A Certificate-Based Provable Data Possession Scheme in the Standard Model

Caixue Zhou

School of Computer and Big Data Science, Jiujiang University, Jiujiang 332005, China

Correspondence should be addressed to Caixue Zhou; charlesjjjx@126.com

Received 31 March 2021; Revised 8 July 2021; Accepted 17 August 2021; Published 2 September 2021

1. Introduction

With the rapid development of the cloud storage technology, more and more users and companies store their data in the cloud. However, a new problem has emerged, that is, how users can ensure the integrity of their data if they no longer physically hold them. Luckily, provable data possession (PDP) [1] can resolve this problem. PDP is a lightweight probabilistic integrity checking model of cloud data. It can ensure the integrity of data stored in the cloud at a high probability even if users save no data or only a small amount of data locally. It is one of the core technologies to support cloud storage security.

The certificate-based cryptosystem [2] was proposed to reduce the high cost of public key certificate management in the public key infrastructure- (PKI-) based cryptosystem [3] and to eliminate the private key escrow problem inherent in the identity-based cryptosystem [4]. Just like in the PKI-based cryptosystem, a user’s public key in the certificate-based cryptosystem needs a certificate generated by the certificate authority (CA), except that in the latter case, the certificate should participate in the decryption or signing process along with the private key. Therefore, the certificate is verified implicitly. In this way, in the process of encryption or signature verification, the user need not care about whether the certificate has been revoked or out of date, and as a result, the cost of certificate management is reduced. In addition, the private key is produced by the user himself, so the private key escrow problem is also eliminated.

By combining the certificate-based cryptosystem and the concept of PDP, we propose a certificate-based PDP scheme. Based on the assumption that the Squ-CDH problem is hard, we prove the security of our scheme in the standard model. Based on the index logical table [5], our scheme can be extended to support dynamic operations easily. At last, we evaluate the efficiency of our scheme, which shows that it is efficient.
to protect users’ privacy when they audit data in public auditing schemes and proposed a privacy-preserving public PDP scheme. Meanwhile, in their paper, Wang et al. [8] also considered batch auditing so that even if there are multiple tasks from different users to be carried out simultaneously, TPA just needs to do one auditing task, which improves efficiency.

All the above schemes were proposed in the PKI-based cryptosystem. The main drawback in the PKI-based cryptosystem is the high cost of certificate management, which hinders its application on a large-scale. To reduce the cost of certificate management, identity-based PDP schemes [9, 10] were proposed. However, what is inherent in it is the private key escrow problem.

The certificate-based cryptosystem can not only resolve the private key escrow problem but also reduce the cost of public key certificate management, making it much superior. In 2015, Wang and Li [11] first proposed a certificate-based PDP scheme, but it supports neither public verification nor dynamic operations. In 2020, Wang et al. [12] proposed a lightweight certificate-based PDP scheme, which supports both private and public audits, but it does not support dynamic operations. Both of the above certificate-based PDP schemes rely on the random oracle model. Canetti et al. [13] showed that the random oracle is just an ideal model and that those schemes that are secure under the random oracle model will not be still secure when the random oracle is replaced by some concrete hash functions. Therefore, it is necessary to design certificate-based PDP schemes in the standard model. To the best of our knowledge, so far there have only been two certificate-based PDP schemes proposed in the literature, as mentioned above.

In terms of PDP schemes in the standard model, Zhang et al. [14] proposed an identity-based public PDP scheme, but Shen et al. [15] pointed out that the scheme [14] is insecure. A malicious cloud server can modify or delete users’ data arbitrarily, while still being able to produce a proof passing the verification equation as long as it keeps just one data block and its corresponding tag valid. Wang et al. [16] proposed a public and dynamic PDP scheme, but it is based on the PKI technology, which thus requires high cost of certificate management. Zhu et al. [17] proposed a shared data PDP scheme, where a group of users share the data stored in the cloud. It also supports malicious member revocation. Thokchom and Saikia [18] proposed an efficient privacy preserving public dynamic PDP scheme and also extended their scheme to support batch auditing in multiuser and multicloud scenarios, but they did not give security proof of their scheme. Yang et al. [19] proposed a quantum resistant lattice-based PDP scheme, which supports privacy preserving public auditing, dynamic operations, and batch auditing.

The rest of this work is organized as follows. In Section 2, we introduce the bilinear pairing and some complexity assumptions. In Section 3, we introduce the system model, the formal definition, and the security model of certificate-based PDP. In Section 4, we propose a concrete certificate-based PDP scheme in the standard model. In Section 5, we extend our scheme to support dynamic operations by using the index logic table. In Section 6, we prove the security of the proposed scheme. In Section 7, we evaluate the efficiency of the proposed scheme. We conclude the work in Section 8.

2. Preliminaries

2.1. Bilinear Pairing. Let $G_1$ and $G_2$ be two multiplicative cyclic groups of prime order $q$ and $g$ be a generator of $G_1$. If the following conditions hold, then it is deemed that the map $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear pairing.

(1) Bilinearity: for any $a$ and $b \in \mathbb{Z}_q$ and $P$ and $Q \in G_1$, the equation $e(P^a, Q^b) = e(P, Q)^{ab}$ holds

(2) Nondegeneracy: $e(g, g) \neq 1_{G_2}$

(3) Computability: for any $P$ and $Q \in G_1$, $e(P, Q)$ can be calculated efficiently

2.2. Complexity Assumption. Square Computational Diffie–Hellman (Squ-CDH) problem: given $g$ and $g^a \in G_1$, where $a \in \mathbb{Z}_q$ is selected randomly, one does not know the value of $a$ and needs to calculate $g^{a^2}$.

Let $A$ be a probabilistic polynomial time (PPT) algorithm, and the advantage of $A$ in solving the Squ-CDH problem on $G_1$ is defined to be $\text{Adv}_{A}^{\text{Squ-CDH}} = \Pr[A(g, g^a) = g^{a^2}, a \in \mathbb{Z}_q]$. Squ-CDH assumption: for every $A$, $\text{Adv}_{A}^{\text{Squ-CDH}}$ is negligible.

