pk}_u = (R_0, R_1, R_2, R_3, X_2)$ as a set of public keys.

TagGen: let $F = \{m_{i0}, \ldots, m_{i(s-1)}\}_{1 \le i \le n}$ be a file. The user computes an authenticated tag t_i for each block m_i as

$$t_i = \left(g_1^{w\widetilde{H}_i} \cdot h_i \cdot \prod_{j=0}^{s-1} g_2^{m_{ij}\alpha^{j+2}} \right)^{\sigma} = \left(g_1^{w\widetilde{H}_i} \cdot h_i \cdot g_2^{f_{\overrightarrow{\beta}_i}(\alpha)} \right)^{\sigma}, \tag{3}$$

where id_i is block identifier of m_i, fname is the file name, $\widetilde{H}_i = \widetilde{H}(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i)$, $h_i = h(\mathrm{fname}, \mathrm{id}_i)$, $\overrightarrow{\beta}_i = (0, 0, m_{i0}, m_{i1}, \ldots, m_{i(s-1)})$, and $f_{\overrightarrow{\beta}_i}$ is a polynomial with a coefficient vector $\overrightarrow{\beta}_i$. Then the user uploads the file $F = \{m_i\}_{1 \le i \le n}$ with the corresponding authenticated tags $\{t_i\}_{1 \le i \le n}$ to the server. The validity of the pair of message and tag could be checked by the equation

$$e(t_i, g) = e\left(g_1^{\widetilde{H}_i}, R_3 \cdot X_2^{H(\mathrm{ID}_u, R_1)}\right) \cdot e\left(h_i \cdot g_2^{f_{\overrightarrow{\beta}_i}(\alpha)}, R_0 \cdot R_1 \cdot X_1^{H(\mathrm{ID}_u, R_1)}\right). \tag{4}$$

Challenge: to check the integrity of F, TPA randomly chooses random number $r_c \leftarrow \mathbb{Z}_q^*$ and d-elements subset D of $[1, n]$. Then, TPA sends the challenge messages (r_c, D) to the server.

Prove: when the server receives the challenge message (r_c, D), the server first computes $p_i = r_c^{\,i} \bmod q$ for all $i \in D$ and $y = f_{\vec{A}}(r_c) \bmod q$, where $\vec{A} = (0, 0, \sum_{i \in D} p_i m_{i0}, \ldots, \sum_{i \in K} p_i m_{i(s-1)})$.

The server computes the quotient polynomial $f_{\vec{\omega}}(\mathbf{x}) = (f_{\vec{A}}(\mathbf{x}) - f_{\vec{A}}(r_c))/(\mathbf{x} - r_c)$, where $\vec{\omega} = (\omega_0, \omega_1, \ldots, \omega_s)$ is the coefficient vector resulting in the quotient polynomial, and $\psi = \prod_{j=0}^{s} (g_2^{\alpha^j})^{w_j} = g_2^{f_{\vec{\omega}}(\alpha)}$. Finally, the server computes $\rho = \prod_{i \in D} h_i^{p_i}$ and $T = \prod_{i \in D} t_i^{p_i}$ and then sends the proof (T, ψ, y, ρ) to TPA.

Verify: upon receiving the proof (T, ψ, y, ρ), TPA checks the validity of the proof. TPA computes $\pi = \sum_{i \in D} \widetilde{H}(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i) p_i$ and checks the equation

$$e(T, g) = e(g_1^{\pi}, g^{w\sigma}) \cdot e(\rho \cdot g_2^{y}, g^{\sigma}) \cdot e\left(\psi, R_2 \cdot g^{\sigma \cdot (-r_c)}\right), \tag{5}$$

where $g^{w\sigma} = R_3 \cdot X_2^{H(\mathrm{ID}_u, R_1)}$ and $g^{\sigma} = R_0 \cdot R_1 \cdot X_1^{H(\mathrm{ID}_u, R_1)}$. If the equation holds, TPA outputs *Accept*; otherwise it outputs *Reject*.

The correctness of our scheme can be proved as follows:

$$\begin{aligned}
&e(g_1^{\pi}, g^{w\sigma}) \cdot e(\rho \cdot g_2^{y}, g^{\sigma}) \cdot e\left(\psi, R_2 \cdot g^{\sigma \cdot (-r_c)}\right) \\
&= e\left(\prod_{i \in D} g_1^{\widetilde{H}_i p_i}, g^{w\sigma}\right) \cdot e\left(\prod_{i \in D} h_i^{p_i}, g^{\sigma}\right) \cdot e(g_2^{y}, g^{\sigma}) \\
&\quad \cdot e\left(\psi, g^{\alpha\sigma} \cdot g^{\sigma \cdot (-r_c)}\right) \\
&= e\left(\prod_{i \in D} g_1^{w\widetilde{H}_i p_i} \cdot h_i^{p_i}, g^{\sigma}\right) \cdot e(g_2^{y}, g^{\sigma}) \\
&\quad \cdot e\left(g_2^{f_{\vec{\omega}}(\alpha)}, g^{\alpha\sigma} \cdot g^{\sigma \cdot (-r_c)}\right) \\
&= e\left(\prod_{i \in D} g_1^{w\widetilde{H}_i p_i} \cdot h_i^{p_i}, g^{\sigma}\right) \cdot e\left(g_2^{f_{\vec{A}}(r_c)}, g^{\sigma}\right) \\
&\quad \cdot e\left(g_2^{f_{\vec{\omega}}(\alpha) \cdot (\alpha - r_c)}, g^{\sigma}\right) \\
&= e\left(\prod_{i \in D} g_1^{w\widetilde{H}_i p_i} \cdot h_i^{p_i}, g^{\sigma}\right) \cdot e\left(g_2^{f_{\vec{A}}(r_c)}, g^{\sigma}\right) \\
&\quad \cdot e\left(g_2^{f_{\vec{A}}(\alpha) - f_{\vec{A}}(r_c)}, g^{\sigma}\right) \\
&= e\left(\prod_{i \in D} g_1^{w\widetilde{H}_i p_i} \cdot h_i^{p_i}, g^{\sigma}\right) \cdot e\left(g_2^{f_{\vec{A}}(\alpha)}, g^{\sigma}\right) \\
&= e\left(\prod_{i \in D} g_1^{w\widetilde{H}_i p_i} \cdot h_i^{p_i} \cdot g_2^{f_{\overrightarrow{\beta}_i}(\alpha) \cdot p_i}, g^{\sigma}\right) \\
&= e\left(\prod_{i \in D} \left(\left(g_1^{w \cdot \widetilde{H}_i} \cdot h_i \cdot g_2^{f_{\overrightarrow{\beta}_i}(\alpha)}\right)^{\sigma}\right)^{p_i}, g\right) \\
&= e\left(\prod_{i \in D} t_i^{p_i}, g\right) = e(T, g).
\end{aligned} \tag{6}$$

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* $\mathbb{G}$.

Proof of Theorem 4. Suppose Type I adversary $\mathcal{A}_1$ 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 ε. Then we can construct an algorithm $\mathcal{B}$ that solves the CDH problem on $\mathbb{G}$.

Given an instance $(g, g^a, g^b) \in \mathbb{G}$ of the CDH problem, the algorithm $\mathcal{B}$ simulates the challenger and interacts with the forger $\mathcal{A}_1$ in Game I.

(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 $\alpha, z_1, z_2 \leftarrow \mathbb{Z}_q^*$ and computes $\{g_2^{\alpha^j}\}_{0 \le j \le s+1}$, where $g_2 = g^{z_2}$. The algorithm $\mathcal{B}$ sets $X_1 = g^a$ and $g_1 = g^{z_1}$ and sends PP to the forger $\mathcal{A}_1$.

