Delegable Provable Data Possession for Remote Data in the Clouds

Shiuan-Tzuo Shen and Wen-Guey Tzeng*

Department of Computer Science,
National Chiao Tung University,
Hsinchu, Taiwan 30010
{vink,wgtzeng}@cs.nctu.edu.tw

Abstract. Many storage systems need to do authorized verification for data integrity. For example, a user stores his data into cloud storage servers and shares his data with his friends. They check data integrity periodically to ensure data intact. However, they don't want a stranger to check data integrity on their data. Therefore, public verification is undesired in this situation. The user can share his private key to his friends for private verification. However, his friends may reveal his private key to others. In this paper, we proposed the delegable provable data possession (delegable PDP) model to solve this problem. Delegable PDP allows a user to control who can check data integrity of his data, and guarantee that delegated verifiers cannot re-delegate this verification capability to others. Delegable PDP enjoys advantage of authorized verification and convenience of public verification.

We define a delegable PDP model and provide a construction for it. User $\mathcal{U}$ generates verifiable tags of his data and the delegation key $dk_{\mathcal{U}\to\mathcal{V}}$ for delegated verifier $\mathcal{V}$. $\mathcal{U}$ uploads his data, tags, and $dk_{\mathcal{U}\to\mathcal{V}}$ to storage servers. When integrity check, storage servers can use $dk_{\mathcal{U}\to\mathcal{V}}$ to transform $\mathcal{U}$'s tags into the form that $\mathcal{V}$ can verify with his private key $sk_{\mathcal{V}}$. Our model allows $\mathcal{U}$ to revoke $\mathcal{V}$'s verification capability by removing $dk_{\mathcal{U}\to\mathcal{V}}$ from storage servers directly. We prove our protocol secure in the random oracle model. Our protocol achieves proof unforgeability, proof indistinguishability, and delegation key unforgeability.

1 Introduction

Cloud computing provides computing services via networks such that a user can access these services anywhere at any time. For example, Amazon Elastic Compute Cloud (Amazon EC2) provides cloud computation and Amazon Simple Storage Service (Amazon S3) provides cloud storage. Storing data in a cloud storage system is quite convenient. One can share data to other users or synchronize copies in local devices. However, it brings security issues, privacy and integrity, on stored data. Users don't want their data leaked or modified without

* This research is supported by parts of NSC projects NSC100-2218-E-009-003-, NSC100-2218-E-009-006-, and NSC100-2219-E-009-005-, Taiwan.

S. Qing et al. (Eds.): ICICS 2011, LNCS 7043, pp. 93–111, 2011.

their permission. In general, encryption can provide data privacy and signature can provide data integrity. Users can encrypt their data and sign ciphertexts before uploading them to cloud storage servers. One way to make sure that a ciphertext is stored intactly is to retrieve the ciphertext together with its signature and verify it. This approach needs large bandwidth since data are retrieved back through networks. Thus, many researchers proposed methods to reduce the bandwidth need.

Ateniese et al. proposed a provable data possession (PDP) model [1]. Their PDP model allows a storage server to generate a probabilistic proof of size $O(1)$ for data integrity check so that a verifier can validate the proof efficiently. Their PDP protocol is asymmetric-key based such that public verification is done by everyone using the public key of the owner. However, public verification is undesirable in many circumstances. In contrast, private verification allows only the owner who possesses the secret key to verify data integrity. The owner can share this secret key to another user for data integrity check. However, the other one may leak this secret key.

In this paper, we define a model for delegable provable data possession (delegable PDP) that allows delegable (authorized) verification. In delegable PDP, a user who owns data can authorize another user to verify data integrity of his data. The authorized user cannot re-delegate this verification capability to others unless the authorized user reveals his private key. The delegable PDP model provides a balance between totally public and totally private integrity checking. Delegable PDP has two goals:

- Proof of data possession. A storage server can generate a valid proof if and only if it really stores the data. This proof can be verified without retrieving back the data from the storage server.
- Delegation of verification capability. A user can delegate his verification capability on his data to another user. The delegated user cannot re-delegate this verification capability to others. The delegated user can verify data integrity with storage servers on behalf of the user. The user can revoke the right of integrity checking from the delegated user directly.

Our delegable PDP model is efficient. To delegate, data owner $\mathcal{U}$ doesn't need to re-tag his data for delegated verifier $\mathcal{V}$. Instead, $\mathcal{U}$ generates the delegation key $dk_{\mathcal{U}\to\mathcal{V}}$ and uploads it to storage servers. Thus, $\mathcal{V}$ doesn't store and doesn't know $dk_{\mathcal{U}\to\mathcal{V}}$. To revoke, $\mathcal{U}$ sends the revoking command of deleting $dk_{\mathcal{U}\to\mathcal{V}}$ to storage servers directly. The cost of delegation is lightweight.

1.1 Delegable Provable Data Possession

There are three roles, user (data owner) $\mathcal{U}$, delegated verifier $\mathcal{V}$, and storage server $\mathcal{S}$, in the delegable PDP model. The data are stored in $\mathcal{S}$ after tagged by $\mathcal{U}$'s private key $sk_{\mathcal{U}}$. For delegation, $\mathcal{U}$ computes the delegation key $dk_{\mathcal{U}\to\mathcal{V}}$ by using $sk_{\mathcal{U}}$ and $\mathcal{V}$'s public key $pk_{\mathcal{V}}$, and sends it to $\mathcal{S}$. $\mathcal{S}$ transforms the tags of the data by using $dk_{\mathcal{U}\to\mathcal{V}}$ such that $\mathcal{V}$ can use his private key $sk_{\mathcal{V}}$ to verify the data by the transformed tags.

A delegable PDP scheme has three phases: the setup phase, the delegation phase, and the integrity check phase. The setup phase consists of three algorithms, Setup, KeyGen, and TagGen, as follows:

- $\mathsf{Setup}(1^k) \rightarrow \pi$. It is a probabilistic polynomial time algorithm run by the system manager to set up a delegable PDP system. Setup takes as input the security parameter k and outputs the public parameter π.
- $\mathsf{KeyGen}(\pi) \rightarrow (sk, pk)$. It is a probabilistic polynomial time algorithm run by a user to generate his key pair. KeyGen takes as input the public parameter π and outputs a private-public key pair (sk, pk) for the user.
- $\mathsf{TagGen}(\pi, sk, m) \rightarrow (\sigma, t)$. It is a deterministic polynomial time algorithm run by a user to generate verifiable tags for his data. TagGen takes as input the public parameter π, the user's private key sk, and the user's data m, and outputs a tag σ for m and an identifier t for σ.

The delegation phase consists of two algorithms, GenDK and VrfyDK, as follows:

- $\mathsf{GenDK}(\pi, sk_{\mathcal{U}}, pk_{\mathcal{V}}) \rightarrow dk_{\mathcal{U}\rightarrow\mathcal{V}}$. It is a deterministic polynomial time algorithm run by $\mathcal{U}$ to generate a delegation key for $\mathcal{V}$. GenDK takes as input the public parameter π, $\mathcal{U}$'s private key $sk_{\mathcal{U}}$, and $\mathcal{V}$'s public key $pk_{\mathcal{V}}$, and outputs the delegation key $dk_{\mathcal{U}\rightarrow\mathcal{V}}$.
- $\mathsf{VrfyDK}(\pi, dk_{\mathcal{U}\rightarrow\mathcal{V}}, pk_{\mathcal{U}}, pk_{\mathcal{V}}) \rightarrow \{true, false\}$. It is a deterministic polynomial time algorithm run by $\mathcal{S}$ to verify delegation keys. VrfyDK takes as input the public parameter π, the delegation key $dk_{\mathcal{U}\rightarrow\mathcal{V}}$, $\mathcal{U}$'s public key $pk_{\mathcal{U}}$, and $\mathcal{V}$'s public key $pk_{\mathcal{V}}$, and outputs the verification result.

The integrity check phase consists of three algorithms, GenChal, GenProof, and VrfyProof, as follows:

- $\mathsf{GenChal}(\pi, t) \rightarrow chal$. It is a probabilistic polynomial time algorithm run by $\mathcal{V}$ to generate a challenge to $\mathcal{S}$ for $\mathcal{U}$'s stored data. GenChal takes as input the public parameter π and tag identifier t, and outputs the challenge $chal$.
- $\mathsf{GenProof}(\pi, m, \sigma, dk_{\mathcal{U}\rightarrow\mathcal{V}}, chal) \rightarrow pf_{chal,\mathcal{V}}$. It is a probabilistic polynomial time algorithm run by $\mathcal{S}$ to generate a proof for integrity of the challenged data. GenProof takes as input the public parameter π, the stored data m, the tag σ for m, the delegation key $dk_{\mathcal{U}\rightarrow\mathcal{V}}$, and the challenge $chal$, and outputs the proof $pf_{chal,\mathcal{V}}$.
- $\mathsf{VrfyProof}(\pi, chal, pf_{chal,\mathcal{V}}, t, sk_{\mathcal{V}}) \rightarrow \{true, false\}$. It is a deterministic polynomial time algorithm run by $\mathcal{V}$ to verify a proof from $\mathcal{S}$. VrfyProof takes as input the public parameter π, the challenge $chal$, the proof $pf_{chal,\mathcal{V}}$, the identifier t, and $\mathcal{V}$'s private key $sk_{\mathcal{V}}$, and outputs the verification result.

1.2 Related Work

Ateniese et al. [1] defined the PDP model. They proposed an asymmetric-key based PDP construction which uses homomorphic verifiable tags on stored data. Under the homomorphic property, storage servers can generate proofs for any

linear combination of the stored data. Later on, Ateniese et al. [2] proposed a symmetric-key PDP construction which supports dynamic operations on stored data. Their construction is scalable and efficient. However, the number of data possession checkings is limited by the number of embedded tokens. Erway et al. [12] proposed dynamic provable data possession (DPDP) which uses rank-based skip list to support dynamic data operations. Ateniese et al. [3] proposed a framework for constructing public-key based PDP protocols. Their framework builds public-key homomorphic linear authenticators (HLAs) from public-key identification schemes, which satisfy certain homomorphic properties, and uses the HLA as a building block to construct PDP protocols.