3. System Model, Definition, and Security Model of the Certificate-Based PDP

3.1. System Model. There are four entities involved in a certificate-based PDP scheme as illustrated in Figure 1.

- CA: it is a trusted third party who initializes the system and issues certificates for users.
- Users: they have a large amount of data to be stored in the cloud. When they decide to store their data in the cloud, they divide their data into file blocks and produce a tag for each file block. Then, they store their data along with these tags in the cloud and delete them locally.
- TPA: TPA is delegated by users to audit the data in the cloud. It periodically audits the data. When TPA plans to audit the data, it will launch a challenge to the cloud by randomly selecting a subset of the index numbers of file blocks to the cloud. After getting a proof from the cloud server, it checks if the proof can pass an equation in advance. If it can, then it is highly probable that the data stored in the cloud are intact.
- Cloud server: it provides cloud storage service. When it receives a challenge from TPA, it generates a proof by using the data blocks selected by TPA and their corresponding tags and sends the proof to TPA.

3.2. Definition of the Certificate-Based PDP. A certificate-based PDP scheme consists of the following seven algorithms.
Setup: given a security parameter $k$, it outputs the system’s public parameters $\text{params}$ and a master private key $s$. It is run by CA.

User-Key-Generation: given the system’s public parameters $\text{params}$, it outputs a public/private key pair $(pk, sk)$. It is run by the user.

Certificate-Gen: given a user’s public key $pk$, the system’s public parameters $\text{params}$, and the master private key $s$, it outputs a certificate $\text{cert}$ for the user. It is run by CA.

Tag-Gen: Given $n$ blocks $m_i \in \mathbb{Z}_q^*$ ($i = 1, ..., n$) of a file $F$, the user’s private key $sk$, and the system’s public parameters $\text{params}$, it outputs a tag $\text{tag}_i$ for each file block $m_i$. It is run by the user. Then, the user stores all $m_i$ and $\text{tag}_i$ to the cloud and deletes them locally.

Proof-Gen: given a subset of the index numbers of file blocks selected by TPA, the corresponding tags $\text{tag}_i$, and the system’s public parameters $\text{params}$, it outputs a proof of data. It is run by CSP and CSP sends the proof to TPA.

Proof-Verify: given the system’s public parameters $\text{params}$ and the proof, it outputs TRUE or FALSE. It is run by TPA.

Update-Op: It is run by CSP.

Inserting: given the system’s public parameters $\text{params}$, the index number $i$, and a file block $m_i$, it does the updating and outputs TRUE or FALSE.

Modifying: given the system’s public parameters $\text{params}$, the index number $i$, and a new file block $m'_i$, it does the updating and outputs TRUE or FALSE.

Deleting: given the system’s public parameters $\text{params}$ and the index number $i$, it does the updating and outputs TRUE or FALSE.

3.3. Security Model of the Certificate-Based PDP. There are two types of adversaries in the certificate-based cryptosystem [2]. The first type of adversary $A_I$ does not know the master private key, but he can replace anyone’s public key, as it models any adversary except CA. On the contrary, the second type of adversary $A_{II}$ knows the master private key, but he cannot replace anyone’s public key, as it models the honest-but-curious CA. The honest-but-curious CA means that CA will honestly execute the system protocol, but then he can attack the system. In 2009, Wu et al. [20] introduced the malicious-but-passive CA $A_{II}$ adversary in the certificate-based cryptosystem. This type of $A_{II}$ adversary may be malicious in the system protocol execution. In our security model, this type of adversary $A_{II}$ is considered.

Definition 1 (type I adversary). A certificate-based PDP scheme is secure if for any PPT adversary $A_I$, the probability that $A_I$ wins the following game is negligible.

Setup. Given a security parameter $k$, challenge $C$ runs the setup algorithm to generate a master private key $s$ and a common parameter $\text{params}$. $\text{params}$ are given to $A_I$ and $C$ keeps $s$ private.

Queries. $A_I$ can make the following queries adaptively.

(1) User-Creation queries: $A_I$ supplies an identity ID. If the key pair of identity ID has not been created, $C$
runs the User-Key-Generation algorithm to create a private/public key pair \((SK_{ID}, PK_{ID})\) of that user and returns \(PK_{ID}\) to \(A_i\); otherwise, \(C\) returns \(PK_{ID}\) to \(A_i\) directly.

(2) User-PrivateKey queries: \(A_i\) supplies an already created identity ID. \(C\) returns ID’s private key \(SK_{ID}\) to \(A_i\).

(3) PublicKey-Replacement queries: \(A_i\) supplies an already created identity ID and a new public/private key pair \((PK_{ID}, SK_{ID})\). \(C\) replaces the current public/private key pair \((PK_{ID}, SK_{ID})\) with the new pair \((PK_{ID}', SK_{ID}')\).

(4) Certificate queries: \(A_i\) supplies an already created identity ID and the corresponding public key \(PK_{ID}\). \(C\) runs the Certificate-Gen algorithm to produce ID’s certificate \(Cert_{ID}\) and sends it to \(A_i\).

(5) Tag-Gen queries: \(A_i\) supplies an already created identity ID and a file block \(m_j\). \(C\) runs the Tag-Gen algorithm to generate a tag of \((ID, m_j)\) and sends it back to \(A_i\).

(6) Update-Op queries: \(A_i\) supplies an already created identity ID and a request for a dynamic operation. \(C\) makes the corresponding dynamic operation.

**Challenge.** \(C\) generates a challenge \((ID^*, I^*, W^*)\) and sends it to \(A_i\), where \(I^*\) is a subset of the index numbers of file blocks and \(W^*\) is a set of random numbers.

**Forge.** \(A_i\) generates a proof \(P^*\) for \((ID^*, I^*, W^*)\) and sends it to \(C\).

\(A_i\) wins the game if the following conditions hold.

1. \(\text{True} = \text{ProofVerify}(ID^*, P^*)\)
2. \(A_i\) has not made a certificate query on \(ID^*\)
3. At least one query of Tag-Gen on \((ID^*, i^*, m_i^*)\) does not happen, where \(i^* \in I^*\)

**Definition 2** (type II adversary). A certificate-based PDP scheme is secure if for any PPT adversary \(A_{II}\), the probability that \(A_{II}\) wins the following game is negligible.