(3) *For* KeyGen *oracle*, at any time the forger $\mathcal{A}_1$ can query the KeyGen oracle for an identity ID_u. The algorithm $\mathcal{B}$ stores Tab1 $= (\mathrm{ID}_u, \epsilon_u, r_u, \widetilde{\sigma_u}, \sigma_u, w_u, k_u, \widetilde{R_{u,2}}, \mathrm{pk}_u)$, where Tab1 is initially empty. When the forger $\mathcal{A}_1$ queries ID_u, the algorithm $\mathcal{B}$ responds as follows:

(a) If $\mathrm{ID}_u \neq \mathrm{ID}_{tg}$, the algorithm $\mathcal{B}$ chooses random number $\epsilon_u, \widetilde{\sigma_u}, w_u, k_u \leftarrow \mathbb{Z}_q^*$ and calculates $\sigma_u = \widetilde{\sigma_u} + k_u$, $R_{u,0} = g^{k_u}$, $R_{u,1} = g^{\widetilde{\sigma_u}}/(g^a)^{\epsilon_u}$, $\widetilde{R_{u,2}} = g^{\alpha\widetilde{\sigma_u}}$, $R_{u,2} = g^{\alpha(\widetilde{\sigma_u}+k_u)}$, $R_{u,3} = (R_{u,1} \cdot R_{u,0})^{w_u}$, and $X_{u,2} = (g^a)^{w_u}$. The algorithm $\mathcal{B}$ sends $\mathrm{pk}_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})$ to the forger $\mathcal{A}_1$ as the set of public keys for the identity ID_u and updates Tab1.

(b) If $\mathrm{ID}_u = \mathrm{ID}_{tg}$, the algorithm $\mathcal{B}$ chooses random number $\epsilon_u, r_u, w_u, k_u \leftarrow \mathbb{Z}_q^*$ and calculates $R_{u,0} = g^{k_u}$, $R_{u,1} = g^{r_u}$, $\widetilde{R_{u,2}} = (R_{u,1} \cdot (g^a)^{\epsilon_u})^{\alpha}$, $R_{u,2} = (R_{u,1} \cdot (g^a)^{\epsilon_u})^{\alpha} \cdot g^{\alpha k_u}$, $R_{u,3} = (R_{u,1} \cdot R_{u,0})^{w_u}$, and $X_{u,2} = (g^a)^{w_u}$. The algorithm $\mathcal{B}$ sends $\mathrm{pk}_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})$ to the forger $\mathcal{A}_1$ 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 $\mathcal{A}_1$ can query the random oracle H for $(\mathrm{ID}_u, R_{u,1})$. 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 $H(\mathrm{ID}_u, R_{u,1}) = \epsilon_u$ in Tab1 to the forger $\mathcal{A}_1$.

For Hash Oracle $\widetilde{H}$. At any time the forger $\mathcal{A}_1$ can query the random oracle $\widetilde{H}$ for $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i)$. To respond to the hash queries, the algorithm $\mathcal{B}$ maintains a table Tab2 $= (\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i, \xi_{u,i})$, where Tab2 is initially empty. When the forger $\mathcal{A}_1$ queries $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i)$, the algorithm $\mathcal{B}$ responds as follows:

(a) If $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i) \notin$ Tab2, the algorithm $\mathcal{B}$ picks a random number $\xi_{u,i} \in \mathbb{Z}_q^*$ and sets the hash value $\widetilde{H_{u,i}} = \xi_{u,i}$. Then the algorithm $\mathcal{B}$ stores the tuple Tab2 $= (\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i, \xi_{u,i})$ and responds with $H(\mathrm{pk}_u, \mathrm{id}_i) = \xi_{u,i}$ to the forger $\mathcal{A}_1$.

(b) If $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i) \in$ Tab2, the algorithm $\mathcal{B}$ returns $\widetilde{H}(\mathrm{pk}_u, \mathrm{id}_i) = \xi_{u,i}$ in Tab2 to the forger $\mathcal{A}_1$.

For Hash Oracle h. At any time the forger $\mathcal{A}_1$ can query the random oracle h for $(\mathrm{fname}, \mathrm{id}_i)$. To respond to the hash queries, the algorithm $\mathcal{B}$ maintains a table Tab3 $= (\mathrm{fname}, \mathrm{id}_i, \delta_i, c_i, h_i)$, where Tab3 is initially empty. When the forger $\mathcal{A}_1$ queries $(\mathrm{fname}, \mathrm{id}_i)$, the algorithm $\mathcal{B}$ responds as follows:

(a) If $(\mathrm{fname}, \mathrm{id}_i) \notin$ Tab3, the algorithm $\mathcal{B}$ 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 $\mathcal{B}$ picks a random number $\delta_i \in \mathbb{Z}_q^*$ and computes $h_i = (g^b)^{(1-c_i)} \cdot g^{\delta_i}$. Then the algorithm $\mathcal{B}$ stores the tuple Tab3 $= (\mathrm{fname}, \mathrm{id}_i, \delta_i, c_i, h_i)$ and returns $h(\mathrm{fname}, \mathrm{id}_i) = h_i$ to the forger $\mathcal{A}_1$.

(b) If $(\mathrm{fname}, \mathrm{id}_i) \in$ Tab3, the algorithm $\mathcal{B}$ responds with $h(\mathrm{fname}, \mathrm{id}_i) = h_i$ in Tab3.

(5) The algorithm $\mathcal{B}$ 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 $\mathcal{A}_1$ can request a partial private key for ID_u to the oracle. When the forger $\mathcal{A}_1$ queries ID_u to the oracle, the algorithm $\mathcal{B}$ responds as follows:

(a) If $\mathrm{ID}_u \neq \mathrm{ID}_{tg}$, the algorithm $\mathcal{B}$ retrieves the tuple which contains the queried ID_u in Tab1 and returns $\mathrm{psk}_u = (R_{u,1}, \widetilde{R_{u,2}}, \sigma_u)$ to the forger $\mathcal{A}_1$.

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

For Public-Key-Replacement *Oracle*. At any time the forger $\mathcal{A}_1$ can replace a set of public keys for ID_u. Upon receiving a public key replacement query $(\mathrm{ID}_u, \mathrm{pk}_u')$, $\mathcal{A}_1$ 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 $\mathcal{A}_1$ is able to request a secret value for ID_u to the oracle. When the forger $\mathcal{A}_1$ queries ID_u to the oracle, $\mathcal{A}_1$ retrieves the tuple which contains ID_u in Tab1 and responds with (w_u, k_u).

For TagGen *Oracle*. At any time the forger $\mathcal{A}_1$ can query the oracle for the tuple $(\mathrm{ID}_u, \mathrm{id}_i, m_i)$. The algorithm $\mathcal{B}$ stores Tab4 $= (\mathrm{ID}_u, \mathrm{id}_i, m_i, \epsilon_u, r_u, w_u, k_u, \sigma_u, \xi_{u,i}, \delta_i, h_i, c_i, t_i)$, where Tab4 is initially empty. When the forger $\mathcal{A}_1$ queries ID_u, the algorithm $\mathcal{B}$ responds as follows:

(a) If $\mathrm{ID}_u \neq \mathrm{ID}_{tg}$, the algorithm $\mathcal{B}$ retrieves the tuple which is corresponding to the queried tuple $(\mathrm{ID}_u, \mathrm{id}_i)$ and computes the tag t_i for the

queried message m_i, where $t_i = (g_1^{w_u\xi_{u,i}} \cdot h_i \cdot g_2^{f_{\overrightarrow{\beta_i}}(\alpha)})^{\sigma_u}$. The algorithm $\mathcal{B}$ responds with t_i and updates Tab4.

(b) If $\mathrm{ID}_u = \mathrm{ID}_{tg}$, the algorithm $\mathcal{B}$ retrieves the tuple which is corresponding to the queried tuple $(\mathrm{ID}_u, \mathrm{id}_i)$. If $c_i = 1$, the algorithm $\mathcal{B}$ computes the tag t_i for the queried message m_i, where $t_i = (g_1^{w_u\xi_{u,i}} \cdot h_i \cdot g_2^{f_{\overrightarrow{\beta_i}}(\alpha)})^{k_u+r_u} \cdot ((g^a)^{w_u z_1 \xi_{u,i}+\delta_i+f_{\overrightarrow{\beta_i}}(\alpha)})^{\epsilon_u}$. The algorithm $\mathcal{B}$ responds with t_i and updates Tab4. Otherwise, the algorithm $\mathcal{B}$ reports failure.