Juels and Kaliski [14] proposed the proofs of retrievability (POR) model. POR ensures that stored data can be retrieved by users, while PDP ensures that data are stored in storage servers. Juels and Kaliski's construction embeds sentinels (verifying information of precomputed challenge-response pairs) into stored data, and the number of checkings is limited by the number of embedded sentinels. Later on, Shacham and Waters [15] proposed a compact POR which achieves an unlimited number of checkings. Bowers et al. [7] proposed a theoretical framework of designing POR protocols. This framework employs two layers of error correcting codes which recover user data from a series of responses. Their framework improves previous results of POR and has security proved in the fully Byzantine adversarial model. Wang et al. [17] proposed a POR scheme which supports dynamic operations and public verification on stored data. Their construction uses Merkle hash tree to support data dynamics.

To simultaneously achieve high availability and integrity checking for stored data, multiple replicas or the coding theory can be employed. Curtmola et al. [10] proposed MR-PDP that makes sure each unique replica exists in storage servers. Curtmola et al. [9] proposed a robust remote data integrity checking method that uses forward error correction codes. Later on, Bowers et al. [6] proposed HAIL which provides a high-availability and integrity layer for cloud storage. HAIL uses erasure codes on the single server layer and multiple sever layer respectively. It ensures data retrievability among distributed storage servers.

A cloud storage system may be viewed as a set of distributed storage servers. One can use the network coding technique to dispatch data to storage servers. For this model, Chen et al. [8] proposed a remote data integrity checking method for network coding-based distributed storage systems.

Wang et al. [16] proposed privacy-preserving public auditing for data storage security in cloud computing. Public data integrity checking may leak information about stored data by proofs to verifiers. Wang et al. use a blinding technique to hide information about stored data in proofs.

2 Preliminary

Our delegable PDP protocol uses the *bilinear map*. The security of our protocol is based on the *truncated (decision) bilinear Diffie-Hellman exponent assumption*, the *inverse computation Diffie-Hellman assumption*, and the *knowledge of exponent assumption* in the random oracle model.

Bilinear Map. Let q be a large prime, $G = \langle g \rangle$ and $G_T = \langle g_T \rangle$ be two multiplicative groups of prime order q. A bilinear map $\hat{e} : G \times G \rightarrow G_T$ should satisfy the following properties:

- Bilinearity. $\forall x, y \in \mathbb{Z}_q$, $\hat{e}(g^x, g^y) = \hat{e}(g, g)^{xy}$.
- Non-Degeneration. $\hat{e}(g, g) = g_T$.
- Computability. $\forall x, y \in \mathbb{Z}_q$, $\hat{e}(g^x, g^y)$ can be computed in polynomial time.

Truncated Bilinear Diffie-Hellman Exponent Assumption. Boneh et al. introduced the *bilinear Diffie-Hellman exponent* (BDHE) problem [4,5]. Later on, Gentry introduced two variants: the *augmented bilinear Diffie-Hellman exponent* (ABDHE) problem and the *truncated version* of the ABDHE problem [13].

The ℓ-BDHE problem is that: given a vector

$$\left(g', g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^{\ell}}, g^{\alpha^{\ell+2}}, g^{\alpha^{\ell+3}}, \ldots, g^{\alpha^{2\ell}}\right) \in G^{2\ell+1} \ ,$$

output $\hat{e}(g, g')^{\alpha^{\ell+1}} \in G_T$. The *truncated version* of the ℓ-BDHE problem, omitting ($g^{\alpha^{\ell+2}}$, $g^{\alpha^{\ell+3}}$, ..., $g^{\alpha^{2\ell}}$) from the input vector, is defined as that: given a vector

$$\left(g', g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^{\ell}}\right) \in G^{\ell+2} \ ,$$

output $\hat{e}(g, g')^{\alpha^{\ell+1}} \in G_T$. The advantage for an algorithm $\mathcal{A}$ that solves the *truncated* ℓ-BDHE problem is defined as:

$$\Pr\left[\mathcal{A}(g', g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^{\ell}}) = \hat{e}(g, g')^{\alpha^{\ell+1}} : g, g' \in_R G, \alpha \in_R \mathbb{Z}_q\right] \ .$$

The advantage for an algorithm $\mathcal{A}$ that solves the *truncated decisional* ℓ-BDHE problem is defined as:

$$\Big|\Pr[\mathcal{A}(g', g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^{\ell}}, \hat{e}(g, g')^{\alpha^{\ell+1}}) = 0 : g, g' \in_R G, \alpha \in_R \mathbb{Z}_q] \ - \\ \Pr[\mathcal{A}(g', g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^{\ell}}, Z) = 0 : g, g' \in_R G, \alpha \in_R \mathbb{Z}_q, Z \in_R G_T]\Big| \ .$$

Definition 1. *We say that the truncated (decisional) BDHE assumption is (t, ϵ, ℓ)-secure if no t-time algorithms have advantage over ϵ in solving the truncated (decisional) ℓ-BDHE problem.*

Inverse Computational Diffie-Hellman Assumption. The InvCDH problem is defined as that: given $(g, g^{\alpha}) \in G^2$ as input, output $g^{\frac{1}{\alpha}} \in G$. The advantage for a probabilistic algorithm $\mathcal{A}$ to solve the InvCDH problem is:

$$\Pr\left[\mathcal{A}(g, g^{\alpha}) = g^{\frac{1}{\alpha}} : \alpha \in_R \mathbb{Z}_q\right] \ .$$

Definition 2. *We say that the InvCDH assumption is (t, ϵ)-secure if no t-time algorithms have advantage over ϵ in solving the InvCDH problem*

Knowledge of Exponent Assumption. Damgard introduced the *knowledge of exponent assumption* (KEA1) [11]. Consider the problem: given $(g, g^\alpha) \in G^2$, output $(C, Y) \in G^2$ such that $C^\alpha = Y$. One way to output the pair is to choose $c \in_R \mathbb{Z}_q$ and let $(C, Y) = (g^c, g^{\alpha c})$. The KEA1 says that this is the only way to output such a pair in polynomial time. That is, if an adversary $\mathcal{A}$ takes (g, g^α) as input and outputs (C, Y) such that $C^\alpha = Y$, he must know the exponent c of $g^c = C$. There exists an extractor $\overline{\mathcal{A}}$ who extracts the exponent c such that $g^c = C$ when he is given the same inputs as $\mathcal{A}$'s.

3 Construction

In this section, we provide a delegable PDP scheme. Let k be the security parameter, q be a large prime with $|q| = k$, and $G = \langle g \rangle$ and $G_T = \langle g_T \rangle$ be two order-q multiplicative groups with a bilinear map $\hat{e} : G \times G \to G_T$. The system manager chooses three cryptographic hash functions $H_1 : \{0,1\}^* \to G$, $H_2 : G \to G$, and $H_3 : (\mathbb{Z}_q)^* \to G$. The public parameter is $\pi = (q, G, g, G_T, g_T, \hat{e}, H_1, H_2, H_3)$.

Key Generation. $\mathcal{U}$ chooses $x \in_R \mathbb{Z}_q$ as his private key $sk_\mathcal{U}$ and computes g^x as his public key $pk_\mathcal{U}$ and $H_2(g^x)^x$ as his key token $kt_\mathcal{U}$. $\mathcal{U}$'s key tuple is $(sk_\mathcal{U}, pk_\mathcal{U}, kt_\mathcal{U}) = (x, g^x, H_2(g^x)^x)$. $\mathcal{U}$ registers $pk_\mathcal{U}$ to the system manager.

Tag Computation. $\mathcal{U}$ has data $\mathcal{M} = (m_1, m_2, \ldots, m_n)$, each block k-bit long, and would like to store them in $\mathcal{S}$. $\mathcal{U}$ chooses data identifier $h_\mathcal{M} \in_R G$ and tag identifier seed $T_\mathcal{M} \in_R \{0,1\}^*$ for $\mathcal{M}$. $\mathcal{U}$ may have many different data. Thus, he needs to choose a unique data identifier for each of his data. Each block m_i is tagged to a homomorphic verifiable tag σ_i which is identified by $h_\mathcal{M}$ and tag identifier $T_\mathcal{M}||i$. $\mathcal{U}$ computes these homomorphic verifiable tags $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ for $\mathcal{M}$ as follows:

$$\sigma_i = [H_1(T_\mathcal{M}||i) h_\mathcal{M}^{m_i}]^{sk_\mathcal{U}} \quad , \text{for } 1 \le i \le n \ .$$

$\mathcal{U}$ uploads $(\mathcal{M}, h_\mathcal{M}, \Sigma)$ to $\mathcal{S}$, and holds $(h_\mathcal{M}, T_\mathcal{M})$ for identifying and verifying Σ.

Delegation. $\mathcal{V}$ gives his key token $kt_\mathcal{V}$ to $\mathcal{U}$ over a secure channel, and obtains $h_\mathcal{M}$ and $T_\mathcal{M}$ from $\mathcal{U}$. $\mathcal{U}$ uses $\mathcal{V}$'s public key $pk_\mathcal{V}$ to verify validity of $kt_\mathcal{V}$ by checking whether $\hat{e}(g, kt_\mathcal{V}) = \hat{e}(pk_\mathcal{V}, H_2(pk_\mathcal{V}))$. Then, $\mathcal{U}$ computes the delegation key

$$dk_{\mathcal{U}\to\mathcal{V}} = kt_\mathcal{V}^{1/sk_\mathcal{U}}$$

and gives it to $\mathcal{S}$. $\mathcal{S}$ uses $pk_\mathcal{U}$ and $pk_\mathcal{V}$ to verify validity of $dk_{\mathcal{U}\to\mathcal{V}}$ by checking whether $\hat{e}(pk_\mathcal{U}, dk_{\mathcal{U}\to\mathcal{V}}) = \hat{e}(pk_\mathcal{V}, H_2(pk_\mathcal{V}))$. To revoke $\mathcal{V}$, $\mathcal{U}$ commands $\mathcal{S}$ to remove $dk_{\mathcal{U}\to\mathcal{V}}$ from its storage directly.