**Setup.** Given a security parameter \(k\), adversary \(A_{II}\) runs the setup algorithm to generate a master private key \(s\) and a common parameter \(Params\). \(Params\) and \(s\) are given to \(C\).

**Queries.** \(A_{II}\) can make User-Creation, User-PrivateKey, Tag-Gen, and Update-Op queries adaptively, and these queries are the same as in Definition 1. Other queries are not needed for \(A_{II}\).

**Challenge.** Same as in Definition 1.

**Forge.** \(A_{II}\) generates a proof \(P^*\) for \((ID^*, I^*, W^*)\) and sends it to \(C\).

\(A_{II}\) wins the game if the following conditions hold.

1. \(\text{True} = \text{ProofVerify}(ID^*, P^*)\)
2. \(A_{II}\) has not made a User-PrivateKey query on \(ID^*\)
3. At least one query of Tag-Gen on \((ID^*, i^*, m_i^*)\) does not happen, where \(i^* \in I^*\)

**Note.** In order to resist the malicious-but-passive CA \(A_{II}\) adversary, it must let adversary \(A_{II}\) run the setup algorithm rather than challenger \(C\).

**4. A Concrete Certificate-Based PDP Scheme**

**4.1. Concrete Scheme.** \(H: \{0,1\}^* \longrightarrow \{0,1\}^n\) denotes a collision-resistant cryptographic hash function for some \(n \in \mathbb{Z}_q^*\), which is used to create identities of the desired length.

1. **Setup:** given a security parameter \(k\), \(CA\) chooses two cyclic groups \(G_1\) and \(G_2\) of prime order \(q\), a random generator \(g\) of \(G_1\), a bilinear map \(e: G_1 \times G_1 \longrightarrow G_2\), three hash functions \(H: \{0,1\}^* \longrightarrow \mathbb{Z}_q^*\), \(H_1: \{0,1\}^* \longrightarrow \{0,1\}^n\), and \(H_2: \{0,1\}^* \longrightarrow G_2\), a pseudorandom function \(f: \mathbb{Z}_q^* \times \{1,2,\ldots,n\} \longrightarrow \mathbb{Z}_q^*\), and a pseudorandom permutation \(\pi: \mathbb{Z}_q^* \times \{1,2,\ldots,n\} \longrightarrow \{1,2,\ldots,n\}\). \(CA\) randomly chooses \(s, v, v_1, v_2, \ldots, v_n \in \mathbb{Z}_q^*\). \(CA\) runs the technique of Fiat–Shamir transform without \(\pi\) does. Let \(\nu_1, \nu_2, \ldots, \nu_n\) be a bit string. The public parameters are \(\text{Params} = \{G_1, G_2, e, g, g_1, u, F, H, H_1, H_2, f, \pi\}\), and the master secret key is \(msk = (s, v, v_1, v_2, \ldots, v_n)\).

2. **User-Key-Generation:** the user \(ID\) randomly selects \(x_{ID}\) and \(y_{ID} \in \mathbb{Z}_q^*\) as his secret keys and computes his public key as \(UPK_{ID} = (UPK_{ID,1}, UPK_{ID,2}) = (g^{y_{ID}}, g_1^{x_{ID}})\), where \(q_{ID} = y_{ID} + x_{ID} H(UPK_{ID}, \text{Params}) \mod q\) is a Schnorr one-time signature. The signature can be generated using the technique of Fiat–Shamir transform without random oracles as described in [21].

3. **Certificate-Gen:** given a user’s identity ID and his public key \(UPK_{ID}\), \(CA\) randomly selects \(s_{ID} \in \mathbb{Z}_q^*\) and computes the user’s certificate as \(Cert_{ID} = (Cert_{ID,1}, Cert_{ID,2}) = (g^{s_{ID} F_{ID}(\text{id}_{ID})}, g^{s_{ID}})\), where \(id_{ID} = H_1(ID, UPK_{ID,1}, UPK_{ID,2})\).

4. **Tag-Gen:** let a file \(F\) be divided into \(n\) blocks \(m_i \in \mathbb{Z}_q^*\), \(i = 1, 2, \ldots, n\). User ID computes \(t_{ID,i} = Cert_{ID,2}, W_{ID,1} = H_2(\text{ID}, UPK_{ID,1}, UPK_{ID,2}, UPK_{ID,3}, t_{ID,1}, f)\), and \(t_{ID,2,i} = W_{ID,1}(Cert_{ID,2})^{y_{ID}}\). Finally, the tags are \(\sigma = (t_{ID,1,i}, t_{ID,2,i})\) (\(i = 1, 2, \ldots, n\)).

5. **Proof-Gen:** TPA selects a random integer \(c\) and two random elements \(k_1, k_2 \in \mathbb{Z}_q^*\). TPA sends the challenge \((ID, c, k_1, k_2)\) to CS. CS computes \(i_j = n_{k_1}(j)\), and \(w_j = f_{k_2}(j)\), where \(j = 1, 2, \ldots, c\). Let \(I = \{i_1, i_2, \ldots, i_c\}\), and CS computes \(S = \prod_{i=1}^{c} t_{ID,2,i}\) and \(\delta = \sum_{i=1}^{c} w_i m_i\) and sends the proof \((I, S, \delta)\) to TPA.

6. **Proof-Verify:** TPA computes \(i_j = n_{k_1}(j)\), and \(w_j = f_{k_2}(j)\), where \(j = 1, 2, \ldots, c\). Let \(I = \{i_1, i_2, \ldots, i_c\}\),
5.2. Dynamic Operations. To make our scheme support dynamic operations, we must replace $W_{ID,j} = H_j(ID, UPK_{ID,1}, UPK_{ID,2}, UPK_{ID,3}, t_{ID,1}, i)$ with $W_{ID,j} = H_j(ID, UPK_{ID,1}, UPK_{ID,2}, UPK_{ID,3}, t_{ID,1}, \ln(i))$ in the Tag-Gen and the Proof-Verify algorithms; in other words, we must replace $i$ with $\ln(i)$. The ILT must be stored in users and TPA locally.

(1) Data modifying: when a user wants to modify a file block $m_i$ at $i_{in} = i$, he runs algorithm 1(b) first. Then, he computes tag’ for $m_i$. He uploads $m_i$, tag’, $i_{in}$, and ILT to the cloud and deletes $m_i$ and tag’ locally. CSP makes the corresponding modification.