(6) Finally, the forger $\mathcal{A}_1$ outputs the forged pair of message and tag $(\mathrm{ID}^*, \mathrm{id}_i, m_i^*, t_i^*)$ such that no TagGen query was issued for m_i^*. If $\mathrm{id}_i \in$ Tab3, $c_i = 0$, and $\mathrm{ID}^* = \mathrm{ID}_{tg}$, t_i^* satisfies the following equation:

$$e(t_i^*, g) = e\left(g_1^{\widetilde{H_{u,i}}}, R_{u,3} \cdot X_{u,2}^{H_u}\right) \cdot e\left(h_i \cdot g_2^{f_{\overrightarrow{\beta_i^*}}(\alpha)}, R_{u,0} \cdot R_{u,1} \cdot X_1^{H_u}\right). \tag{7}$$

By using the oracle-replay technique [37] on hash function H, the algorithm $\mathcal{B}$ can obtain another forged tag $t_i^{*\prime}$ for the same message m_i^*. The forged tag $t_i^{*\prime}$ also satisfies the following equation:

$$e(t_i^{*\prime}, g) = e\left(g_1^{\widetilde{H_{u,i}}}, R_{u,3} \cdot X_{u,2}^{H_u'}\right) \cdot e\left(h_i \cdot g_2^{f_{\overrightarrow{\beta_i^*}}(\alpha)}, R_{u,0} \cdot R_{u,1} \cdot X_1^{H_u'}\right). \tag{8}$$

With the above equations, the algorithm $\mathcal{B}$ could compute

$$e\left(\frac{t_i^*}{t_i^{*\prime}}, g\right) = e\left(g_1^{\widetilde{H_{u,i}}}, X_{u,2}^{H_u-H_u'}\right) \cdot e\left(h_i \cdot g_2^{f_{\overrightarrow{\beta_i^*}}(\alpha)}, X_1^{H_u-H_u'}\right). \tag{9}$$

From the right-hand-side of the above equation, the algorithm $\mathcal{B}$ could get

$$\begin{aligned} &e\left(g_1^{\widetilde{H_{u,i}}}, X_{u,2}^{H_u-H_u'}\right) \cdot e\left(h_i \cdot g_2^{f_{\overrightarrow{\beta_i^*}}(\alpha)}, X_1^{H_u-H_u'}\right) \\ &= e\left((g^{z_1})^{\widetilde{H_{u,i}}}, (g^a)^{w_u(H_u-H_u')}\right) \cdot e\left((g^b)^{\delta_i} \cdot g_2^{f_{\overrightarrow{\beta_i^*}}(\alpha)}, (g^a)^{H_u-H_u'}\right) = e\left((g^a)^{w_u z_1 \widetilde{H_{u,i}}(H_u-H_u')}, g\right) \\ &\cdot e\left((g^b)^{\delta_i} \cdot g^{z_2 f_{\overrightarrow{\beta_i^*}}(\alpha)}, (g^a)^{H_u-H_u'}\right) \\ &= e\left((g^a)^{w_u z_1 \widetilde{H_{u,i}}(H_u-H_u')}, g\right) \\ &\cdot e\left((g^{ab})^{\delta_i(H_u-H_u')}, g\right) \cdot e\left((g^a)^{(H_u-H_u')z_2 f_{\overrightarrow{\beta_i^*}}(\alpha)}, g\right) \\ &= e\left((g^a)^{(w_u z_1 \widetilde{H_{u,i}}+z_2 f_{\overrightarrow{\beta_i^*}}(\alpha))(H_u-H_u')}\right. \\ &\cdot \left.(g^{ab})^{\delta_i(H_u-H_u')}, g\right). \end{aligned} \tag{10}$$

Then the algorithm $\mathcal{B}$ outputs

$$g^{ab} = \left(\frac{(t_i^*/t_i^{*\prime})}{(g^a)^{\mathfrak{C}}}\right)^{1/\delta_i(H_u-H_u')}, \tag{11}$$

where $\mathfrak{C} = (w_u z_1 \widetilde{H_{u,i}} + z_2 f_{\overrightarrow{\beta_i^*}}(\alpha))(H_u - H_u')$.

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

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

(ii) E_2: the forger $\mathcal{A}_1$ outputs a valid pair of message and tag $(\mathrm{ID}^*, \mathrm{id}_i, m_i^*, t_i^*)$, which has not been queried before.

(iii) E_3: event E_2 occurs and $c_i = 0$ for the tuple containing id_i on Tab3.

In the case that the all events happen, the algorithm $\mathcal{B}$ can solve the CDH problem. Thus, the probability of success is $\Pr[E_1 \wedge E_3]$, and we can compute it as

$$\Pr[E_1 \wedge E_3] = \Pr[E_1] \cdot \Pr[E_2 \mid E_1] \cdot \Pr[E_3 \mid E_1 \wedge E_2]. \tag{12}$$

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 $\mathcal{B}$ does not abort in the Partial-Private-Key-Extract queries and the TagGen queries by the forger $\mathcal{A}_1$ is at least $1/e^2$.

Proof of Claim 1. We assume that the forger $\mathcal{A}_1$ does not query for the same index twice. At first, in the Partial-Private-Key-Extract oracle the algorithm $\mathcal{B}$ cannot respond to the forger $\mathcal{A}_1$'s query, if $\mathrm{ID}_u = \mathrm{ID}_{tg}$. This probability is equal to $1/q_k$, where q_k is the number of KeyGen 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 TagGen oracle, the algorithm $\mathcal{B}$ 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 id_i in Tab3. By the probability of c_i, the probability that the algorithm $\mathcal{B}$ does not abort is $1 - 1/(q_t + 1)$. Similarly, the probability that the algorithm $\mathcal{B}$ does not

abort in the TagGen oracle is at least $(1 - 1/(q_t + 1))^{q_t} \geq 1/e$. 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 $(1-1/q_k)^{q_k} \cdot (1-1/(q_t+1))^{q_t} \geq 1/e^2$.

Claim 2. The probability $\Pr[E_2 \mid E_1]$ that the forger $\mathcal{A}_1$ outputs a valid pair of message and tag under the condition that the event E_1 has occurred is at least ε.

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 the responses to the TagGen oracle 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 ϵ.

Claim 3. The probability $\Pr[E_3 \mid E_1 \wedge E_2]$ that the algorithm $\mathcal{B}$ does not abort after the forger $\mathcal{A}_1$ 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 $\mathcal{B}$ will abort only in the case that $c_i = 0$ for the tuple containing i corresponding to a forged pair of message and tag $(\mathrm{id}_i, m_i^*, t_i^*)$ in Tab1. Since the value c_i is randomly chosen with the probability $\Pr[c_i = 0] = 1/(q_t + 1)$, the probability $\Pr[E_3 \mid E_1 \wedge E_2]$ is at least $1/(q_t + 1)$.

With the above claims, the success probability of $\mathcal{B}$ is $\varepsilon/(e^2 \cdot (q_t + 1))$. Thus we can conclude that if the CDH assumption holds in $\mathbb{G}$, 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* $\mathbb{G}$.