Integrity Check. To check integrity of $\mathcal{M}$, $\mathcal{V}$ chooses coefficients $C = (c_1, c_2, \ldots, c_n) \in_R \mathbb{Z}_q^n$ and gives $\mathcal{S}$ the challenge

$$chal = (C,\ C',\ C'') = (C,\ h_{\mathcal{M}}^s,\ H_3(C)^s)\,, \quad \text{where } s \in_R \mathbb{Z}_q\ .$$

After receiving $chal$, $\mathcal{S}$ verifies it by checking whether $\hat{e}(C', H_3(C)) = \hat{e}(h_{\mathcal{M}}, C'')$. If so, $\mathcal{S}$ uses $(\mathcal{M}, \Sigma, dk_{\mathcal{U}\to\mathcal{V}}, chal)$ to generate a proof $pf_{chal,\mathcal{V}} = (\rho, V, V', V'', V''')$ and gives it to $\mathcal{V}$ as a response. $pf_{chal,\mathcal{V}}$ is computed as follows:

$$pf_{chal,\mathcal{V}} = \left(\hat{e}(\prod_{i=1}^{n}\sigma_i^{c_i},\ dk_{\mathcal{U}\to\mathcal{V}})^t,\ C'^{\sum_{i=1}^n c_i m_i},\ C''^{\sum_{i=1}^n c_i m_i}, H_2(pk_{\mathcal{V}})^t,\ g^t\right),\ \text{where } t \in_R \mathbb{Z}_q.$$

After receiving $pf_{chal,\mathcal{V}}$, $\mathcal{V}$ uses $(sk_{\mathcal{V}}, h_{\mathcal{M}}, T_{\mathcal{M}}, C, s)$ to verify $pf_{chal,\mathcal{V}}$ by checking whether

- $\rho^s = \hat{e}\left(\prod_{i=1}^n H_1(T_{\mathcal{M}}||i)^{sc_i} V,\ V''\right)^{sk_{\mathcal{V}}}$
- $\hat{e}\left(V,\ H_3(C)\right) = \hat{e}\left(h_{\mathcal{M}},\ V'\right)$
- $\hat{e}\left(V'',\ g\right) = \hat{e}\left(H_2(pk_{\mathcal{V}}),\ V'''\right)$

$\mathcal{V}$ can verify data integrity of $\mathcal{M}$ multiple times to achieve a desire security level.

Correctness. Although tag $\sigma = [H_1(T_{\mathcal{M}})h_{\mathcal{M}}^m]^{sk_{\mathcal{U}}}$ is called homomorphic verifiable in the literature, it is really not homomorphic. Instead, σ is combinably verifiable since we can combine multiple tags together and verify them at the same time. Nevertheless, we cannot obtain a tag for the combined data. For example, combing $\sigma_i = [H_1(T_{\mathcal{M}}||i)h_{\mathcal{M}}^{m_i}]^{sk_{\mathcal{U}}}$ and $\sigma_j = \left[H_1(T_{\mathcal{M}}||j)h_{\mathcal{M}}^{m_j}\right]^{sk_{\mathcal{U}}}$ together results in $\sigma' = \left[H_1(T_{\mathcal{M}}||i)H_1(T_{\mathcal{M}}||j)h_{\mathcal{M}}^{m_i+m_j}\right]^{sk_{\mathcal{U}}}$. Although we have $m_i + m_j$ in the exponent of $h_{\mathcal{M}}$, we don't have $H_1(T_{\mathcal{M}}||k)$ in the combined tag σ' for some k (treat $m_i + m_j = m_k$). The tag is unforgeable, proved in Sect. 4.1 (proof unforgeability implies tag unforgeability). It is hard to obtain $\sigma' = \left[H_1(T_{\mathcal{M}}||k)h_{\mathcal{M}}^{m_i+m_j}\right]^{sk_{\mathcal{U}}}$ without the knowledge of private key $sk_{\mathcal{U}}$.

In integrity check, $\mathcal{V}$ chooses coefficients $C = (c_1, c_2, \ldots, c_n)$ and then $\mathcal{S}$ combines stored tags $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ as $\prod_{i=1}^n \sigma_i^{c_i} = \left[\prod_{i=1}^n H_1(T_{\mathcal{M}}||i)^{c_i} \times h_{\mathcal{M}}^{\sum_{i=1}^n c_i m_i}\right]^{sk_{\mathcal{U}}}$ by C. If $\mathcal{S}$ deviates, the combination will not be identical to $\prod_{i=1}^n H_1(T_{\mathcal{M}}||i)^{c_i}$. Once $\mathcal{S}$ combines these tags correctly, we have $\sum_{i=1}^n c_i m_i$ in the exponent of $h_{\mathcal{M}}$. On the other hand, $\mathcal{S}$ has to use stored data $\mathcal{M} = (m_1, m_2, \ldots, m_n)$ to compute $V = C'^{\sum_{i=1}^n c_i m_i} = (h_{\mathcal{M}}^s)^{\sum_{i=1}^n c_i m_i}$ and $V' = C''^{\sum_{i=1}^n c_i m_i} = [H_3(C)^s]^{\sum_{i=1}^n c_i m_i}$. Our verification, $\rho^s = \hat{e}\left(\prod_{i=1}^n H_1(T_{\mathcal{M}}||i)^{sc_i} V, V''\right)^{sk_{\mathcal{V}}}$, checks whether ρ contains the correct combination $\prod_{i=1}^n H_1(T_{\mathcal{M}}||i)^{c_i}$ and whether V contains the same exponent

$\sum_{i=1}^{n} c_i m_i$, with respect to $h_{\mathcal{M}}^{s}$, as that in ρ, with respect to $h_{\mathcal{M}}$. If $\mathcal{S}$ passes this verification, he possesses $\mathcal{M}$.

Let's examine the verification equations. Assume that $pf_{chal,\mathcal{V}}$ is well-formed and $chal = (C, C', C'') = (C, h_{\mathcal{M}}^{s}, H_3(C)^s)$, that is,

$$\rho = \hat{e}(\prod_{i=1}^{n} \sigma_i^{c_i}, dk_{\mathcal{U}\to\mathcal{V}})^t \tag{1}$$

$$V = C'^{\sum_{i=1}^{n} c_i m_i} = h_{\mathcal{M}}^{s\sum_{i=1}^{n} c_i m_i} \tag{2}$$

$$V' = C''^{\sum_{i=1}^{n} c_i m_i} = H_3(C)^{s\sum_{i=1}^{n} c_i m_i} \tag{3}$$

$$V'' = H_2(pk_{\mathcal{V}})^t \tag{4}$$

$$V''' = g^t \tag{5}$$

We have:

– $\rho^s = \hat{e}(\prod_{i=1}^{n} H_1(T_{\mathcal{M}}||i)^{sc_i} V, V'')^{sk_{\mathcal{V}}}$ by (1), (2), and (4)

$$\begin{aligned}
\rho^s &= \hat{e}(\prod_{i=1}^{n} \sigma_i^{c_i}, dk_{\mathcal{U}\to\mathcal{V}})^{ts} \\
&= \hat{e}(\prod_{i=1}^{n} (H_1(T_{\mathcal{M}}||i) h_{\mathcal{M}}^{m_i})^{sk_{\mathcal{U}} c_i}, H_2(pk_{\mathcal{V}})^{sk_{\mathcal{V}}/sk_{\mathcal{U}}})^{ts} \\
&= \hat{e}(\prod_{i=1}^{n} (H_1(T_{\mathcal{M}}||i) h_{\mathcal{M}}^{m_i})^{sc_i}, H_2(pk_{\mathcal{V}})^{sk_{\mathcal{V}}})^{t} \\
&= \hat{e}(\prod_{i=1}^{n} H_1(T_{\mathcal{M}}||i)^{sc_i} h_{\mathcal{M}}^{s\sum_{i=1}^{n} c_i m_i}, H_2(pk_{\mathcal{V}})^{t})^{sk_{\mathcal{V}}} \\
&= \hat{e}(\prod_{i=1}^{n} H_1(T_{\mathcal{M}}||i)^{sc_i} V, V'')^{sk_{\mathcal{V}}}
\end{aligned}$$

– $\hat{e}(V, H_3(C)) = \hat{e}(h_{\mathcal{M}}, V')$ by (2) and (3)

$$\begin{aligned}
\hat{e}(V, H_3(C)) &= \hat{e}(h_{\mathcal{M}}^{s\sum_{i=1}^{n} c_i m_i}, H_3(C)) \\
&= \hat{e}(h_{\mathcal{M}}, H_3(C)^{s\sum_{i=1}^{n} c_i m_i}) \\
&= \hat{e}(h_{\mathcal{M}}, V')
\end{aligned}$$

– $\hat{e}(V'', g) = \hat{e}(H_2(pk_{\mathcal{V}}), V''')$ by (4) and (5)

$$\begin{aligned}
\hat{e}(V'', g) &= \hat{e}(H_2(pk_{\mathcal{V}})^t, g) \\
&= \hat{e}(H_2(pk_{\mathcal{V}}), g^t) \\
&= \hat{e}(H_2(pk_{\mathcal{V}}), V''')
\end{aligned}$$

3.1 Performance

We analyze performance of our construction in three aspects: the computation cost of each algorithm, the storage cost of each party, and the communication cost of each phase. Table 1 shows the computation cost of each algorithm.[1] We measure the numbers of additions in $\mathbb{Z}_q$, multiplications in G, scalar exponentiations in G, hashes, and pairings.

Table 1. Computation cost of each algorithm

Algorithm	Addition	Multiplication	Scalar Exponentiation	Hash	Pairing
Setup	0	0	0	0	0
KeyGen	0	0	2	1	0
TagGen	0	n	$2n$	n	0
GenDK	0	0	1	1	2
VrfyDK	0	0	0	1	2
GenChal	0	0	2	1	0
GenProof	$n-1$	$2n-1$	$n+5$	2	3
VrfyProof	0	n	$n+3$	$n+2$	5

– n is the number of data blocks.