(2) Data inserting: when a user wants to insert a file block $m''$ after $i_{in} = i$, he runs algorithm 1(c) first. Then, he computes tag” for $m''$. He uploads $m''$, tag”, $i_{in}$, and ILT to the cloud and deletes $m''$ and tag” locally. CSP makes the corresponding insertion.

(3) Data deleting: when a user wants to delete a file block $m_i$ at $i_{in} = i$, he runs algorithm 1(d) first. Then, he uploads $i_{in}$ and ILT to the cloud. CSP makes the corresponding deletion.

6. Unforgeability

Theorem 1 (type I unforgeability). In the standard model, if a PPT attacker $A_i$ has a nonnegligible advantage $\varepsilon$ in winning the game of Definition 1, running for time $t$ and performing at

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
\text{bin} & \text{ln} & \text{blocks} \\
\hline
\text{(a) Initial} & 1 & M1 & \\
\hline
2 & 2 & M2 & \\
3 & 3 & M3 & \\
4 & 4 & M4 & \\
5 & 5 & M5 & \\
6 & 6 & M6 & \\
7 & 7 & null & \\
\hline
\text{(b) Modify at 4} & 1 & M1 & \\
2 & 2 & M2 & \\
3 & 3 & M3 & \\
4 & 7 & M4’ & \\
5 & 5 & M5 & \\
6 & 6 & M6 & \\
7 & 8 & null & \\
\hline
\text{(c) Insert after 4} & 1 & M1 & \\
2 & 2 & M2 & \\
3 & 3 & M3 & \\
4 & 7 & M4’ & \\
5 & 8 & M” & \\
6 & 5 & M5 & \\
7 & 6 & M6 & \\
8 & 9 & null & \\
\hline
\text{(d) Delete at 2} & 1 & M1 & \\
2 & 3 & M3 & \\
3 & 7 & M4’ & \\
4 & 8 & M” & \\
5 & 5 & M5 & \\
6 & 6 & M6 & \\
7 & 9 & null & \\
\hline
\end{tabular}
\caption{Index logic table.}
\end{table}
most \( q_{\text{Creation}} \) user creation queries, \( q_{\text{Cert}} \) certificate queries, and \( q_{\text{Tag}} \) Tag-Gen queries, then there is an algorithm \( C \) that solves the Squ-CDH problem with an advantage \( \text{Adv}_{\text{Squ-CDH}} \geq \epsilon/4 (q_{\text{Cert}} + q_{\text{Tag}}) (n_u + 1) \) for time \( t + \hat{O}(q_{\text{Cert}}, n_u, t_m + (q_{\text{Creation}} + q_{\text{Cert}} + q_{\text{Tag}}) t_q) \), where \( t_m \) and \( t_q \) denote the time for a multiplication and an exponentiation on \( G_1 \), respectively.

Proof. Suppose \( C \) is given \( (g, B = g^b) \in G_1 \) for randomly chosen \( b \in \mathbb{Z}_q^* \). \( C \) does not know the value of \( b \) and is asked to compute \( g^{kb} \). To utilize adversary \( A_1 \), challenger \( C \) will all the oracles defined in Definition 1. \( C \) maintains a table \( tb = \{\text{ID}, \text{CertID}, \text{XID}, \text{YID}, \text{UPKID}, \text{tag} = 0\} \), which is initially empty. \( \text{tag} = 0 \) means that the public key is not replaced.

Setup Let \( l_u = 2 (q_{\text{Cert}} + q_{\text{Tag}}) \). \( C \) randomly chooses the following elements:

1. An integer \( k_u \) \((0 \leq k_u \leq n_u)\). We assume that \( l_u (n_u + 1) < q \) for the given values of \( q_{\text{Cert}}, q_{\text{Tag}} \), and \( n_u \).
2. An integer \( x_u \in \mathbb{Z}_q^* \), and a vector \( X_u = (x_{u_1}) \).
3. An integer \( y_u \in \mathbb{Z}_q^* \), and a vector \( Y_u = (y_{u_1}) \).

For convenience, we define the following functions:

\[
J_u (i, d) = x_u^i - l_u k_u + \sum_{j=1}^{n_u} j_i x_u, i, j)
\]

\[
K_u (i, d) = y_u^i + \sum_{j=1}^{n_u} j_i y_u, i, j)
\]

where \( id = i_1 i_2 ... i_{n_u} \) is a bit string.

Then, \( C \) randomly chooses two cyclic groups \( G_1 \) and \( G_2 \) of prime order \( q \), a random generator \( g \) of \( G_1 \), a bilinear map \( e: G_1 \times G_1 \rightarrow G_2 \), and three hash functions \( H: [0, 1]^* \rightarrow \mathbb{Z}_q^* \), \( H_1: [0, 1]^* \rightarrow \{0, 1\}^{n_u} \), and \( H_2: [0, 1]^* \rightarrow G_1 \). It sets \( g_1 = B, u_u = B^{x_u} \), and \( U = (u_u) \in G_1^{n_u} \). This assignment means \( F_u (id) = B^{x_u} \), and that the master secret key \( s = b \), which is not known to \( C \). \( C \) chooses a pseudorandom function \( f: \mathbb{Z}_q^* \times \{1, 2, ..., n\} \rightarrow \mathbb{Z}_q^* \) and a pseudorandom permutation \( \pi: \mathbb{Z}_q^* \times \{1, 2, ..., n\} \rightarrow \{1, 2, ..., n\} \). \( C \) outputs the public parameters \( \text{Params} = \{G_1, G_2, e, g, g_1, u_u, U, F_u, H, H_1, H_2, f, \pi\} \).

Queries. \( A_1 \) can adaptively make a polynomial bounded number of queries as follows.

1. User-Creation queries: \( A_1 \) supplies an identity \( ID \). \( C \) first checks the table \( tb \) to see whether it contains the item or not. If it does, \( C \) returns \( ID \)’s public key \( \text{UPKID} \) to \( A_1 \); otherwise, \( C \) executes the User-Key-Generation algorithm to produce \( ID \)’s private/public key pair \((x_{\text{ID}}, y_{\text{ID}}, \text{UPKID})\). \( C \) puts \((ID, \pi, x_{\text{ID}}, y_{\text{ID}}, \text{UPKID}, \text{tag} = 0)\) into the table \( tb \) and returns \( \text{UPKID} \) to \( A_1 \).