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

Given an instance $(g, g^a, g^b) \in \mathbb{G}$ 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 α, x, $z_1, z_2 \leftarrow \mathbb{Z}_q^*$ and computes $\{g_2^{\alpha^j}\}_{0 \leq j \leq s+1}$, where $g_2 = g^{z_2}$. The algorithm $\mathcal{B}$ sets $X_1 = g^x$ 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_u. The algorithm $\mathcal{B}$ stores $\mathrm{Tab1} = (\mathrm{ID}_u, \epsilon_u, r_u, \widetilde{\sigma_u}, \sigma_u, w_u, k_u, \widetilde{R_{u,2}}, \mathrm{pk}_u)$, where Tab1 is initially empty. When the forger $\mathcal{A}_2$ queries ID_u, the algorithm $\mathcal{B}$ responds as follows:

(a) If $\mathrm{ID}_u \neq \mathrm{ID}_{tg}$, the algorithm $\mathcal{B}$ chooses random number $\epsilon_u, r_u, w_u, k_u \leftarrow \mathbb{Z}_q^*$ and calculates $R_{u,0} = g^{k_u}$, $R_{u,1} = g^{r_u}$, $\widetilde{\sigma_u} = r_u + x\epsilon_u$, $\sigma_u = k_u + r_u + \epsilon_u$, $\widetilde{R_{u,2}} = g^{\alpha\widetilde{\sigma_u}}$, $R_{u,2} = g^{\alpha\sigma_u}$, $R_{u,3} = (R_{u,1} \cdot R_{u,0})^{w_u}$, and $X_{u,2} = X_1^{w_u}$. The algorithm $\mathcal{B}$ sends $\mathrm{pk}_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})$ to the forger $\mathcal{A}_2$ as the set of public keys for the identity ID_u and updates Tab1.

(b) If $\mathrm{ID}_u = \mathrm{ID}_{tg}$, the algorithm $\mathcal{B}$ chooses random number $\epsilon_u, r_u, w_u \leftarrow \mathbb{Z}_q^*$ and calculates $\widetilde{\sigma_u} = r_u + x\epsilon_u$, $R_{u,0} = g^{k_u} = g^a$, $R_{u,1} = g^{r_u}$, $\widetilde{R_{u,2}} = g^{\alpha\widetilde{\sigma_u}}$, $R_{u,2} = \widetilde{R_{u,2}} \cdot (g^a)^\alpha$, $R_{u,3} = (R_{u,1} \cdot R_{u,0})^{w_u}$, and $X_{u,2} = X_1^{w_u}$. The algorithm $\mathcal{B}$ sends $\mathrm{pk}_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})$ 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 $(\mathrm{ID}_u, R_{u,1})$. 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 $H(\mathrm{ID}_u, R_{u,1}) = \epsilon_u$ in Tab1 to the forger $\mathcal{A}_2$.

For Hash Oracle $\widetilde{H}$. At any time the forger $\mathcal{A}_2$ can query the random oracle $\widetilde{H}$ for $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i)$. To respond to the hash queries, the algorithm $\mathcal{B}$ maintains a table $\mathrm{Tab2} = (\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i, \xi_{u,i})$, where Tab2 is initially empty. When the forger $\mathcal{A}_2$ queries $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i)$, the algorithm $\mathcal{B}$ responds as follows:

(a) If $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i) \notin \mathrm{Tab2}$, the algorithm $\mathcal{B}$ picks a random number $\xi_{u,i} \in \mathbb{Z}_q^*$ and sets the hash value $\widetilde{H_u} = \xi_{u,i}$. Then the algorithm $\mathcal{B}$ stores the tuple $\mathrm{Tab2} = (\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i, \xi_{u,i})$ and responds with $H(\mathrm{pk}_u, \mathrm{id}_i) = \xi_{u,i}$ to the forger $\mathcal{A}_2$.

(b) If $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i) \in \mathrm{Tab2}$, the algorithm $\mathcal{B}$ returns $\widetilde{H}(\mathrm{pk}_u, \mathrm{id}_i) = \xi_{u,i}$ 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 $(\mathrm{fname}, \mathrm{id}_i)$. To respond to the hash queries, the algorithm $\mathcal{B}$ maintains a table $\mathrm{Tab3} = (\mathrm{fname}, \mathrm{id}_i, \delta_i, c_i, h_i)$, where

Tab3 is initially empty. When the forger $\mathcal{A}_2$ queries $(\text{fname}, \text{id}_i)$, the algorithm $\mathcal{B}$ responds as follows:

(a) If $(\text{fname}, \text{id}_i) \notin$ Tab3, the algorithm $\mathcal{B}$ 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 $\mathcal{B}$ picks a random number $\delta_i \in \mathbb{Z}_q^*$ and computes $h_i = (g^b)^{(1-c_i)} \cdot g^{\delta_i}$. Then the algorithm $\mathcal{B}$ stores the tuple Tab3 $= (\text{fname}, \text{id}_i, \delta_i, c_i, h_i)$ and returns $H(\text{fname}, \text{id}_i) = h_i$ to the forger $\mathcal{A}_2$.

(b) If $(\text{fname}, \text{id}_i) \in$ Tab3, the algorithm $\mathcal{B}$ responds with $h(\text{fname}, \text{id}_i) = h_i$ in Tab3.

(5) The algorithm $\mathcal{B}$ 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 $\mathcal{A}_2$ can request a partial private key for ID_u to the oracle. When the forger $\mathcal{A}_2$ queries ID_u to the oracle, $\mathcal{B}$ retrieves the tuple which contains ID_u in Tab1 and responds with $(R_{u,1}, \widetilde{R_{u,2}}, \widetilde{\sigma_u})$.

For Secret-Value-Extract *Oracle.* At any time the forger $\mathcal{A}_2$ is able to request a secret value for ID_u to the oracle. When the forger $\mathcal{A}_1$ queries ID_u to the oracle, $\mathcal{B}$ responds as follows:

(a) If $\text{ID}_u \neq \text{ID}_{tg}$, the algorithm $\mathcal{B}$ retrieves the tuple which contains the queried ID_u in Tab1 and returns (w_u, k_u) to the forger $\mathcal{A}_2$.

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

For TagGen *Oracle.* At any time the forger $\mathcal{A}_2$ can query the oracle for the tuple $(\text{ID}_u, \text{id}_i, m_i)$. The algorithm $\mathcal{B}$ stores Tab4 $= (\text{ID}_u, \text{id}_i, m_i, \epsilon_u, r_u, w_u, k_u, \sigma_u, \xi_{u,i}, \delta_i, h_i, c_i, t_i)$, where Tab4 is initially empty. When the forger $\mathcal{A}_2$ queries ID_u, the algorithm $\mathcal{B}$ responds as follows:

(a) If $\text{ID}_u \neq \text{ID}_{tg}$, the algorithm $\mathcal{B}$ retrieves the tuple which is corresponding to the queried tuple $(\text{ID}_u, \text{id}_i)$ and computes the tag t_i for the queried message m_i, where $t_i = (g_1^{w_u \xi_{u,i}} \cdot h_i \cdot g_2^{f_{\vec{\beta}_i}(\alpha)})^{\sigma_u}$. The algorithm $\mathcal{B}$ responds with t_i and updates Tab4.

(b) If $\text{ID}_u = \text{ID}_{tg}$, the algorithm $\mathcal{B}$ retrieves the tuple which is corresponding to the queried tuple $(\text{ID}_u, \text{id}_i)$. If $c_i = 1$, the algorithm $\mathcal{B}$ computes the tag t_i for the queried message m_i, where $t_i = (g_1^{w_u \xi_{u,i}} \cdot h_i \cdot g_2^{f_{\vec{\beta}_i}(\alpha)})^{\widetilde{\sigma_u}} \cdot (g^a)^{w_u z_2 \xi_{u,i} + \delta_i + f_{\vec{\beta}_i}(\alpha)}$. The algorithm $\mathcal{B}$ responds with t_i and updates Tab4. Otherwise, the algorithm $\mathcal{B}$ reports failure.