Table 2 shows the storage cost of each party. User $\mathcal{U}$ stores his key tuple $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, kt_{\mathcal{U}})$, data identifier $h_{\mathcal{M}}$, and tag identifier seed $T_{\mathcal{M}}$. Delegated verifier $\mathcal{V}$ stores his key tuple $(sk_{\mathcal{V}}, pk_{\mathcal{V}}, kt_{\mathcal{V}})$, data identifier $h_{\mathcal{M}}$, and tag identifier seed $T_{\mathcal{M}}$. Storage server $\mathcal{S}$ stores $\mathcal{U}$'s data $\mathcal{M}$, data identifier $h_{\mathcal{M}}$, tags Σ, and the delegation keys. Table 3 shows the communication cost of each phase. In setup phase, $\mathcal{U}$ uploads $\mathcal{M}$, Σ, and $h_{\mathcal{M}}$ to $\mathcal{S}$. In delegation phase, $\mathcal{V}$ gives $\mathcal{U}$ his key token $kt_{\mathcal{V}}$. $\mathcal{U}$ gives $h_{\mathcal{M}}$ and $T_{\mathcal{M}}$ to $\mathcal{V}$, and gives the delegation key $dk_{\mathcal{U}\to\mathcal{V}}$ to $\mathcal{S}$. In integrity check phase, $\mathcal{V}$ gives $\mathcal{S}$ the challenge $chal$,[2] and $\mathcal{S}$ gives $\mathcal{V}$ the proof $pf_{chal,\mathcal{V}}$.

[1] To achieve better performance, one can choose binary coefficient $c_i \in \{0,1\}$, $1 \le i \le n$, to reduce computation on multiplications and scalar exponentiations. Thus, in algorithm GenProof, we don't need to do scalar exponentiations on σ_i to compute $\sigma_i^{c_i}$, and multiplications on m_i to compute $c_i m_i$. Thus, it reduces the computation cost from $2n-1$ multiplications in G and $n+5$ scalar exponentiations in G to $n-1$ multiplications in G and 5 scalar exponentiations in G for GenProof. And in algorithm VrfyProof, we don't really do scalar exponentiations on $H_1(T_{\mathcal{M}}||i)$ to compute $H_1(T_{\mathcal{M}}||i)^{c_i}$, either. Thus, it reduces the computation cost from $n+3$ scalar exponentiations in G to 3 scalar exponentiations in G for VrfyProof.

[2] To reduce the communication cost on transmitting $chal$, one can choose a random seed c of size ℓ' for computing coefficients $c_i = H(c,i)$, $1 \le i \le n$, and send c only. Thus, it reduces the communication cost from $nk + 6p + p_T$ bits to $\ell' + 6p + p_T$ bits in integrity check phase.

Table 2. Storage cost of each party

Party	Storage Cost (Bit)
User	$k + 3p + \ell$
Delegated Verifier	$k + 3p + \ell$
Storage Server	$nk + (1 + n + v)p$

- k is the security parameter
- p is the size of an element in G
- l is the length of tag identifier seed $T_{\mathcal{M}}$
- n is the number of data blocks
- v is the number of delegated verifiers

Table 3. Communication cost of each phase

Phase	Communication Cost (Bit)
Setup	$\mathcal{U} \rightarrow \mathcal{S} : nk + p + np$
Delegation	$\mathcal{V} \leftrightarrow \mathcal{U} : 2p + \ell$ $\mathcal{U} \rightarrow \mathcal{S} : p$
Integrity Check	$\mathcal{V} \leftrightarrow \mathcal{S} : nk + 6p + p_T$

- k is the security parameter
- p is the size of an element in G
- p_T is the size of an element in G_T
- l is the length of tag identifier $h_{\mathcal{M}}$
- n is the number of data blocks

4 Security Analysis

The security requirements of a delegable PDP model consists of **proof unforgeability**, **proof indistinguishability**, and **delegation key unforgeability**. We introduce the security games in the rest subsections and prove that our construction satisfies these security requirements in the random oracle model.

4.1 Proof Unforgeability

This game models the notion that a storage server cannot modify stored data without being detected by verifiers. In this game, the challenger $\mathcal{C}$ plays the role of the verifier and the adversary $\mathcal{A}$ plays the role of the storage server. $\mathcal{A}$ is given the access right to oracles $\mathcal{O}_{\mathsf{Tag}}$ and $\mathcal{O}_{\mathsf{DK}}$. $\mathcal{A}$ chooses data adaptively and obtains corresponding tags. Once $\mathcal{A}$ decides the target data $\mathcal{M}^*$, he modifies $\mathcal{M}^*$ to $\mathcal{M}'$ such that $\mathcal{M}' \neq \mathcal{M}^*$ and receives a challenge from $\mathcal{C}$. If $\mathcal{A}$ returns a proof that passes the verification algorithm, he wins this game.

The proof unforgeability game $\mathsf{Game}^{\mathsf{PF-UF}}$ is as follows:

Setup. $\mathcal{C}$ generates public parameter π, user $\mathcal{U}$'s key tuple $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, kt_{\mathcal{U}})$, delegated verifier $\mathcal{V}$'s key tuple $(sk_{\mathcal{V}}, pk_{\mathcal{V}}, kt_{\mathcal{V}})$, and the delegation key $dk_{\mathcal{U}\rightarrow\mathcal{V}}$. $\mathcal{C}$ forwards $(\pi, pk_{\mathcal{U}}, pk_{\mathcal{V}}, dk_{\mathcal{U}\rightarrow\mathcal{V}})$ to $\mathcal{A}$.

Query. $\mathcal{A}$ queries oracle $\mathcal{O}_{\mathsf{Tag}}$ and oracle $\mathcal{O}_{\mathsf{DK}}$ to obtain tags and delegation keys.

- $\mathcal{O}_{\mathsf{Tag}}$: $\mathcal{A}$ chooses data $\mathcal{M}$ and obtains tags Σ, data identifier $h_{\mathcal{M}}$, and tag identifier seed $T_{\mathcal{M}}$ for $\mathcal{M}$.
- $\mathcal{O}_{\mathsf{DK}}$: $\mathcal{A}$ chooses a user $\mathcal{U}'$ and obtains the delegation key $dk_{\mathcal{U}\to\mathcal{U}'}$.

Challenge. After the query phase, $\mathcal{A}$ indicates which $\mathcal{O}_{\mathsf{Tag}}$-oracle query is the target, denoted as $(\mathcal{M}^*, \Sigma^*, h_{\mathcal{M}^*}, T_{\mathcal{M}^*})$, and modifies $\mathcal{M}^* = (m_1^*, m_2^*, \ldots, m_n^*)$ to $\mathcal{M}' = (m_1', m_2', \ldots, m_n')$ such that $\mathcal{M}' \neq \mathcal{M}^*$ $(\exists i, m_i' \neq m_i^*)$. $\mathcal{C}$ gives challenge $chal = (C, h_{\mathcal{M}^*}^s, H_3(C)^s)$.
Answer. $\mathcal{A}$ returns proof $pf_{chal,\mathcal{V}}$ by using $\mathcal{M}'$. $\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{PF-UF}}$ if $\mathsf{VrfyProof}(\pi, chal, pf_{chal,\mathcal{V}}, h_{\mathcal{M}^*}, T_{\mathcal{M}^*}, sk_{\mathcal{V}}) = true$ and $\mathcal{M}' \neq \mathcal{M}^*$ $(\exists i, m_i' \neq m_i^*)$. The advantage $Adv_{\mathcal{A}}^{\mathsf{Pf-UF}}$ is defined as $\Pr[\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{PF-UF}}]$.

We show that our scheme is proof unforgeable under the *truncated* 1-BDHE assumption and the KEA1.

Theorem 1. *If the truncated BDHE problem is* $(t, \epsilon, 1)$*-secure, the above scheme is* $(t - q_1 t_1 - q_2 t_2 - q_T t_T - q_K t_K - 2t_{\overline{A}}, \frac{2^\ell}{2^\ell - (q_1+q_T)q_T} \frac{2^k}{2^k-1}\epsilon)$ *proof unforgeable in the random oracle model, where hash functions* H_1 *and* H_2 *are modeled as random oracles* $\mathcal{O}_{\mathsf{H}_1}$ *and* $\mathcal{O}_{\mathsf{H}_2}$*,* (q_1, q_2, q_T, q_K) *are the numbers of times that an adversary queries* $(\mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Tag}}, \mathcal{O}_{\mathsf{DK}})$*-oracles,* (t_1, t_2, t_T, t_K) *are the time used by* $(\mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Tag}}, \mathcal{O}_{\mathsf{DK}})$*-oracles to respond an oracle query,* $t_{\overline{A}}$ *is the time used by the KEA1 extractor* $\overline{A}$ *to extract an exponent,* k *is the security parameter, and* ℓ *is the bit-length of a tag identifier seed.*

Proof. Let $\mathcal{A}$ be a probabilistic black-box adversary who wins the proof unforgeability game $\mathsf{Game}^{\mathsf{PF-UF}}$ with advantage ϵ' in time t'. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve the *truncated* 1-BDHE problem as follows:

Setup. Given an instance (g, g^α, g') of the *truncated* 1-BDHE problem, $\mathcal{B}$ sets the public parameter $\pi = (q, G, g, G_T, g_T, \hat{e}, H_3)$, user $\mathcal{U}$'s key tuple $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, kt_{\mathcal{U}}) = (\alpha u, g^{\alpha u}, \mathsf{H}_2(g^{\alpha u})^{\alpha u})$, where $\mathsf{H}_2(g^{\alpha u}) = g^{\alpha u'}$ and $u, u' \in_R \mathbb{Z}_q$, and delegated verifier $\mathcal{V}$'s key tuple $(sk_{\mathcal{V}}, pk_{\mathcal{V}}, kt_{\mathcal{V}}) = (\alpha v, g^{\alpha v}, \mathsf{H}_2(g^{\alpha v})^{\alpha v})$, where $\mathsf{H}_2(g^\alpha v) = g^{\alpha v'}$ and $v, v' \in_R \mathbb{Z}_q$. Then $\mathcal{B}$ computes the delegation key

$$dk_{\mathcal{U}\to\mathcal{V}} = \mathsf{H}_2(pk_{\mathcal{V}})^{sk_{\mathcal{V}}/sk_{\mathcal{U}}} = (g^{\alpha v'})^{\alpha v/\alpha u} = g^{\alpha v v'/u}$$

and invokes $\mathcal{A}$ as a subroutine: $\mathcal{A}^{\mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Tag}}, \mathcal{O}_{\mathsf{DK}}}(\pi, pk_{\mathcal{U}}, pk_{\mathcal{V}}, dk_{\mathcal{U}\to\mathcal{V}})$.