2. User-PrivateKey queries: \( A_1 \) supplies an already created identity \( ID \). \( C \) searches the table \( tb \) to find out the private keys \( x_{\text{ID}} \) and \( y_{\text{ID}} \) and returns them to \( A_1 \).

3. PublicKey-Replacement queries: \( A_1 \) supplies an already created identity \( ID \) and a new public/private key pair \((\text{UPKID}, \text{UPKID}')\). \( C \) replaces the current public/private key pair \((\text{UPKID}, x_{\text{ID}}, y_{\text{ID}})\) with the new key pair \((\text{UPKID}', x_{\text{ID}}', y_{\text{ID}}')\) in the table \( tb \) and sets \( \text{tag} = 1 \).

4. Certificate queries: \( A_1 \) supplies an already created identity \( ID \) and ID’s public key \( \text{UPKID} \). \( C \) first checks table \( tb \) to see whether the certificate \( \text{CertID} \) is produced or not. If it does, \( C \) returns the certificate \( \text{CertID} \) to \( A_1 \); otherwise, \( C \) computes \( id = H_1 (ID, \text{UPKID}, \text{UPKID}) \) and produces the certificate as follows (\( C \) does not know the master private key). Then, \( C \) returns the certificate \( \text{CertID} \) to \( A_1 \) and updates “−” with the certificate in table \( tb \).

(a) \( J_u (id) \neq 0 \mod q \). \( C \) randomly selects \( r_{\text{ID}} \in \mathbb{Z}_q^* \) and computes the user’s certificate as \( \text{CertID} = (\text{CertID}_1, \text{CertID}_2) = (B^{K_{\text{ID}} (id)} r_{\text{ID}}^\pi) \). \( J_u (id) \).

(b) \( J_u (id) \equiv 0 \mod q \). \( C \) aborts.

To make the analysis of the simulation easier, we will force \( C \) to abort whenever \( J_u (id) = 0 \mod q \).
implies

Let $I_u$ (id) $\neq 0 \mod d_u$ implies $I_u$ (id) $\neq 0 \mod q_u$, given the assumption $l_u (n_u + 1) < q_u$.

(5) Tag-Gen queries: $A_1$ supplies an already created identity ID and a file block $m_j$. $C$ computes $id = H_1 (ID, UPK_{ID,1}, UPK_{ID,2}, UPK_{ID,3})$.

(a) $I_u$ (id) $\neq 0 \mod d_u$. $C$ produces the tag as normal because $C$ can get the certificate and private key of ID.
(b) $I_u$ (id) $= 0 \mod d_u$. $C$ aborts.

(6) Update-Op queries: $A_1$ supplies an already created identity ID. For inserting operation, $A_1$ also supplies the “in” number in the ILT table and a new file block $m_i$; for modifying operation, $A_1$ also supplies the “in” number in the ILT table, the old file block $m_i$ and the new file block $m_i$; for deleting operation, $A_2$ also supplies the “in” number in the ILT table. CSP makes the corresponding dynamic operations and returns ILT to the user.

Challenge: $C$ generates a challenge $(ID^*, G, k, j, k_2)$ to $A_1$. $A_1$ computes $t_i = \pi_{k_j} (j)$, where $j = 1, 2, \ldots, c$. Let $I_1 = \{t_{i_1}, t_{i_2}, \ldots, t_{i_j}\}$.

Forge: $A_1$ generates a forged proof $(t_{1D^*,1}, S^*, \delta^*)$ for $(ID^*, I^*, (w_{i,j})_{i,j})$. According to Definition 1, at least one query of Tag-Gen on $(ID^*, i^*, m_i)$, where $i^* \in I^*$, must not happen that is, $A_1$ has generated a forged tag $\sigma^* = (t_{1D^*,1}, t_{1D^*,j^*}, i^*)$, $i^* \in I^*$, for file block $m_i$. $C$ computes $id^* = H_1 (ID^*, UPK_{ID,1}, UPK_{ID,2}, UPK_{ID,3})$.

If $I_u$ (id*) $\neq 0 \mod d_u$, then $C$ aborts; otherwise, we have $F_u (id*) = g^{k_2 (id*)}$. If $(t_{1D^*,1}, S^*, \delta^*)$ can pass the Proof-Verify algorithm and $A_1$ does not violate the restrictions of Definition 1, then $C$ retrieves the private key $x_{ID^*}$ of $ID^*$ and computes $W_{ID^*, \delta^*} = H_2 (ID^*, UPK_{ID,1}, UPK_{ID,2}, UPK_{ID,3}, t_{1D^*,1}, \delta^*)$ and $t_{1D^*, \delta^*} / W_{ID^*, \delta^*} = \text{Cert}_{ID^*} (\{t_i^*, \delta_i^* \text{mid} m_i^* \}) = W_{ID^*}^{t_{1D^*,1}} (\text{Cert}_{ID^*} (\{t_i^*, \delta_i^* \text{mid} m_i^* \}))$.

Now, we assess the probability of success. If the simulation is not aborted, the following conditions must be met.

(1) In all certificate and signature queries, $I_u$ (id) $\neq 0 \mod d_u$.

(2) In the forgery stage, $I_u$ (id*) $\neq 0 \mod d_u$.

Let ID$_1$, ID$_2$, …, ID$_q$ be the identities appearing in these queries, but they do not involve any of the challenge identities. Clearly, we have $q_1 \leq q_{\text{Cert}} + q_{\text{tag}}$. Define the events $A_1^*$, $A_2^*$, and $A_3^*$ as $A_1^*$: $I_u$ (id) $\neq 0 \mod d_u$ and $A_2^*$: $I_u$ (id) $\neq 0 \mod d_u$. Therefore, the probability of $C$ not aborting the simulation is $Pr[\text{abort}] \geq Pr[A_1^* \cap A_2^*]$.