(6) Finally, the forger $\mathcal{A}_2$ outputs the forged pair of message and tag $(\text{ID}^*, \text{id}_i, m_i^*, t_i^*)$ such that no TagGen query was issued for m_i^*. If $\text{id}_i \in$ Tab3, $c_i = 0$, and $\text{ID}^* = \text{ID}_{tg}$, t_i^* satisfies the equation

$$e(t_i^*, g) = e\left(g_1^{\widetilde{H_{u,i}}}, R_{u,3} \cdot X_{u,2}^{H_u}\right) \cdot e\left(h_i \cdot g_2^{f_{\vec{\beta}_i^*}(\alpha)}, R_{u,0} \cdot R_{u,1} \cdot X_1^{H_u}\right). \tag{13}$$

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

$$\begin{aligned} &e\left(g_1^{\widetilde{H_{u,i}}}, R_{u,3} \cdot X_{u,2}^{H_u}\right) \cdot e\left(h_i \cdot g_2^{f_{\vec{\beta}_i^*}(\alpha)}, R_{u,0} \cdot R_{u,1} \cdot X_1^{H_u}\right) = e\left(g^{z_1 \widetilde{H_{u,i}}}, g^{(a+r_u)w_u} \cdot g^{x w_u H_i}\right) \cdot e\left(g^{b+\delta_i} \cdot g^{z_2 f_{\vec{\beta}_i^*}(\alpha)}, g^a \cdot g^{r_u + x H_u}\right) \\ &= e\left(g^{w_u z_1 \widetilde{H_{u,i}}}, g^a \cdot g^{r_u + x H_u}\right) \cdot e\left(g^b \cdot g^{\delta_i + z_2 f_{\vec{\beta}_i^*}(\alpha)}, g^a \cdot g^{r_u + x H_u}\right) \\ &= e\left((g^a)^{w_u z_1 \widetilde{H_{u,i}}} \cdot g^{(r_u + x H_u)(w_u z_1 \widetilde{H_{u,i}})}, g\right) \cdot e\left(g^{ab} \cdot (g^a)^{\delta_i + z_2 f_{\vec{\beta}_i^*}(\alpha)} \cdot (g^b)^{r_u + x H_u} \cdot g^{(\delta_i + z_2 f_{\vec{\beta}_i^*}(\alpha))(r_u + x H_u)}, g\right) \\ &= e\left(g^{ab} \cdot (g^a)^{w_u z_1 \widetilde{H_i} + \delta_i + z_2 f_{\vec{\beta}_i^*}(\alpha)} \cdot (g^b)^{r_u + x H_u} \cdot g^{(w_u z_1 \widetilde{H_{u,i}} + \delta_i + z_2 f_{\vec{\beta}_i^*}(\alpha))(r_u + x H_u)}, g\right). \end{aligned} \tag{14}$$

Then the algorithm $\mathcal{B}$ outputs

$$g^{ab} = \frac{t_i^*}{(g^a)^{\theta_1} \cdot (g^b)^{\theta_2} \cdot g^{\theta_3}}, \tag{15}$$

where $\theta_1 = w_u z_1 \widetilde{H_{u,i}} + \delta_i + z_2 f_{\overleftarrow{\beta_i^*}}(\alpha)$, $\theta_2 = r_u + x H_u$, and $\theta_3 = r_u + x H_u (w_u z_1 \widetilde{H_{u,i}}) + (\delta_i + z_2 f_{\overleftarrow{\beta_i^*}}(\alpha))(r_u + x H_u)$.

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

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

(ii) E_5: the forger $\mathcal{A}_2$ outputs a valid pair of message and tag $(\text{ID}^*, \text{id}_i, m_i^*, t_i^*)$, which has not been queried before.

(iii) E_6: event E_5 occurs and $c_i = 0$ for the tuple containing id_i on Tab3.

In the case that the all events happen, the algorithm $\mathscr{B}$ can solve the CDH problem. Thus, the probability of success is $\Pr[E_4 \wedge E_6]$, and we can compute it as

$$\Pr\left[E_4 \wedge E_6\right] = \Pr\left[E_4\right] \cdot \Pr\left[E_5 \mid E_4\right] \cdot \Pr\left[E_6 \mid E_4 \wedge E_5\right]. \tag{16}$$

We analyze the lower bounds of $\Pr[E_4]$, $\Pr[E_5 \mid E_4]$, and $\Pr[E_6 \mid E_4 \wedge E_5]$ in the following claims.

Claim 4. The probability $\Pr[E_4]$ that the algorithm $\mathscr{B}$ does not abort in the Secret-Value-Extract queries and the TagGen queries by the forger $\mathscr{A}_2$ is at least $1/e^2$.

Claim 5. The probability $\Pr[E_5 \mid E_4]$ that the forger $\mathscr{A}_2$ outputs a valid pair of message and tag under the condition that the event E_4 has occurred is at least ε.

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

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

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* $\mathbb{G}$.

Proof of Theorem 6. Suppose Type III adversary $\mathscr{A}_3$ is the proof-forger that could break the scheme by generating a fake proof for an arbitrary challenge message with a probability ε. We does 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 $\mathscr{A}_1$ and Type II adversary $\mathscr{A}_2$, it can be easily applied to Type III adversary $\mathscr{A}_3$ by eliminating the oracles related to the key extraction in the proof of Theorems 4 and 5). Using the proof-forger, we can construct an algorithm $\mathscr{B}$ that solves the DCDH problem on $\mathbb{G}$.

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

(1) The forger $\mathscr{A}_3$ outputs an identity ID_{tg} where it wishes to be forged.

(2) The algorithm $\mathscr{B}$ chooses random numbers $\alpha, x, z_1, z_2 \leftarrow \mathbb{Z}_q^*$ and computes $\{g_2^{\alpha^j}\}_{0 \le j \le s+1}$, where $g_2 = g^{z_2}$. The algorithm $\mathscr{B}$ sets $X_1 = g^x$ and $g_1 = g^{z_1}$ and sends PP to the forger $\mathscr{A}_3$.

(3) For KeyGen oracle, at any time the forger $\mathscr{A}_3$ can query the KeyGen oracle for an identity ID_u. The algorithm $\mathscr{B}$ stores Tab1 $= (\mathrm{ID}_u, \epsilon_u, r_u, \widetilde{\sigma_u}, \sigma_u, w_u, k_u, \gamma_u, \widetilde{R_{u,2}}, \mathrm{pk}_u)$, where Tab1 is initially empty. When the forger $\mathscr{A}_3$ queries ID_u, the algorithm $\mathscr{B}$ responds as follows:

(a) If $\mathrm{ID}_u \neq \mathrm{ID}_{tg}$, the algorithm $\mathscr{B}$ chooses random number $\epsilon_u, r_u, w_u, k_u \leftarrow \mathbb{Z}_q^*$ and calculates $R_{u,0} = g^{k_u}$, $R_{u,1} = g^{r_u}$, $\widetilde{\sigma_u} = r_u + x\epsilon_u$, $\sigma_u = k_u + r_u + \epsilon_u$, $\widetilde{R_{u,2}} = g^{\alpha\widetilde{\sigma_u}}$, $R_{u,2} = g^{\alpha\sigma_u}$, $R_{u,3} = (R_{u,1} \cdot R_{u,0})^{w_u}$, and $X_{u,2} = X_1^{w_u}$. The algorithm $\mathscr{B}$ sends $\mathrm{pk}_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})$ to the forger $\mathscr{A}_2$ as the set of public keys for the identity ID_u and updates Tab1.

(b) If $\mathrm{ID}_u = \mathrm{ID}_{tg}$, the algorithm $\mathscr{B}$ chooses random number $\epsilon_u, \gamma_u, r_u \leftarrow \mathbb{Z}_q^*$ and calculates $\widetilde{\sigma_u} = r_u + x\epsilon_u$, $R_{u,0} = g^{k_u} = g^a / g^{\widetilde{\sigma_u}}$, $R_{u,1} = g^{r_u}$, $\widetilde{R_{u,2}} = g^{\alpha\widetilde{\sigma_u}}, R_{u,2} = (g^a)^\alpha$, $X_{u,2} = g^{\gamma_u}$, and $R_{u,3} = g^b/(g^{\gamma_u\epsilon_u})$. The algorithm $\mathscr{B}$ sends $\mathrm{pk}_u = (R_{u,0}, R_{u,1}, R_{u,2}, R_{u,3}, X_{u,2})$ to the forger $\mathscr{A}_3$ as the set of public keys for the identity ID_u and updates Tab1.