Query. $\mathcal{A}$ can query oracles $\mathcal{O}_{\mathsf{H}_1}$, $\mathcal{O}_{\mathsf{H}_2}$, $\mathcal{O}_{\mathsf{Tag}}$, and $\mathcal{O}_{\mathsf{DK}}$ during his execution. $\mathcal{B}$ handles these oracles as follows:

- $\mathcal{O}_{\mathsf{H}_1}$. $\mathcal{B}$ maintains a table $\mathcal{T}_{\mathsf{H}_1} = \{(x, \mathsf{H}_1(x), r)\}$ to look up the $\mathcal{O}_{\mathsf{H}_1}$-query records. $\mathcal{B}$ takes $x \in \{0,1\}^*$ as input and outputs y if record $(x, y, *)$ exists in $\mathcal{T}_{\mathsf{H}_1}$. Otherwise, $\mathcal{B}$ outputs $\mathsf{H}_1(x) = g^r$ and inserts (x, g^r, r) into $\mathcal{T}_{\mathsf{H}_1}$, where $r \in_R \mathbb{Z}_q$.

- $\mathcal{O}_{\mathsf{H}_2}$. $\mathcal{B}$ maintains a table $\mathcal{T}_{\mathsf{H}_2} = \{(g^x, \mathsf{H}_2(g^x), r)\}$ to look up the $\mathcal{O}_{\mathsf{H}_2}$-query records. $\mathcal{B}$ takes g^x as input and outputs y if record $(g^x, y, *)$ exists in $\mathcal{T}_{\mathsf{H}_2}$. Otherwise, $\mathcal{B}$ outputs $\mathsf{H}_2(g^x) = g^{\alpha r}$ and inserts $(g^x, g^{\alpha r}, r)$ into $\mathcal{T}_{\mathsf{H}_2}$, where $r \in_R \mathbb{Z}_q$.
- $\mathcal{O}_{\mathsf{Tag}}$. $\mathcal{B}$ maintains a table $\mathcal{T}_{\mathsf{Tag}} = \{(\mathcal{M}, h_{\mathcal{M}}, r, T_{\mathcal{M}}, \Sigma)\}$ to look up the $\mathcal{O}_{\mathsf{Tag}}$-query records. $\mathcal{B}$ takes $\mathcal{M} = (m_1, m_2, \ldots, m_n)$ as input, sets data identifier $h_{\mathcal{M}} = g'^r$, where $r \in_R Z_q$, and chooses tag identifier $T_{\mathcal{M}} \in_R \{0,1\}^\ell$ randomly. For $1 \le i \le n$, if $T_{\mathcal{M}}||i$ has been queried to oracle $\mathcal{O}_{\mathsf{H}_1}$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ inserts each $(T_{\mathcal{M}}||i, g^{r_i}/h_{\mathcal{M}}^{m_i}, r_i)$ into table $\mathcal{T}_{\mathsf{H}_1}$, where $r_i \in_R \mathbb{Z}_q$, outputs $(h_{\mathcal{M}}, T_{\mathcal{M}}, \Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n))$, where

$$\sigma_i = (\mathsf{H}_1(T_{\mathcal{M}}||i)h_{\mathcal{M}}^{m_i})^{sk_{\mathcal{U}}} = ((g^{r_i}/h_{\mathcal{M}}^{m_i})h_{\mathcal{M}}^{m_i})^{\alpha u} = g^{\alpha u r_i} \ ,$$

  and inserts $(\mathcal{M}, h_{\mathcal{M}}, r, T_{\mathcal{M}}, \Sigma)$ into $\mathcal{T}_{\mathsf{Tag}}$.
- $\mathcal{O}_{\mathsf{DK}}$. $\mathcal{B}$ takes user $\mathcal{U}'$'s public key $pk_{\mathcal{U}'} = g^x$ and key token $kt_{\mathcal{U}'}$ as input, looks up whether record $(g^x, *, *)$ exists in table $\mathcal{T}_{\mathsf{H}_2}$, and checks whether $\hat{e}(g, kt_{\mathcal{U}'}) = \hat{e}(g^x, \mathsf{H}_2(g^x))$. If not, $\mathcal{B}$ rejects. Otherwise, $\mathcal{B}$ outputs the delegation key

$$dk_{\mathcal{U}\to\mathcal{U}'} = \mathsf{H}_2(g^x)^{sk_{\mathcal{U}'}/sk_{\mathcal{U}}} = (g^{\alpha r})^{x/\alpha u} = g^{xr/u} \ .$$

Challenge. After the query phase, $\mathcal{A}$ indicates which $\mathcal{O}_{\mathsf{Tag}}$-query is the target and modifies data to $\mathcal{M}'$. $\mathcal{B}$ looks up the corresponding record in table $\mathcal{T}_{\mathsf{Tag}}$, denoted as $(\mathcal{M}^* = (m_1^*, m_2^*, \ldots, m_n^*), h_{\mathcal{M}^*} = g'^{r^*}, r^*, T_{\mathcal{M}^*}, \Sigma^* = (\sigma_1^*, \sigma_2^*, \ldots, \sigma_n^*))$, and returns challenge $chal = (C = (c_1, c_2, \ldots, c_n), h_{\mathcal{M}^*}^s, H_3(C)^s)$, where $c_i, s \in_R \mathbb{Z}_q$.

Answer. $\mathcal{A}$ returns integrity proof $pf_{chal,\mathcal{V}} = (\rho, V, V', V'', V''')$ using $\mathcal{M}'$. If $\mathcal{M}' \neq \mathcal{M}^*$, we have $V \neq h_{\mathcal{M}^*}^{s\sum_{i=1}^n c_i m_i^*}$ except for a negligible probability. That is, $\Pr[\mathcal{A}$ guesses $\sum_{i=1}^n c_i m_i^* : c_i \in_R \mathbb{Z}_q$ and $\mathcal{M}' \neq \mathcal{M}^*] = \frac{1}{q}$. Otherwise, $\mathcal{A}$ knows the knowledge of $\mathcal{M}^*$[3]. Thus, if $pf_{chal,\mathcal{V}}$ can pass the verification procedure and $\mathcal{M}' \neq \mathcal{M}^*$, we have:

$$\rho^s = \hat{e}(\prod_{i=1}^n \mathsf{H}_1(T_{\mathcal{M}^*}||i)^{sc_i} V, V'')^{sk_{\mathcal{V}}} \tag{6}$$

$$\hat{e}(V, H_3(C)) = \hat{e}(h_{\mathcal{M}^*}, V') \tag{7}$$

$$\hat{e}(V'', g) = \hat{e}(\mathsf{H}_2(pk_{\mathcal{V}}), V''') \tag{8}$$

$$V \neq h_{\mathcal{M}^*}^{s\sum_{i=1}^n c_i m_i^*} \tag{9}$$

[3] $\mathcal{B}$ can extract $\mathcal{M}^*$ by choosing a sequence of linearly independent coefficients adaptively until collecting n valid responses from $\mathcal{A}$. These n linearly independent vectors C_i, $1 \le i \le n$, form an $n \times n$ non-singular matrix $[C_1\ C_2\ \ldots\ C_n]^\mathsf{T} = [c_{i,j}]_{1\le i\le n,\ 1\le j\le n}$. $\mathcal{B}$ uses the KEA1 extractor $\overline{\mathcal{A}}$ to extract the n constant terms $\sum_{j=1}^n c_{i,j} m_j^*$ from V_i and V_i', $1 \le i \le n$, and solves the system of linear equations to obtain $\mathcal{M}^*$.

$\mathcal{B}$ can compute $\hat{e}(g,g')^{\alpha^2}$ as follows:

1. Since (7) holds, we have $h_{\mathcal{M}^*}^{\Delta} = H_3(C)$ and $V^{\Delta} = V'$ for some $\Delta \in_R \mathbb{Z}_q$. Thus, $\mathcal{B}$ can use the $\mathsf{KEA1}$ extractor $\overline{\mathcal{A}}$ to extract $m' = \overline{\mathcal{A}}(h_{\mathcal{M}^*}^{s}, H_3(C)^s, V, V')$ such that $V = (h_{\mathcal{M}^*}^{s})^{m'}$. Since (9) holds, we have $m' \neq \sum_{i=1}^{n} c_i m_i^*$.
2. Similarly, since (8) holds, $\mathcal{B}$ can use the $\mathsf{KEA1}$ extractor $\overline{\mathcal{A}}$ to extract $t = \overline{\mathcal{A}}(\mathsf{H}_2(pk_{\mathcal{V}}), g, V'', V''')$ such that $V'' = \mathsf{H}_2(pk_{\mathcal{V}})^t$.
3. After knowing m' and t, since (6) and $m' \neq \sum_{i=1}^{n} c_i m_i^*$ hold, $\mathcal{B}$ can compute $\hat{e}(g,g')^{\alpha^2}$ as follows:

$$
\begin{aligned}
& \rho^s = \hat{e}(\prod_{i=1}^{n} \mathsf{H}_1(T_{\mathcal{M}^*}||i)^{sc_i} V, V'')^{sk_{\mathcal{V}}} \\
\Rightarrow\ & \rho^s = \hat{e}(\prod_{i=1}^{n} (g^{r_i^*}/h_{\mathcal{M}^*}^{m_i^*})^{sc_i} h_{\mathcal{M}^*}^{sm'}, \mathsf{H}_2(pk_{\mathcal{V}})^t)^{\alpha v} \\
\Rightarrow\ & \rho^s = \hat{e}(\prod_{i=1}^{n} (g^{r_i^*}/g'^{r^* m_i^*})^{sc_i} g'^{r^* sm'}, g^{\alpha v' t})^{\alpha v} \\
\Rightarrow\ & \rho = \hat{e}(\prod_{i=1}^{n} (g^{c_i r_i^*}/g'^{r^* c_i m_i^*}) g'^{r^* m'}, g^{\alpha v' t})^{\alpha v} \\
\Rightarrow\ & \rho = \hat{e}(g^{\sum_{i=1}^{n} c_i r_i^*} g'^{r^*(m' - \sum_{i=1}^{n} c_i m_i^*)}, g^{\alpha v' t})^{\alpha v} \\
\Rightarrow\ & \hat{e}(g'^{r^*(m' - \sum_{i=1}^{n} c_i m_i^*)}, g^{\alpha v' t})^{\alpha v} = \frac{\rho}{\hat{e}(g^{\alpha \sum_{i=1}^{n} c_i r_i^*}, g^{\alpha v' t})^v} \\
\Rightarrow\ & \hat{e}(g,g')^{\alpha^2} = \left(\frac{\rho}{\hat{e}(g^{\alpha \sum_{i=1}^{n} c_i r_i^*}, g^{\alpha v' t})^v}\right)^{1/vv'tr^*(m' - \sum_{i=1}^{n} c_i m_i^*)}
\end{aligned}
$$