The time complexity of challenger $C$ depends on the exponentiations, multiplications, and pairing operations needed in all above queries. The user creation queries need $O(1)$ exponentiations. The certificate queries need $O(n_u)$ multiplications and $O(1)$ exponentiations. The Tag-Gen queries need $O(1)$ exponentiations. Therefore, the time complexity of $C$ is $t + O(q_{\text{Cert}} \cdot n_u \cdot t_m + (q_{\text{Creation}} + q_{\text{Cert}} + q_{\text{tag}}) t_c)$.

Theorem 2 (type II unforgeability). In the standard model, if a PPT attacker $A_2$ has a nonnegligible advantage $\varepsilon$ in winning the game of Definition 2 running for time $t$ and performing at most $d_{\text{Creation}}$ user creation queries, $q_u$ private key queries, and $q_{\text{tag}}$ Tag-Gen queries, then there is an algorithm $C$ that solves the Squ-CDH problem with an advantage $Adv_{\text{C}}^{\text{Squ-CDH}} \geq \varepsilon / 4 (q_{\text{Cert}} + q_{\text{tag}}) (n_u + 1)$.

Proof. Let us suppose $C$ is given $(g, B = g^q) \in G_1$. For randomly chosen $b \in Z_q$, $C$ does not know the value of $b$ and is asked to compute $g^q$. To utilize the adversary $A_2$, challenger $C$ will simulate all the oracles defined in Definition 2. $C$ maintains a table $tb = \{ID, x_{ID}, y_{ID}, UPK_{ID}\}$, which is initially empty.

Setup. $A_2$ randomly chooses two cyclic groups $G_1$ and $G_2$ of prime order $q$, a random generator $g$ of $G_1$, a bilinear map $e: G_1 \times G_1 \rightarrow G_2$, and three hash functions $H: \{0, 1\}^* \rightarrow Z_q^*$, $H_1: \{0, 1\}^* \rightarrow \{0, 1\}^n$, and $H_2: \{0, 1\}^* \rightarrow G_1$, and $\alpha, \gamma$, $\gamma'_1, \gamma'_2, \gamma''_1, \gamma''_2, \gamma_1, \gamma_2, \gamma_3$. $C$ sets $g_1 = g^q$, $\gamma''_1 = g^{q_1} = g^{\gamma'_{1,1}} = g^{\gamma'_{1,2}} \ldots g^{\gamma'_{1,n}} \in Z_q^*$, $\gamma_1 = g^{\gamma''_1}$, $\gamma_2 = g^{\gamma''_2}$, and $\gamma_3 = g^{\gamma_1 + \gamma_2}$. $A_2$ chooses a pseudorandom function $f: \{0, 1\}^* \rightarrow \{0, 1\}^n$, and a pseudorandom permutation $\pi: \sum \rightarrow \sum$. $\pi_\gamma = \{\pi_\gamma (0), \pi_\gamma (1), \ldots, \pi_\gamma (n)\}$.

The master secret key is $(\alpha, \gamma, \gamma_1, \gamma_2, \gamma_3)$. $A_2$ outputs the public parameters $\gamma': [G_1, G_2, e, h, g_1, \alpha, \gamma', \gamma''_1, \gamma''_2, \gamma_1, \gamma_2, \gamma_3, \pi_\gamma, \gamma_4], \gamma''_1 = g^{\gamma''_1}, \gamma''_2 = g^{\gamma''_2}$, and $\pi_\gamma = \{\pi_\gamma (0), \pi_\gamma (1), \ldots, \pi_\gamma (n)\}$ to $C$.

As $h$ is a generator of $G_1$, there must be a $b' \in Z_q$ to make the equation $B = h^{b'}$ hold. Knowing $(h, B = h^q) \in G_1$, now $C$’s goal becomes computing $h^{b'}$. $C$ randomly selects an index $\pi$ from $\{1, 2, \ldots, n\}$.

Queries. $A_2$ can make the following queries adaptively.

(1) User-Creation queries: $A_2$ supplies an identity ID$_1$. $C$ first checks table $tb$ to see whether it contains the item or not. If it does, $C$ returns ID$_1$’s public key $UPK_{ID_1}$ to $A_2$. Otherwise, $C$ produces the public/private key pair as follows and returns $UPK_{ID_1}$ to $A_2$.

(a) $ID \neq ID_1$. $C$ executes the User-Key-Generation algorithm as normal to produce ID$_1$’s private/
public key pair \((x_{ID_i}, y_{ID_i}, UPK_{ID_i})\), \(C\) puts \((ID_i, x_{ID_i}, y_{ID_i}, UPK_{ID_i})\) into table \(tb\).

(b) \(ID_i = ID_{\pi}\). \(C\) randomly selects \(y_{ID_i} \in Z_q^*\) and computes \(ID_i\)’s public key as \(UPK_{ID_i} = (h^{m_{ID_i}}, h^{y_{ID_i}}, b^c)\), where \(c = H(K, UPK_{ID_1}, UPK_{ID_2}, \text{Params})\). \(C\) puts \((ID_i, y_{ID_i}, UPK_{ID_i})\) into table \(tb\) (the implicitly defined private key \(x_{ID_i} = b^i\), which is not known to \(C\)).

(2) Private Key queries: \(A_{\Pi}\) supplies an already created identity \(ID_i\).

(a) \(ID_i \neq ID_{\pi}\). \(C\) searches table \(tb\) to find out the private keys \(x_{ID_i}\) and \(y_{ID_i}\) and returns them to \(A_{\Pi}\).

(b) \(ID_i = ID_{\pi}\). \(C\) aborts.

(3) Tag-Gen queries: \(A_{\Pi}\) supplies an already created identity \(ID_i\) and a file block \(m_i\). \(C\) computes \(id = H_1(\text{id}, UPK_{ID_1}, UPK_{ID_2}, UPK_{ID_3})\).

(a) \(ID_i \neq ID_{\pi}\). \(C\) produces the tag as normal because \(C\) can get the certificate and private key of \(ID_i\).

(b) \(ID_i = ID_{\pi}\). \(C\) aborts.

(4) Update-Op queries: same as in Theorem 1.

7. Analysis of Efficiency

We analyse the efficiency of our scheme in terms of computational time, communication overhead, and storage cost. Let \(h, c_1, c_2, \text{ and } p\) denote a map-to-point hash computation, an exponentiation computation on \(G_1\), an exponentiation computation on \(G_2\), and a bilinear pairing computation, respectively. We compare our scheme with those proposed also in the standard model, which include schemes [14, 16, 18]. To show a more direct comparison with these schemes, we also adopt the experiment results of the scheme [23].