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

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

For Hash Oracle $\widetilde{H}$. At any time the forger $\mathscr{A}_3$ can query the random oracle $\widetilde{H}$ for $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i)$. To respond to the hash queries, the algorithm $\mathscr{B}$ maintains a table Tab2 $= (\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i, \xi_{u,i})$, where Tab2 is initially empty. When the forger $\mathscr{A}_3$ queries $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i)$, the algorithm $\mathscr{B}$ responds as follows:

(a) If $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i) \notin$ Tab2, the algorithm $\mathscr{B}$ picks a random number $\xi_{u,i} \in \mathbb{Z}_q^*$ and sets the hash value $\widetilde{H_{u,i}} = \xi_{u,i}$. Then the algorithm $\mathscr{B}$ stores the tuple Tab2 $= (\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i, \xi_{u,i})$ and responds with $H(\mathrm{pk}_u, \mathrm{id}_i) = \xi_{u,i}$ to the forger $\mathscr{A}_3$.

(b) If $(\mathrm{ID}_u, \mathrm{pk}_u, \mathrm{id}_i) \in$ Tab2, the algorithm $\mathscr{B}$ returns $\widetilde{H}(\mathrm{pk}_u, \mathrm{id}_i) = \xi_{u,i}$ in Tab2 to the forger $\mathscr{A}_3$.

For Hash Oracle h. At any time the forger $\mathscr{A}_3$ can query the random oracle h for $(\mathrm{fname}, \mathrm{id}_i)$. To respond to the hash queries, the algorithm $\mathscr{B}$ maintains a table Tab3 $= (\mathrm{fname}, \mathrm{id}_i, \delta_i, c_i, h_i)$, where Tab3 is initially empty. When the forger $\mathscr{A}_2$ queries $(\mathrm{fname}, \mathrm{id}_i)$, the algorithm $\mathscr{B}$ responds as follows:

(a) If $(\mathrm{fname}, id_i) \notin$ Tab3, the algorithm $\mathscr{B}$ picks a random number $\delta_i \in \mathbb{Z}_q^*$ and computes $h_i = g^{\delta_i}$. Then the algorithm $\mathscr{B}$ stores the tuple Tab3 $= (\mathrm{fname}, \mathrm{id}_i, \delta_i, h_i)$ and returns $h(\mathrm{fname}, \mathrm{id}_i) = h_i$ to the forger $\mathscr{A}_3$.

(b) If $(\text{fname}, \text{id}_i) \in$ Tab3, the algorithm $\mathscr{B}$ responds with $h(\text{fname}, \text{id}_i) = h_i$ in Tab3.

(5) The algorithm $\mathscr{B}$ 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 $\mathscr{A}_3$ can request a partial private key for ID_u to the oracle. When the forger $\mathscr{A}_3$ queries ID_u to the oracle, $\mathscr{B}$ responds as follows:

(a) If $\text{ID}_u \neq \text{ID}_{tg}$, the algorithm $\mathscr{B}$ retrieves the tuple which contains the queried ID_u in Tab1 and returns $(R_{u,1}, \widetilde{R_{u,2}}, \widetilde{\sigma_u})$ to the forger $\mathscr{A}_3$.

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

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

(a) If $\text{ID}_u \neq \text{ID}_{tg}$, the algorithm $\mathscr{B}$ retrieves the tuple which contains the queried ID_u in Tab1 and returns (w_u, k_u) to the forger $\mathscr{A}_3$.

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

For TagGen *Oracle.* At any time the forger $\mathscr{A}_3$ can query the oracle for the tuple $(\text{ID}_u, \text{id}_i, m_i)$. The algorithm $\mathscr{B}$ stores Tab4 $= (\text{ID}_u, \text{id}_i, m_i, \epsilon_u, r_u, w_u, k_u, \gamma_u, \sigma_u, \xi_{u,i}, \delta_i, h_i, t_i)$, where Tab4 is initially empty. When the forger $\mathscr{A}_3$ queries ID_u, the algorithm $\mathscr{B}$ responds as follows:

(a) If $\text{ID}_u \neq \text{ID}_{tg}$, the algorithm $\mathscr{B}$ retrieves the tuple which is corresponding to the queried tuple $(\text{ID}_u, \text{id}_i)$ and computes the tag t_i for the queried message m_i, where $t_i = (g_1^{w_u \xi_{u,i}} \cdot h_i \cdot g_2^{f_{\vec{\beta}_i}(\alpha)})^{\sigma_u}$. The algorithm $\mathscr{B}$ responds with t_i and updates Tab4.

(b) If $\text{ID}_u = \text{ID}_{tg}$, the algorithm $\mathscr{B}$ retrieves the tuple which is corresponding to the queried tuple $(\text{ID}_u, \text{id}_i)$. The algorithm $\mathscr{B}$ computes the tag t_i for the queried message m_i, where $t_i = ((g^b)^{\xi_{u,i}}) \cdot (g^a)^{\delta_i + z_2 f_{\vec{\beta}_i}(\alpha)}$.

(6) The algorithm $\mathscr{B}$ chooses random number $r_c \leftarrow Z_q^*$ and a random d-elements subset $D \subset [1, n]$. In this process, $\mathscr{B}$ should choose a random subset which contains at least an index id_i, where id_i 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 $\mathscr{B}$ sends (r_c, D) as the challenge message.

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

$$e(T, g) = e\left(g_1^{\pi}, g^b\right) \cdot e\left(\rho^* \cdot g_2^{y^*}, g^a\right) \cdot e\left(\psi^*, R_{u,2} \cdot g^{a \cdot (-r_c)}\right). \tag{17}$$

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

$$\begin{aligned} &e\left(g_1^{\pi}, g^b\right) \cdot e\left(\rho^* \cdot g_2^{y^*}, g^a\right) \cdot e\left(\psi^*, R_{u,2} \cdot g^{a \cdot (-r_c)}\right) \\ &= e\left(g^{z_1 \pi}, g^b\right) \cdot e\left(\rho^* \cdot g^{z_2 y^*}, g^a\right) \cdot e\left(\psi^*, g^{a(\alpha - r_c)}\right) \\ &= e\left(g^{z_1 \pi b}, g\right) \cdot e\left(\rho^* \cdot g^{z_2 y^*}, g^a\right) \cdot e\left(\psi^{*(\alpha - r_c)}, g^a\right) \\ &= e\left(g^{z_1 \pi b}, g\right) \cdot e\left(\rho^* \cdot g^{z_2 y^*} \cdot \psi^{*(\alpha - r_c)}, g^a\right). \end{aligned} \tag{18}$$

To satisfy (18), the value $(\rho^* \cdot g^{z_2 y^*} \cdot \psi^{*(\alpha - r_c)})$ should contain $(g^{a^{-1} b})^{\pi'}$, where $\pi' = \sum_{i \in D'} \xi_{u,i} p_i$. Thus, we can compute $g^{a^{-1} b}$ as

$$g^{a^{-1} b} = \left(\frac{\left(\rho^* \cdot g^{z_2 y^*} \cdot \psi^{*(\alpha - r_c)}\right)}{g^{\mathfrak{C}}} \right)^{1/\pi'}, \tag{19}$$

where $\mathfrak{C} = \sum_{i \in D} \delta_i p_i + z_2 f_{\vec{A}}(\alpha)$.

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

(i) E_7: the algorithm $\mathscr{B}$ does not abort in the Partial-Private-Key-Extract queries and Secret-Value-Extract queries.

(ii) E_8: the forger $\mathscr{A}_3$ outputs a valid proof (T, ρ^*, y^*, ψ^*), which has not been queried before.

In the case that the both events happen, the algorithm $\mathscr{B}$ can solve the DCDH problem. Thus, the probability of success is $\Pr[E_7 \wedge E_8] = \Pr[E_7] \cdot \Pr[E_8]$. We analyze the lower bounds of $\Pr[E_7]$ in the following claim.