$\mathcal{B}$ aborts on handling oracle $\mathcal{O}_{\mathsf{Tag}}$ if tag identifier $T_{\mathcal{M}}$ has been queried to oracle $\mathcal{O}_{\mathsf{H}_1}$. That is, record $(T_{\mathcal{M}}, *, *)$ exists in table $\mathcal{T}_{\mathsf{H}_1}$. For each $\mathcal{O}_{\mathsf{Tag}}$-query, we have $\Pr[(T_{\mathcal{M}}, *, *) \in \mathcal{T}_{\mathsf{H}_1}] = |\mathcal{T}_{\mathsf{H}_1}|/2^{|T_{\mathcal{M}}|} \leq (q_1 + q_T)/2^{\ell}$. Take the union bound on the q_T $\mathcal{O}_{\mathsf{Tag}}$-queries, we have $\Pr[\mathcal{B}\text{ aborts}] \leq (q_1 + q_T)q_T/2^{\ell}$. Moreover, $\mathcal{B}$ loses a negligible portion $\frac{1}{q} = \frac{1}{2^k}$ that $\mathcal{M}' \neq \mathcal{M}^*$ but $V = h_{\mathcal{M}^*}^{s\sum_{i=1}^{n} c_i m_i^*}$. Therefore, the reduced advantage is $\epsilon = (1 - \frac{(q_1+q_T)q_T}{2^{\ell}})(1 - \frac{1}{2^k})\epsilon'$. Besides of handling $(\mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Tag}}, \mathcal{O}_{\mathsf{DK}})$-oracles, $\mathcal{B}$ uses the $\mathsf{KEA1}$ extractor $\overline{\mathcal{A}}$ two times to extract two exponents. Therefore, the reduced time is $t = t' + q_1 t_1 + q_2 t_2 + q_T t_T + q_K t_K + 2t_{\overline{\mathcal{A}}}$. By choosing appropriate $q_1, q_2, q_T, q_K, \ell \in \mathsf{Poly}(k)$, we have $((q_1 + q_T)q_T/2^{\ell}, 1/2^k) \in \mathsf{negl}(k)^2$ and $q_1 t_1 + q_2 t_2 + q_T t_T + q_K t_K \in \mathsf{Poly}(k)$. □

In the proof unforgeability game $\mathsf{Game}^{\mathsf{PF-UF}}$, the challenge *chal* is chosen by the challenger $\mathcal{C}$. $\mathsf{Game}^{\mathsf{PF-UF}}$ can be adapted for existential unforgeability by letting adversary $\mathcal{A}$ choose *chal* by himself. In our security proof, this modification only needs one more execution of the $\mathsf{KEA1}$ extractor to know $\mathcal{A}$'s choice for the randomness s of *chal*.

4.2 Proof Indistinguishability

This game models the notion that a third party without being authorized cannot verify validity of data integrity proofs even if he eavesdrops network communications after the setup phase. In this game, the challenger $\mathcal{C}$ plays the role of the storage server, and the adversary $\mathcal{A}$ plays the role of the third-party user. $\mathcal{A}$ is given access right to oracle $\mathcal{O}_{\mathsf{Proof}}$. $\mathcal{A}$ is trained with valid proofs and tries to verify validity of the target proof. If $\mathcal{A}$ answers validity correctly, he wins this game.

The proof indistinguishability game $\mathsf{Game}^{\mathsf{PF-IND}}$ is as follows:

Setup. $\mathcal{C}$ generates public parameter π, user $\mathcal{U}$'s key tuple $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, kt_{\mathcal{U}})$, delegated verifier $\mathcal{V}$'s key tuple $(sk_{\mathcal{V}}, pk_{\mathcal{V}}, kt_{\mathcal{V}})$, delegation key $dk_{\mathcal{U}\to\mathcal{V}}$, data $\mathcal{M} = (m_1, m_2, \ldots, m_n)$, tags $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, data identifier $h_{\mathcal{M}}$, and tag identifier seed $T_{\mathcal{M}}$ for $\mathcal{M}$. $\mathcal{C}$ forwards $(\pi, pk_{\mathcal{U}}, pk_{\mathcal{V}}, kt_{\mathcal{V}}, dk_{\mathcal{U}\to\mathcal{V}}, h_{\mathcal{M}}, T_{\mathcal{M}})$ to $\mathcal{A}$.

Query-1. $\mathcal{A}$ queries oracle $\mathcal{O}_{\mathsf{Proof}}$ to obtain samples of valid proofs.

- $\mathcal{O}_{\mathsf{Proof}}$: $\mathcal{A}$ chooses challenge $chal$ and obtains a valid proof $pf_{chal,\mathcal{V}}$ for $(\mathcal{M}, chal)$.

Challenge. Same as the query-1 phase except that validity of the returned proof $pf^{*}_{chal,\mathcal{V}}$ depends on an uniform bit b. If $b = 1$, the $\mathcal{C}$ returns a valid proof. Otherwise, $\mathcal{C}$ returns an invalid proof.

Query-2. Same as the query-1 phase.

Answer. $\mathcal{A}$ answers b' for the challenged proof $pf^{*}_{chal,\mathcal{V}}$. $\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{PF-IND}}$ if $b' = b$. The advantage $Adv^{\mathsf{Prf-IND}}_{\mathcal{A}}$ is defined as $\left|\Pr[\mathcal{A} \text{ wins } \mathsf{Game}^{\mathsf{PF-IND}}] - \frac{1}{2}\right|$.

We show that our scheme is proof indistinguishable under the *truncated decisional* 1-BDHE assumption.

Theorem 2. *If the truncated decisional BDHE problem is $(t, \epsilon, 1)$-secure, the above scheme is $(t - q_1t_1 - q_2t_2 - q_Pt_P, 2\epsilon)$ proof indistinguishable in the random oracle model, where hash functions H_1 and H_2 are modeled as random oracles $\mathcal{O}_{\mathsf{H}_1}$ and $\mathcal{O}_{\mathsf{H}_2}$, (q_1, q_2, q_P) are the numbers of times that an adversary queries $(\mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Proof}})$-oracles, and (t_1, t_2, t_P) are the time used by $(\mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Proof}})$-oracles to respond an oracle query.*

Proof. Let $\mathcal{A}$ be a probabilistic black-box adversary who wins the proof indistinguishability game $\mathsf{Game}^{\mathsf{PF-IND}}$ with advantage ϵ' in time t'. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve the *truncated decisional* 1-BDHE problem as follows:

Setup. Given an instance (g, g^{α}, g', Z) of the *truncated decision* 1-BDHE problem, $\mathcal{B}$ sets the public parameter $\pi = (q, G, g, G_T, g_T, \hat{e}, H_3)$, user $\mathcal{U}$'s key tuple $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, kt_{\mathcal{U}}) = (\alpha u, g^{\alpha u}, \mathsf{H}_2(g^{\alpha u})^{\alpha u})$, where $\mathsf{H}_2(g^{\alpha u}) = g^{u'}$ and $u, u' \in_R \mathbb{Z}_q$, delegated verifier $\mathcal{V}$'s key tuple $(sk_{\mathcal{V}}, pk_{\mathcal{V}}, kt_{\mathcal{V}}) = (\alpha v, g^{\alpha v}, \mathsf{H}_2(g^{\alpha v})^{\alpha v})$, where $\mathsf{H}_2(g^{\alpha v}) = g^{v'}$ and $v, v' \in_R \mathbb{Z}_q$, and the delegation key $dk_{\mathcal{U}\to\mathcal{V}} = kt_{\mathcal{V}}^{1/sk_{\mathcal{U}}} =$

$g^{vv'/u}$. Then $\mathcal{B}$ chooses data $\mathcal{M} = (m_1, m_2, \ldots, m_n)$, sets data identifier $h_{\mathcal{M}} = g'^r$, where $r \in_R \mathbb{Z}_q$, and chooses tag identifier seed $T_{\mathcal{M}}$. For $1 \leq i \leq n$, $\mathcal{B}$ sets $\mathsf{H}_1(T_{\mathcal{M}}||i) = g^{r_i}$, where $r_i \in_R \mathbb{Z}_q$. Then $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine: $\mathcal{A}^{\mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Proof}}}(\pi, pk_{\mathcal{U}}, pk_{\mathcal{V}}, kt_{\mathcal{V}}, dk_{\mathcal{U}\to\mathcal{V}}, h_{\mathcal{M}}, T_{\mathcal{M}})$.