The computation time of each operation is shown in Table 3 and of each scheme is shown in Table 4. By combining Tables 3 and 4, we get Table 5. Since the scheme [14, 16] also divided a file block into \(s\) sectors, for the sake of fairness, we set \(s = 1\) in Table 5. From Table 5, we can get Figure 2 for CSP’s running time and Figure 3 for TPA’s running time. From Table 5, we can see that the scheme [18] is the most efficient one in terms of the Tag-Gen algorithm. From Figure 2, we can see that our scheme is the most efficient one in terms of TPA’s running time (the Proof-Gen algorithm). From Figure 3, we can see that scheme [14] is the most efficient one in terms of TPA’s running time (the Proof-Verify algorithm).

The communication overhead of each scheme is shown in Table 6. Based on the experiment results of the scheme [23], the size of \(q\) is 512 bits, and if the technique of point compression is used, the size of an element in \(G_1\) or \(G_2\) is 512 bits. An integer is represented by 64 bits. From this, we get Table 7. From Table 7 and by taking \(s = 1\), we obtain Figure 4. From Table 7, we can see that our scheme is the most efficient one in terms of both storage and communication of tags. From Figure 4, we can see that our scheme is the most efficient one in the communication of Proof-Gen.

We also compare our scheme with those certificate-based PDP schemes, which include schemes [11, 12]. The computation time of each scheme is shown in Table 8. By combining Tables 3 and 8, we get Table 9. From Table 9, we can get Figure 5 for CSP’s running time and Figure 6 for TPA’s running time. From Table 9, we can see that the scheme [11] is the most efficient one in terms of the Tag-Gen algorithm. From Figure 5, we can see that all schemes are the same in terms of the CSP’s running time (the Proof-Gen algorithm). From Figure 6, we can see that scheme [11] is the most efficient one in terms of TPA’s running time (the Proof-Verify algorithm).
The communication overhead of each scheme is shown in Table 10. By using the concrete parameters, we get Table 11. From Table 11, we can see that the scheme [11] is the most efficient one in terms of the Tag-Gen algorithm, and thus, the scheme [11] is the most efficient one in terms of both storage and communication of tags. Also from Table 11, we can see that all schemes are almost the same in the communication of Proof-Gen. Therefore, it can be concluded that our scheme is an efficient scheme.

In our system, the storage cost of ILT is linear to the size of the outsourced data and the file block is directly used as an exponent in the tag generation algorithm. Now, let us analyse how the file block size affects the efficiency. If the file

Table 3: Computational time (ms).

|   | H    | E1   | E2   |  p  |
|---|------|------|------|-----|
|   | 18.673 | 7.736 | 0.160 | 15.500 |

Table 4: Computational time.

| Schemes | Tag-Gen | Proof-Gen | Proof-Verify |
|---------|---------|-----------|-------------|
| [14]    | (2n + s) e1 | 3ce1 + 3p + (s + 1) e1 | |
| [16]    | ne1 + 6e1 | e1 + e1 | 6p + (c + s + 4) e1 |
| [18]    | nc1 | c (p + 3e1) | c (p + e2) |
| Ours    | (n + 1) e1 + nh | ce1 | 7p + ce1 + 3e2 + h |

n denotes the total number of file blocks, s means that a file block is divided into s sectors, and c denotes the total number of challenge blocks.

Table 5: Computational time.

| Schemes | Tag-Gen | Proof-Gen | Proof-Verify |
|---------|---------|-----------|-------------|
| [14]    | 15.472n + 7.736 | 23.208c | 61.972c |
| [16]    | 30.944n + 46.416 | 15.472c + 69.708 | 131.68c |
| [18]    | 7.736n | 38.708c | 15.66c |
| Ours    | 26.409n + 7.736 | 7.736c | 7.736c + 127.653 |

Table 6: Communication overhead (bit).

| Schemes | Tag-Gen | Proof-Gen |
|---------|---------|-----------|
| [14]    | 2n|G1| | |N| + |q| + 2|G1| + 3|q| |
| [16]    | n|G1| + n|q| + 4| |N| + |q| |c| + (s + 2)|q| + (c + 4) |G1| |
| [18]    | n| (|G1| + |q|) | (|N| + |q|)|c| + |G2| |
| Ours    | (n + 1)|G1| | |N| + 3|q| + |G1| |

Table 7: Communication overhead (bit).

| Schemes | Tag-Gen | Proof-Gen |
|---------|---------|-----------|
| [14]    | 1024n | 64c + 160s + 2284 |
| [16]    | 1184n + 2048 | 736c + 160s + 2368 |
| [18]    | 672n | 224c + 512 |
| Ours    | 512n + 512 | 1056 |

Figure 2: Comparisons of CSP’s running time.

Figure 3: Comparisons of TPA’s running time.

Figure 4: Comparisons of CSP’s communication overhead.
### Table 8: Computational time.

| Schemes | Tag-Gen       | Proof-Gen | Proof-Verify |
|---------|---------------|-----------|--------------|
| [11]    | $(n+1)e_1$   | $ce_1$    | $2p + (c + 3)e_1$ |
| [12]    | $(3n + s)e_1 + (n + 2)h$ | $ce_1$    | $2p + (c + 4)e_1 + h$ |
| Ours    | $(n + 1)e_1 + nh$ | $ce_1$    | $7p + ce_1 + 3e_2 + h$ |

### Table 9: Computational time.

| Schemes | Tag-Gen       | Proof-Gen | Proof-Verify |
|---------|---------------|-----------|--------------|
| [11]    | $7.736n + 7.736$ | $7.736c$  | $7.736c + 54.208$ |
| [12]    | $41.881n + 7.736s + 37.346$ | $7.736c$  | $7.736c + 80.617$ |
| Ours    | $26.409n + 7.736$ | $7.736c$  | $7.736c + 127.653$ |

**Figure 5:** Comparisons of CSP’s running time.

**Figure 6:** Comparisons of TPA’s running time.
a certificate-based PDP scheme is proposed. Based on the
To exploit the advantages of certificate-based cryptosystems,

8. Conclusions
To exploit the advantages of certificate-based cryptosystems, a certificate-based PDP scheme is proposed. Based on the

Data Availability
Previously reported simulation results of PBC library were used to support this study and are available at DOI: 10.1109/TII.2017.2761806. These prior studies are cited at relevant places within the text as references [23].