Claim 7. The probability $\Pr[E_7]$ that the algorithm $\mathscr{B}$ does not abort in the Partial-Private-Key-Extract queries and the Secret-Value-Extract queries by the forger $\mathscr{A}_1$ is at least $1/e^2$.

Proof of Claim 7. We assume that the forger $\mathscr{A}_3$ does not query for the same index twice. In the Partial-Private-Key-Extract oracle, the algorithm $\mathscr{B}$ aborts only if $\text{ID}_u = \text{ID}_{tg}$. 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 $\mathscr{B}$ 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 $\mathscr{B}$ does not abort is at least $(1 - 1/q_k)^{q_k} \geq 1/e$. In the Secret-Value-Extract oracle, the algorithm $\mathscr{B}$ only aborts if $\text{ID}_u = \text{ID}_{tg}$. The probability that the algorithm $\mathscr{B}$ does not abort is $1 - 1/q_k$. Similarly, the probability that the algorithm $\mathscr{B}$ does not abort

Table 1: Security levels of certificateless public auditing schemes. Let $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ be a Type I adversary, Type II adversary, and Type III adversary as defined in Section 3, respectively. Type I adversary and Type II adversary are classed as the normal, strong, and super adversaries (ordered by the capability of the adversaries).

	Security against $\mathcal{A}_1$	Security against $\mathcal{A}_2$	Security against $\mathcal{A}_3$
[28]	Strong $\mathcal{A}_1$	Super $\mathcal{A}_2$	Secure
[29]	Super $\mathcal{A}_1$	Super $\mathcal{A}_2$	No formal proof provided
Ours	Super $\mathcal{A}_1$	Super $\mathcal{A}_2$	Secure

in the Secret-Value-Extract oracle is at least $(1-1/q_k)^{q_k} \geq 1/e$. Thus, the probability $\Pr[E_7]$ that the algorithm $\mathcal{B}$ does not abort in the forger $\mathcal{A}_3$'s Partial-Private-Key-Extract queries and Secret-Value-Extract queries is at least $(1 - 1/q_k)^{2q_k} \geq 1/e^2$.

With the above claim, the success probability of $\mathcal{B}$ is ε/e^2, since the probability $\Pr[E_8] = \varepsilon$. Thus we can conclude that if the DCDH assumption holds in $\mathbb{G}$, 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* $\mathbb{G}$.

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 computation 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 *Pair* be the bilinear pairing computation and $\text{Mul}_{\mathbb{G}}$ be the multiplication in group $\mathbb{G}$. Let $\text{Hash}_{\mathbb{G}}$ and $\text{Exp}_{\mathbb{G}}$ be the hash-to-point computation and the exponentiation in group $\mathbb{G}$, 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: let *Pair* be the bilinear pairing computation and $\text{Mul}_{\mathbb{G}}$ be the multiplication in group $\mathbb{G}$. Let $\text{Hash}_{\mathbb{G}}$ and $\text{Exp}_{\mathbb{G}}$ be the hash-to-point computation and the exponentiation in the group $\mathbb{G}$, respectively.

Computations	*Pair*	$\text{Mul}_{\mathbb{G}}$	$\text{Hash}_{\mathbb{G}}$	$\text{Exp}_{\mathbb{G}}$
Times (in milliseconds)	3.16	1.17	<1	0.62

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

	Computational cost of TPA	$d = 300$	$d = 460$
[28]	$3 \cdot Pair + (2d-1) \cdot \text{Mul}_{\mathbb{G}} + (d+1) \cdot \text{Hash}_{\mathbb{G}} + (2d+1) \cdot \text{Exp}_{\mathbb{G}} \approx (\mathbf{4.58} \times \mathbf{d} + \mathbf{9.93})$ ms	1383.93 ms	2116.73 ms
[29]	$2 \cdot Pair + (d+2) \cdot \text{Mul}_{\mathbb{G}} + (d+1) \cdot \text{Hash}_{\mathbb{G}} + (d+3) \cdot \text{Exp}_{\mathbb{G}} \approx (\mathbf{2.79} \times \mathbf{d} + \mathbf{11.52})$ ms	848.52 ms	1294.92 ms
Ours	$4 \cdot Pair + 7 \cdot \text{Mul}_{\mathbb{G}} + 6 \cdot \text{Exp}_{\mathbb{G}} \approx \mathbf{23.93}$ ms	23.93 ms	23.93 ms

In [2], Ateniese 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
$\mathbb{G}, \mathbb{G}_T$: Multiplicative cyclic groups of prime order q

e: A bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$
g, g_1, g_2: Generators of group $\mathbb{G}$
ID_u: A user identity
psk_u: A set of partial private keys for the identity ID_u
sk_u: A set of secret keys for the identity ID_u
pk_u: A set of public keys for the identity ID_u
F: A file $F = \{m_1, \ldots, m_n\}$
m_i: An ith block of the file F
id_i: A block identifier of the block m_i
fname: A file name
$f_{\overrightarrow{\beta_i}}$: A polynomial with a coefficient vector $\overrightarrow{\beta_i}$
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_c: 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

[1] P. Mell and T. Grance, The NIST Definition of Cloud Computing, 2011, http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[2] G. Ateniese, R. Burns, R. Curtmola et al., "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 598–610, November 2007.

[3] A. Juels and B. S. Kaliski Jr., "PORs: proofs of retrievability for large files," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 584–597, Alexandria, Va, USA, November 2007.

[4] H. Shacham and B. Waters, "Compact proofs of retrievability," in *International Conference on the Theory and Application of Cryptology and Information Security: ASIACRYPT 2008: Advances in Cryptology—ASIACRYPT*, vol. 5350 of *Lecture Notes in Computer Science*, pp. 90–107, Springer, Berlin, Germany, 2008.

[5] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pp. 213–222, Chicago, Ill, USA, November 2009.

[6] B. Wang, B. Li, and H. Li, "Oruta: privacy-preserving public auditing for shared data in the cloud," in *Proceedings of the IEEE 5th International Conference on Cloud Computing (CLOUD '12)*, pp. 295–302, June 2012.

[7] 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.

[8] B. Wang, B. Li, and H. Li, "Public auditing for shared data with efficient user revocation in the cloud," in *Proceedings of the 32nd IEEE Conference on Computer Communications (INFOCOM '13)*, pp. 2904–2912, April 2013.

[9] 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.

[10] T. Jiang, X. Chen, and J. Ma, "Public integrity auditing for shared dynamic cloud data with group user revocation," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2363–2373, 2016.

[11] C. Ellison and B. Schneier, "Ten risks of PKI: what you're not being told about public key infrastructure," *Computer Security Journal*, vol. 16, no. 1, pp. 1–7, 2000.

[12] P. Gutmann, "PKI: it's not dead, just resting," *Computer*, vol. 35, no. 8, pp. 41–49, 2002.

[13] A. Shamir, "Identity-based Cryptosystems and Signature Schemes," in *Proceedings of the CRYPTO: Advances in Cryptology*, pp. 47–53, 1984.

[14] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *Annual International Cryptology Conference: CRYPTO 2001: Advances in Cryptology—CRYPTO 2001*, vol. 2139 of *Lecture Notes in Computer Science*, pp. 213–229, Springer, Berlin, Germany, 2001.

[15] D. Boneh, R. Canetti, and S. Halevi, "Chosen-Ciphertext Security from Identity-Based Encryption," in *EUROCRYPT 2004: Advances in Cryptology*, vol. 3027 of *Lecture Notes in Computer Science*, pp. 207–222, 2004.

[16] D. Boneh and X. Boyen, "Efficient selective-id secure identity based encryption without random oracles," in *International Conference on the Theory and Applications of Cryptographic Techniques EUROCRYPT 2004: Advances in Cryptology—EUROCRYPT 2004*, vol. 3027 of *Lecture Notes in Computer Science*, pp. 223–238, Springer, Berlin, Germany, 2004.