Query-1. $\mathcal{A}$ can query oracles $\mathcal{O}_{\mathsf{H}_1}$, $\mathcal{O}_{\mathsf{H}_2}$, and $\mathcal{O}_{\mathsf{Proof}}$ during his execution. $\mathcal{B}$ handles these oracles as follows:

- $\mathcal{O}_{\mathsf{H}_1}$. $\mathcal{B}$ maintains a table $\mathcal{T}_{\mathsf{H}_1} = \{(x, \mathsf{H}_1(x), r)\}$ to look up the $\mathcal{O}_{\mathsf{H}_1}$-query records. $\mathcal{B}$ takes $x \in \{0,1\}^*$ as input and outputs y if record $(x, y, *)$ exists in $\mathcal{T}_{\mathsf{H}_1}$. Otherwise, $\mathcal{B}$ outputs $\mathsf{H}_1(x) = g^r$ and inserts (x, g^r, r) into $\mathcal{T}_{\mathsf{H}_1}$, where $r \in_R \mathbb{Z}_q$.
- $\mathcal{O}_{\mathsf{H}_2}$. $\mathcal{B}$ maintains a table $\mathcal{T}_{\mathsf{H}_2} = \{(g^x, \mathsf{H}_2(g^x), r)\}$ to look up the $\mathcal{O}_{\mathsf{H}_2}$-query records. $\mathcal{B}$ takes g^x as input and outputs y if record $(g^x, y, *)$ exists in $\mathcal{T}_{\mathsf{H}_2}$. Otherwise, $\mathcal{B}$ outputs $\mathsf{H}_2(g^x) = g^r$ and inserts (g^x, g^r, r) into $\mathcal{T}_{\mathsf{H}_2}$, where $r \in_R \mathbb{Z}_q$.
- $\mathcal{O}_{\mathsf{Proof}}$. $\mathcal{B}$ takes challenge $chal = (C = (c_1, c_2, \ldots, c_n), C', C'')$ as input and checks whether $\hat{e}(C', H_3(C)) = \hat{e}(h_{\mathcal{M}}, C'')$. If not, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ outputs a valid proof $pf_{chal,\mathcal{V}} = (\rho, V, V', V'', V''')$ as below: Let $t \in_R \mathbb{Z}_q$.

$$\begin{aligned}
\rho &= \hat{e}(\prod_{i=1}^{n} \sigma_i^{c_i}, dk_{\mathcal{U}\to\mathcal{V}})^t \\
&= \hat{e}(\prod_{i=1}^{n} (\mathsf{H}_1(T_{\mathcal{M}}||i) h_{\mathcal{M}}^{m_i})^{sk_{\mathcal{U}} c_i}, \mathsf{H}_2(pk_{\mathcal{V}})^{sk_{\mathcal{V}}/sk_{\mathcal{U}}})^t \\
&= \hat{e}(\prod_{i=1}^{n} (g^{r_i} g'^{r m_i})^{c_i}, g^{v'\alpha v})^t \\
&= \hat{e}(g^{\sum_{i=1}^{n} c_i r_i} g'^{r \sum_{i=1}^{n} c_i m_i}, g^{\alpha})^{vv't} \ ,
\end{aligned}$$

$$V = C'^{\sum_{i=1}^{n} c_i m_i} \ , \ V' = C''^{\sum_{i=1}^{n} c_i m_i} \ , \ V'' = g^{v't} \ , \text{ and } V''' = g^t \ .$$

Challenge. After the query-1 phase, $\mathcal{A}$ chooses challenge $chal = (C = (c_1, c_2, \ldots, c_n), C', C'')$, and $\mathcal{B}$ checks whether $\hat{e}(C', H_3(C)) = \hat{e}(h_{\mathcal{M}}, C'')$. If not, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ outputs a proof $pf_{\mathcal{V}}^* = (\rho^*, V, V', V'', V''')$ as follows, where $pf_{\mathcal{V}}^*$ is valid if $Z = \hat{e}(g, g')^{\alpha^2}$. Let $t = \alpha$.

$$\begin{aligned}
\rho^* &= \hat{e}(g^{\sum_{i=1}^{n} c_i r_i} g'^{r \sum_{i=1}^{n} c_i m_i}, g^{\alpha})^{vv't} \\
&= \hat{e}(g^{\sum_{i=1}^{n} c_i r_i}, g^{\alpha})^{vv'\alpha} \times \hat{e}(g'^{r \sum_{i=1}^{n} c_i m_i}, g^{\alpha})^{vv'\alpha} \\
&= \hat{e}(g^{\alpha \sum_{i=1}^{n} c_i r_i}, g^{\alpha})^{vv'} \times \hat{e}(g', g)^{\alpha^2 vv' r \sum_{i=1}^{n} c_i m_i} \\
&= \hat{e}(g^{\alpha \sum_{i=1}^{n} c_i r_i}, g^{\alpha})^{vv'} \times Z^{vv' r \sum_{i=1}^{n} c_i m_i} \ ,
\end{aligned}$$

$$V = C'^{\sum_{i=1}^{n} c_i m_i} \ , \ V' = C''^{\sum_{i=1}^{n} c_i m_i} \ , \ V'' = g^{\alpha v'} \ , \text{ and } V''' = g^{\alpha} \ .$$

Query-2. Same as the query-1 phase.

Answer. $\mathcal{A}$ answers validity b of $pf^*_{\mathcal{V}}$, and $\mathcal{B}$ uses b to answer the *truncated decisional* 1-BDHE problem directly.

In the above reduction, $\mathcal{B}$ doesn't abort. When $Z = \hat{e}(g, g')^{\alpha^2}$, $\mathcal{A}$ has advantage ϵ' to break proof indistinguishability game. Therefore, the reduced advantage of $\mathcal{B}$ is $\epsilon = \epsilon'/2$ and the reduced time is $t = t' + q_1 t_1 + q_2 t_2 + q_P t_P$. By choosing appropriate $(q_1, q_2, q_P) \in \mathsf{Poly}(k)^3$, we have $q_1 t_1 + q_2 t_2 + q_P t_P \in \mathsf{Poly}(k)$. □

4.3 Delegation Key Unforgeability

This game models the notion that a third party cannot generate a valid delegation key even if he eavesdrops network communications during the delegation phase and corrupts some delegated verifiers. In this game, the challenger $\mathcal{C}$ provides samples of public keys, key tokens, and delegation keys. The adversary $\mathcal{A}$ corrupts some of the samples to obtain the corresponding private keys and tries to generate a valid delegation key for a user $\mathcal{V}^*$.

The delegation key unforgeability game $\mathsf{Game}^{\mathsf{DK-UF}}$ is as follows:

Setup. $\mathcal{C}$ generates public parameter π and user $\mathcal{U}$'s key tuple $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, kt_{\mathcal{U}})$. $\mathcal{C}$ forwards $(\pi, pk_{\mathcal{U}})$ to $\mathcal{A}$.

Query. $\mathcal{A}$ queries oracle $\mathcal{O}_{\mathsf{Dlg}}$ and oracle $\mathcal{O}_{\mathsf{Cor}}$ to obtain samples of public keys, key tokens, and delegation keys, and the corresponding private keys.

- $\mathcal{O}_{\mathsf{Dlg}}$: It samples a user $\mathcal{V}$ and returns $(pk_{\mathcal{V}}, kt_{\mathcal{V}}, dk_{\mathcal{U}\to\mathcal{V}})$.
- $\mathcal{O}_{\mathsf{Cor}}$: $\mathcal{A}$ chooses $(pk_{\mathcal{V}}, kt_{\mathcal{V}}, dk_{\mathcal{U}\to\mathcal{V}})$ from $\mathcal{O}_{\mathsf{Dlg}}$ and obtains $sk_{\mathcal{V}}$ from $\mathcal{O}_{\mathsf{Cor}}$.

Answer. $\mathcal{A}$ generates a valid delegation key $dk_{\mathcal{U}\to\mathcal{V}^*}$ for a user $\mathcal{V}^*$. $\mathcal{A}$ returns $(sk_{\mathcal{V}^*}, pk_{\mathcal{V}^*}, kt_{\mathcal{V}^*}, dk_{\mathcal{U}\to\mathcal{V}^*})$ to $\mathcal{C}$, where $(sk_{\mathcal{V}^*}, pk_{\mathcal{V}^*}, kt_{\mathcal{V}^*})$ is a valid key tuple for $\mathcal{V}^*$. $\mathcal{A}$ wins $\mathsf{Game}^{\mathsf{DK-UF}}$ if $\mathsf{VrfyDK}(\pi, dk_{\mathcal{U}\to\mathcal{V}^*}, pk_{\mathcal{U}}, pk_{\mathcal{V}^*}) = true$. The advantage $Adv^{\mathsf{DK-UF}}_{\mathcal{A}}$ is defined as $\Pr[\mathcal{A} \text{ wins } \mathsf{Game}^{\mathsf{DK-UF}}]$.

We show that our scheme is delegation key unforgeable under the InvCDH assumption.

Theorem 3. *If the InvCDH problem is (t, ϵ)-secure, the above scheme is $(t - q_2 t_2 - q_D t_D - q_C t_C, eq_C\epsilon)$ delegation key unforgeable in the random oracle model, where hash function H_2 is modeled as random oracle $\mathcal{O}_{\mathsf{H}_2}$, e is the Euler's number, (q_2, q_D, q_C) are the numbers of times that an adversary queries $(\mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Dlg}}, \mathcal{O}_{\mathsf{Cor}})$-oracles, and (t_2, t_D, t_C) are the time used by $(\mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Dlg}}, \mathcal{O}_{\mathsf{Cor}})$-oracles to respond an oracle query.*

Proof. Let $\mathcal{A}$ be a probabilistic black-box adversary who wins the delegation key unforgeability game $\mathsf{Game}^{\mathsf{DK-UF}}$ with advantage ϵ' in time t'. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve the InvCDH problem as follows:

Setup. Given an instance (g, g^α) of the InvCDH problem, $\mathcal{B}$ sets the public parameter $\pi = (q, G, g, G_T, g_T, \hat{e}, H_1, H_3)$ and user $\mathcal{U}$'s key tuple $(sk_{\mathcal{U}}, pk_{\mathcal{U}}, kt_{\mathcal{U}}) = (\alpha, g^\alpha, \mathsf{H}_2(g^\alpha)^\alpha)$, where $\mathsf{H}_2(g^\alpha) = g^u$ and $u \in_R \mathbb{Z}_q$. $\mathcal{B}$ invokes $\mathcal{A}$ as a subroutine: $\mathcal{A}^{\mathcal{O}_{\mathsf{H}_2}, \mathcal{O}_{\mathsf{Dlg}}, \mathcal{O}_{\mathsf{Cor}}}(\pi, pk_{\mathcal{U}})$.