Conflicts of Interest
The author declares that there are no conflicts of interest.

Acknowledgments
The author thanks Ms Yan Di for checking the manuscript. This work was supported by the Scientific and Technological Research Project of Jiangxi Provincial Education Department of China under Grant GJJ201807, the National Natural Science Foundation of China under Grant 62041603, and the Key Project of Jiangxi Provincial Natural Science Foundation of China under Grant 20202ACBL202005.

References
[1] G. Atieniese, R. Burns, R. Curtmola et al., “Provable data possession at untrusted stores,” in Proceedings of the 2007 ACM Conference on Computer and Communications Security, pp. 598–609, CCSP, Alexandria, VA, USA, November 2007.
[2] C. Gentry, “Certificate-based encryption and the certificate revocation problem,” Advances in Cryptology-EUROCRYPT, LNCS, vol. 2656, pp. 272–293, 2003.
[3] S. Park, H. Kim, and J. Ryu, “Utilizing a lightweight PKI mechanism to guarantee a secure service in a cloud environment,” The Journal of Supercomputing, vol. 74, no. 12, pp. 6988–7002, 2018.
[4] Q. Feng, D. B. He, H. Q. Wang, N. Kumar, and K. K. R. Choo, “White-box implementation of shamir’s identity-based signature scheme,” IEEE Systems Journal, vol. 14, no. 2, pp. 1820–1829, 2020.
[5] F. Wang, L. Xu, H. Q. Wang, and Z. D. Chen, “Identity-based non-repudiable dynamic provable data possession in cloud storage,” Computers & Electrical Engineering, vol. 69, pp. 521–533, 2018.
[6] C. C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, “Dynamic provable data possession,” in Proceedings of the 2009 ACM Conference on Computer and Communications Security, pp. 213–222, CCS 2009, Chicago, IL, USA, November 2009.
[7] Q. Wang, C. Wang, J. Li, K. Ren, and W. J. Lou, “Enabling public verifiability and data dynamics for storage security in cloud computing,” in Proceedings of 14th European Symposium on Research in Computer Security (ESORICS 2009), pp. 355–370, Saint-Malo, France, September 2009.
[8] C. Wang, Q. Wang, K. Ren, and W. J. Lou, “Privacy-preserving public auditing for data storage security in cloud computing,” in Proceedings of 29th IEEE International Conference on Computer Communications (Infocom 2010), pp. 525–533, San Diego, CA, USA, March 2010.
[9] R. Rabaninejad, M. A. Attari, M. R. Asaar, and M. R. Aref, “A lightweight identity-based provable data possession supporting users’ identity privacy and traceability,” Journal of Information Security and Applications, vol. 51, Article ID 102454, 2020.

[10] H. Q. Wang, D. B. He, J. Yu, and Z. W. Wang, “Incentive and unconditionally anonymous identity-based public provable data possession,” IEEE Transactions on Services Computing, vol. 12, no. 5, pp. 824–835, 2020.

[11] H. Q. Wang and J. G. Li, “Private certificate-based remote data integrity checking in public clouds,” in Proceedings of 21st International Conference of Computing and Combinatorics (COCOON) 2015, pp. 575–586, Beijing, China, August 2015.

[12] F. Wang, L. Xu, K. K. R. Choo, Y. X. Zhang, H. Q. Wang, and J. G. Li, “Lightweight certificate-based public/private auditing scheme based on bilinear pairing for cloud storage,” IEEE Access, vol. 8, pp. 2258–2271, 2020.

[13] R. Canetti, O. Goldreich, and S. Halevi, “The random oracle methodology, revisited,” Journal of the ACM, vol. 51, no. 4, pp. 557–594, 2004.

[14] J. H. Zhang, P. Y. Li, and J. Mao, “IPad: ID-based public auditing for the outsourced data in the standard model,” Cluster Computing-the Journal of Networks Software Tools and Applications, vol. 19, no. 1, pp. 127–138, 2016.

[15] W. T. Shen, J. Yu, H. Xia, and R. Hao, “A note on “IPad: id-based public auditing for the outsourced data in the standard model”” in Proceedings of Second International Conference of Cloud Computing and Security (ICCCS 2016), pp. 335–342, Nanjing, China, July 2016.

[16] T. Wang, B. Yang, H. Y. Liu, Y. Yu, G. Y. Qiu, and Z. Xia, “An alternative approach to public cloud data auditing supporting data dynamics,” Soft Computing, vol. 23, no. 13, pp. 4939–4953, 2019.

[17] L. H. Zhu, H. Y. Wang, C. Xu, K. Sharif, and R. X. Lu, “Efficient group proof of storage with malicious-member distinction and revocation,” IEEE Access, vol. 7, pp. 75476–75489, 2019.

[18] S. Thokchom and D. K. Saikia, “Privacy preserving and public auditable integrity checking on dynamic cloud data,” International Journal on Network Security, vol. 21, no. 2, pp. 221–229, 2019.

[19] Y. Yang, Q. Huang, and F. Chen, “Secure cloud storage based on RLWE problem,” IEEE Access, vol. 7, pp. 27604–27614, 2019.

[20] W. Wu, Y. Mu, W. Susilo, and X. Y. Huang, “Certificate-based signatures revisited,” Journal of Universal Computer Science, vol. 15, no. 8, pp. 1659–1684, 2009.

[21] M. Bellare and S. Shoup, “Two-tier signatures, strongly unforgeable signatures, and Fiat-Shamir without random oracles,” in Proceedings of 10th International Conference on Practice and Theory in Public-Key Cryptography (PKC 2007), pp. 201–216, Beijing, China, April 2007.

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

[23] D. B. He, N. Kumar, S. Zeadally, and H. Q. Wang, “Certificateless provable data possession scheme for cloud-based smart grid data management systems,” IEEE Transactions on Industrial Informatics, vol. 14, no. 3, pp. 1232–1241, 2018.