[17] B. Waters, "Efficient identity based encryption without random oracles," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques: UROCRYPT 2005: Advances in Cryptology—EUROCRYPT 2005*, vol. 3494 of *Lecture Notes in Computer Science*, pp. 114–127, Springer, Berlin, Germany, 2005.

[18] H. Wang, J. Domingo-Ferrer, Q. Wu, and B. Qin, "Identity-based remote data possession checking in public clouds," *IET Information Security*, vol. 8, no. 2, pp. 114–121, 2014.

[19] H. Wang, "Identity-based distributed provable data possession in multicloud storage," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 328–340, 2015.

[20] Y. Yu, Y. Zhang, Y. Mu, W. Susilo, and H. Liu, "Provably secure identity based provable data possession," in *Provable Security: 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24–26, 2015, Proceedings*, vol. 9451 of *Lecture Notes in Computer Science*, pp. 310–325, Springer, Berlin, Germany, 2015.

[21] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Advances in Cryptology—ASIACRYPT 2003*, C.-S. Laih, Ed., vol. 2894 of *Lecture Notes in Computer Science*, pp. 452–473, Springer, Berlin, Germany, 2003.

[22] B. Libert and J.-J. Quisquater, "On constructing certificateless cryptosystems from the identity based encryption," in *International Workshop on Public Key Cryptography PKC 2006: Public Key Cryptography—PKC 2006*, vol. 3958 of *Lecture Notes in Computer Science*, pp. 474–490, Springer, Berlin, Germany, 2006.

[23] A. Dent, B. W. Libert, and K. Paterson, "Certificateless encryption schemes storngly secure in the standard model," in *Proceedings of the International Conference on the Theory and Practice of Public-Key Cryptography (PKC '08)*, vol. 4939 of *Lecture Notes in Computer Science*, pp. 344–359, 2008.

[24] X. Huang, Y. Mu, W. Susilo, D. S. Wong, and W. Wu, "Certificateless signature revisited," in *Information Security and Privacy*, J. Pieprzyk, H. Ghodosi, and E. Dawson, Eds., vol. 4586 of *Lecture Notes in Computer Science*, pp. 308–322, Springer, Berlin, Germany, 2007.

[25] D. He, J. Chen, and R. Zhang, "An efficient and provably-secure certificateless signature scheme without bilinear pairings," *International Journal of Communication Systems*, vol. 25, no. 11, pp. 1432–1442, 2012.

[26] D. He, Y. Chen, and J. Chen, "An efficient certificateless proxy signature scheme without pairing," *Mathematical and Computer Modelling*, vol. 57, no. 9-10, pp. 2510–2518, 2013.

[27] D. He, B. Huang, and J. Chen, "New certificateless short signature scheme," *IET Information Security*, vol. 7, no. 2, pp. 113–117, 2013.

[28] B. Wang, B. Li, H. Li, and F. Li, "Certificateless public auditing for data integrity in the cloud," in *Proceedings of the 1st IEEE International Conference on Communications and Network Security (CNS '13)*, pp. 136–144, National Harbo, Md, USA, October 2013.

[29] D. He, S. Zeadally, and L. Wu, "Certificateless public auditing scheme for cloud-assisted wireless body area networks," *IEEE Systems Journal*, no. 99, pp. 1–10, 2015.

[30] J. Liu, Z. Zhang, X. Chen, and K. S. Kwak, "Certificateless remote anonymous authentication schemes for wirelessbody area networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 332–342, 2014.

[31] Z. Zhao, "An efficient anonymous authentication scheme for wireless body area networks using elliptic curve cryptosystem," *Journal of Medical Systems*, vol. 38, no. 2, article 13, 2014.

[32] X. Hu, "Cost-effective scalable and anonymous certificateless remote authentication protocol," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 2, pp. 2327–2339, 2014.

[33] S. Chatterjee, A. K. Das, and J. K. Sing, "A novel and efficient user access control scheme for wireless body area sensor networks," *Journal of King Saud University—Computer and Information Sciences*, vol. 26, no. 2, pp. 181–201, 2014.

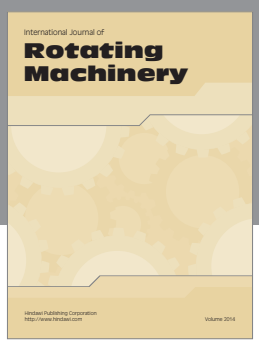
[34] A. K. Das, S. Chatterjee, and J. K. Sing, "A new biometric-based remote user authentication scheme in hierarchical wireless body area sensor networks," *Ad-Hoc and Sensor Wireless Networks*, vol. 28, no. 3-4, pp. 221–256, 2015.

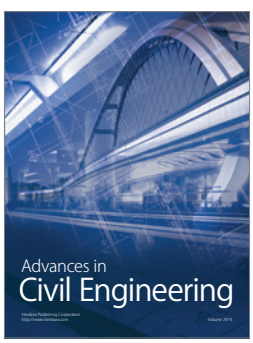
[35] M. H. Ibrahim, S. Kumari, A. K. Das, M. Wazid, and V. Odelu, "Secure anonymous mutual authentication for star two-tier wireless body area networks," *Computer Methods and Programs in Biomedicine*, vol. 135, pp. 37–50, 2016.

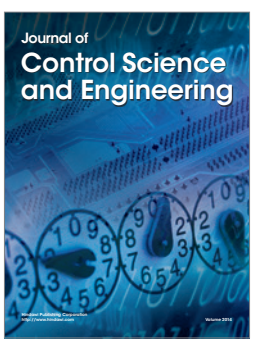
[36] F. Bao, R. H. Deng, and H. Zhu, "Variations of diffie-hellman problem," in *Proceedings of the 5th International Conference of Information and Communications Security (ICICS '03)*, pp. 301–312, 2003.

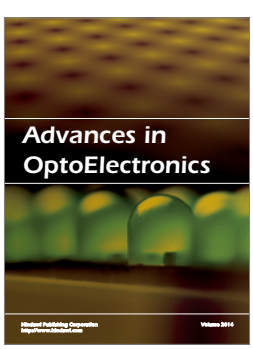
[37] D. Pointcheval and J. Stern, "Security arguments for digital signatures and blind signatures," *Journal of Cryptology*, vol. 13, no. 3, pp. 361–396, 2000.

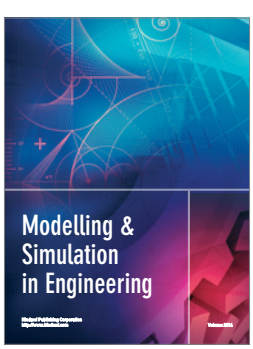
[38] M. Scott, N. Costigan, and W. Abdulwahab, "Implementing cryptographic pairings on smartcards," in *International Workshop on Cryptographic Hardware and Embedded Systems: CHES 2006: Cryptographic Hardware and Embedded Systems—CHES 2006*, Lecture Notes in Computer Science, pp. 134–147, Springer, Berlin, Germany, 2006.

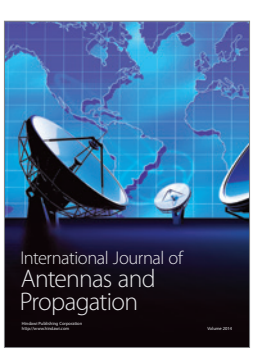


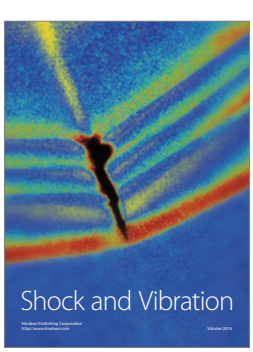


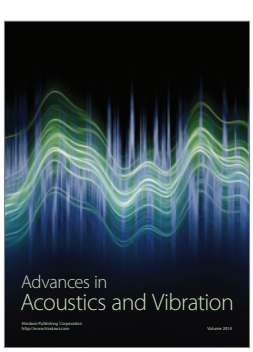












Hindawi

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