Query. $\mathcal{A}$ can query oracles $\mathcal{O}_{\mathsf{H}_2}$, $\mathcal{O}_{\mathsf{Dlg}}$, and $\mathcal{O}_{\mathsf{Cor}}$ during his execution. $\mathcal{B}$ handles these oracles as follows: $\mathcal{B}$ chooses probability $\delta = \frac{q_C}{q_C+1}$.

- $\mathcal{O}_{\mathsf{H}_2}$. $\mathcal{B}$ maintains a table $\mathcal{T}_{\mathsf{H}_2} = \{(g^x, \mathsf{H}_2(g^x), r)\}$ to look up the $\mathcal{O}_{\mathsf{H}_2}$-query records. $\mathcal{B}$ takes g^x as input and outputs y if record $(g^x, y, *)$ exists in $\mathcal{T}_{\mathsf{H}_2}$. Otherwise, $\mathcal{B}$ outputs $\mathsf{H}_2(g^x) = g^{\alpha r}$ with probability δ or outputs $\mathsf{H}_2(g^x) = g^r$ with probability $1-\delta$, and inserts $(g^x, \mathsf{H}_2(g^x), r)$ into $\mathcal{T}_{\mathsf{H}_2}$, where $r \in_R \mathbb{Z}_q$.
- $\mathcal{O}_{\mathsf{Dlg}}$. $\mathcal{B}$ maintains a table $\mathcal{T}_{\mathsf{Dlg}} = \{(v, pk_{\mathcal{V}})\}$ to look up the $\mathcal{O}_{\mathsf{Dlg}}$-query records. $\mathcal{B}$ samples a fresh delegated verifier $\mathcal{V}$, $(pk_{\mathcal{V}}, *, *)$ doesn't exist in table $\mathcal{T}_{\mathsf{H}_2}$, randomly and generates $\mathcal{V}$'s key tuple $(sk_{\mathcal{V}}, pk_{\mathcal{V}}, kt_{\mathcal{V}}) = (v, g^v, \mathsf{H}_2(g^v)^v = (g^{\alpha v'})^v)$ with probability δ or $(sk_{\mathcal{V}}, pk_{\mathcal{V}}, kt_{\mathcal{V}}) = (\alpha v, g^{\alpha v}, \mathsf{H}_2(g^{\alpha v})^{\alpha v} = (g^{v'})^{\alpha v})$ with probability $1-\delta$, where $v, v' \in_R \mathbb{Z}_q$. $\mathcal{B}$ inserts $(pk_{\mathcal{V}}, \mathsf{H}_2(pk_{\mathcal{V}}), v')$ into $\mathcal{T}_{\mathsf{H}_2}$ and inserts $(v, pk_{\mathcal{V}})$ into $\mathcal{T}_{\mathsf{Dlg}}$. Then $\mathcal{B}$ outputs $(pk_{\mathcal{V}}, kt_{\mathcal{V}}, dk_{\mathcal{U}\to\mathcal{V}})$, where

$$dk_{\mathcal{U}\to\mathcal{V}} = kt_{\mathcal{V}}^{1/sk_{\mathcal{U}}} = (g^{\alpha v v'})^{1/\alpha} = g^{vv'} \ .$$

- $\mathcal{O}_{\mathsf{Cor}}$. $\mathcal{B}$ takes $(pk_{\mathcal{V}}, kt_{\mathcal{V}}, dk_{\mathcal{U}\to\mathcal{V}})$ as input and rejects if either record $(pk_{\mathcal{V}}, *, *)$ doesn't exist in table $\mathcal{T}_{\mathsf{H}_2}$, record $(*, pk_{\mathcal{V}})$ doesn't exist in table $\mathcal{T}_{\mathsf{Dlg}}$, or $(kt_{\mathcal{V}}, dk_{\mathcal{U}\to\mathcal{V}})$ isn't consistent with $\mathcal{T}_{\mathsf{H}_2}$ and $\mathcal{T}_{\mathsf{Dlg}}$. $\mathcal{B}$ outputs $sk_{\mathcal{V}} = v$ if $pk_{\mathcal{V}} = g^v$. Otherwise, $sk_{\mathcal{V}} = \alpha v$, and $\mathcal{B}$ aborts.

Answer. $\mathcal{A}$ forges a delegation key $dk_{\mathcal{U}\to\mathcal{V}^*}$ for a user $\mathcal{V}^*$ whose key tuple is $(sk_{\mathcal{V}^*}, pk_{\mathcal{V}^*}, kt_{\mathcal{V}^*})$, and outputs $(sk_{\mathcal{V}^*}, pk_{\mathcal{V}^*}, kt_{\mathcal{V}^*}, dk_{\mathcal{U}\to\mathcal{V}^*})$. $\mathcal{B}$ rejects if either record $(pk_{\mathcal{V}^*}, *, *)$ doesn't exist in table $\mathcal{T}_{\mathsf{H}_2}$ or $\hat{e}(pk_{\mathcal{U}}, dk_{\mathcal{U}\to\mathcal{V}^*}) \neq \hat{e}(pk_{\mathcal{V}^*}, \mathsf{H}_2(pk_{\mathcal{V}^*}))$. If $\mathsf{H}_2(pk_{\mathcal{V}^*}) = g^{\alpha r^*}$, $\mathcal{B}$ aborts. Otherwise, $\mathsf{H}_2(pk_{\mathcal{V}^*}) = g^{r^*}$, and $\mathcal{B}$ computes $g^{\frac{1}{\alpha}}$ as follows:

$$\begin{aligned}
&\hat{e}(pk_{\mathcal{U}}, dk_{\mathcal{U}\to\mathcal{V}^*}) = \hat{e}(pk_{\mathcal{V}^*}, \mathsf{H}_2(pk_{\mathcal{V}^*})) \\
\Rightarrow\ &\hat{e}(g^{\alpha}, dk_{\mathcal{U}\to\mathcal{V}^*}) = \hat{e}(g^{sk_{\mathcal{V}^*}}, g^{r^*}) \\
\Rightarrow\ &dk_{\mathcal{U}\to\mathcal{V}^*} = g^{sk_{\mathcal{V}^*} r^*/\alpha} \\
\Rightarrow\ &g^{\frac{1}{\alpha}} = (dk_{\mathcal{U}\to\mathcal{V}^*})^{1/sk_{\mathcal{V}^*} r^*}
\end{aligned}$$

In the above reduction, $\mathcal{B}$ doesn't abort with probability $\delta^{q_C}(1-\delta)$. When choosing $\delta = \frac{q_C}{q_C+1}$, we have $\delta^{q_C}(1-\delta) = (1-\frac{1}{q_C+1})^{q_C}\frac{1}{q_C+1} = (1-\frac{1}{q_C+1})^{q_C+1}\frac{1}{q_C} \geq \frac{1}{eq_C}$. Therefore, the reduced advantage is $\epsilon = \frac{\epsilon'}{eq_C}$, and the reduced time is $t = t' + q_2 t_2 + q_D t_D + q_C t_C$. By choosing appropriate $(q_2, q_D, q_C) \in \mathsf{Poly}(k)^3$, we have $(eq_C, q_2 t_2 + q_D t_D + q_C t_C) \in \mathsf{Poly}(k)^2$. □

5 Conclusion

We proposed a delegable provable data possession model that provides delegable (authorized) verification on remote data. Delegable PDP allows a trusted third party to check data integrity under data owner's permission and prevents the trusted third party to re-delegate this verification capability to others. This

feature is desired on private data in the public cloud. We provided a construction for the delegable PDP problem and proved its security in the random oracle model.

Due to using pairing operations on blocks directly, each block m_i is limited to k-bit long. We shall develop a new delegation method without using pairing in the future. Dynamic operations, such as insertion, deletion, modification, etc., on stored data is useful. Supporting efficient dynamic operations on stored data is another direction of our future works.

References

1. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007, pp. 598–609 (2007)
2. Ateniese, G., Di Pietro, R., Mancini, L.V., Tsudik, G.: Scalable and efficient provable data possession. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, SecureComm 2008, pp. 9:1–9:10 (2008)
3. Ateniese, G., Kamara, S., Katz, J.: Proofs of Storage from Homomorphic Identification Protocols. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 319–333. Springer, Heidelberg (2009)
4. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical Identity Based Encryption with Constant Size Ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)
5. Boneh, D., Gentry, C., Waters, B.: Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 258–275. Springer, Heidelberg (2005)
6. Bowers, K.D., Juels, A., Oprea, A.: Hail: a High-availability and Integrity Layer for Cloud Storage. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 187–198 (2009)
7. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: theory and implementation. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, CCSW 2009, pp. 43–54 (2009)
8. Chen, B., Curtmola, R., Ateniese, G., Burns, R.: Remote data checking for network coding-based distributed storage systems. In: Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop, CCSW 2010, pp. 31–42 (2010)
9. Curtmola, R., Khan, O., Burns, R.: Robust remote data checking. In: Proceedings of the 4th ACM International Workshop on Storage Security and Survivability, StorageSS 2008, pp. 63–68 (2008)
10. Curtmola, R., Khan, O., Burns, R., Ateniese, G.: Mr-pdp: Multiple-replica provable data possession. In: Proceedings of the 2008 the 28th International Conference on Distributed Computing Systems, ICDCS 2008, pp. 411–420 (2008)
11. Damgård, I.B.: Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 445–456. Springer, Heidelberg (1992)
12. Erway, C., Küpçü, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 213–222 (2009)

13. Gentry, C.: Practical Identity-based Encryption without Random Oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 445–464. Springer, Heidelberg (2006)
14. Juels, A., Kaliski Jr., B.S.: Pors: proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS 2007, pp. 584–597 (2007)
15. Shacham, H., Waters, B.: Compact Proofs of Retrievability. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 90–107. Springer, Heidelberg (2008)
16. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: Proceedings of the 29th Conference on Information Communications, INFOCOM 2010, pp. 525–533 (2010)
17. Wang, Q., Wang, C., Li, J., Ren, K., Lou, W.: Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 355–370. Springer, Heidelberg (2009)