# Cooperative Provable Data Possession

Yan Zhu, Huaixi Wang, Zexing Hu
Peking University, Beijing, 100871, China
{yan.zhu,wanghx,huzx}@pku.edu.cn

Gail-Joon Ahn, Hongxin Hu, Stephen S. Yau
Arizona State University, Tempe, Arizona, 85281
{gahn,hxhu,yau}@asu.edu

April 26, 2010

## Contents

**Abstract**

Provable data possession (PDP) is a technique for ensuring the integrity of data in outsourcing storage service. In this paper, we address the construction of efficient PDP schemes on hybrid clouds to support scalability of service and data migration, in which we consider the existence of multiple cloud service providers (CSP) to cooperatively store and maintain the clients' data. The proposed PDP schemes include an *interactive* PDP (IPDP) and a *cooperative* PDP (CPDP) schemes adopting zero-knowledge property and three-layered index hierarchy, respectively. In particular, we present an efficient method for selecting the optimal number of sectors in each block to minimize the computation costs of clients and storage service providers. Our experiments show that the verification requires a small, constant amount of overhead, which minimizes communication complexity.

# 1 Introduction

In recent years, storage service in clouds has become a new profit growth point by providing a comparably low-cost, scalable, location-independent platform for managing clients' data. However, if such an important service is vulnerable to security attacks, it would bring irretrievable losses to the clients since their data and archives are stored into an uncertain storage pool outside the enterprise. Therefore, it is necessary for cloud service providers (CSP) to provide secure management techniques to their storage services. A provable data possession (PDP) [1] is a probabilistic proof technique for a storage provider to prove that clients' data remains intact. In other words, the clients can fully recover their data and have confidence to use the recovered data. This highlights a strong need to seek an effective solution for checking if their data has been tampered with or deleted without downloading the latest version of data.

**Motivation.** A hybrid cloud is a cloud computing environment in which an organization provides and manages some internal resources as well as external resources. For example, as shown in Figure 1, an organization, Hybrid Cloud I, uses a public cloud service such as Amazon's EC2 for general computing purposes while storing customers' data within its own data center in a private cloud.

Although cloud computing is often said to be the future of the industry, the hybrid model is more prevalent for a number of reasons. Large enterprisers have already substantial investments in the infrastructure required to provide local resources. Furthermore, many organizations would prefer to keep sensitive data under their own control, ensuring security requirements.

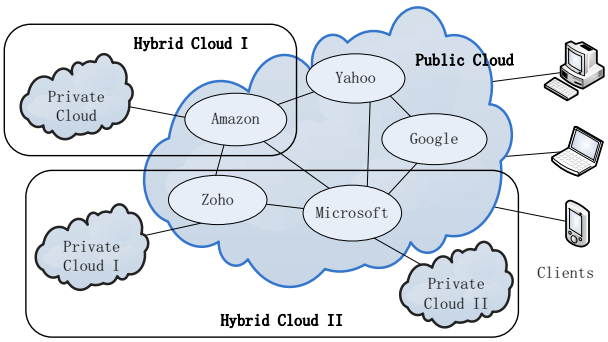


Figure 1: Cloud computing types: private cloud, public cloud and hybrid cloud.

The previous schemes focus on the PDP issues at untrusted servers (public clouds). For ease of use, some existing PDP schemes work in a publicly verifiable way so that anyone can use the verification protocol of PDP schemes to prove the availability of the stored data. However, when such schemes are used for private clouds, the adversary could exploit the public verification service to obtain the information of private data.

In cloud computing, one of the core design principles is dynamic scalability, which guarantees cloud storage service to either handle growing amounts of application data in a flexible manner or to be readily enlarged. By integrating multiple private and public cloud services, hybrid clouds can effectively provide dynamic scalability of service and data migration. For example, a client might integrate the data from multiple private or public providers into a backup or archive file (see Hybrid Cloud II in Figure 1), or a service might capture the data from the other services from private clouds, but the intermediate data and results are stored in hybrid clouds [13, 14]. Although PDP schemes evolved around public clouds offer a publicly accessible remote interface to check and manage the tremendous amount of data, the majority of today's PDP schemes is incapable of satisfying such an inherent requirement of hybrid clouds in terms of bandwidth and time. In order to solve this problem, we consider a hybrid cloud storage service involving three different

entities, as illustrated in Figure 2: the cloud client who stores or uses data in the cloud; the cloud service provider (CSP) which has significant storage space and computation resources to manage and provide storage services; the trusted third party (TTP) who is trusted to store the clients' audit data and offer the query services for their data.

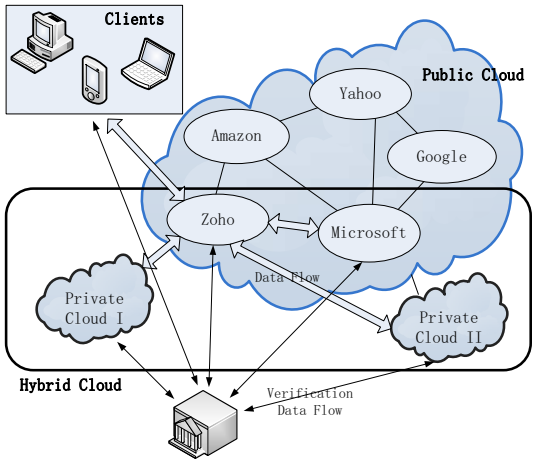


Figure 2: Cloud data storage architecture

In this architecture, we consider the existence of multiple CSPs to cooperatively store and maintain the clients' data, and a publicly verifiable PDP is used to verify the integrity and availability of their stored data in CSPs. The clients are allowed to dynamically access and update their data for various applications and, the verification process of PDP is seamlessly performed for the clients in hybrid cloud. This characteristic is important for hybrid clouds to support dynamic scalability.

**Background.** To check the availability and integrity of the stored data in cloud storage, the researchers have proposed two basic approaches called Provable Data Possession (PDP) [1] and Proofs of Retrievability (POR)[12]. Ateniese et al. [1] first proposed the PDP model for ensuring possession of files on untrusted storages and provided a RSA-based scheme for the static case that achieves the $O(1)$ communication costs. They also proposed a publicly verifiable version, which allows anyone, not just the owner, to challenge the server for data possession. This property greatly extended application areas of PDP protocol due to the separation of data owners and the users. However, similar to replay attacks, these schemes are insecure in a dynamic scenario because of the dependence on the index of blocks. Moreover, it does not fit to hybrid clouds due to the loss of homomorphism in a proof of possession.

In order to support dynamic data operations, Ateniese et al. have developed a dynamic PDP solution called Scalable PDP [2]. They proposed a lightweight PDP scheme based on cryptographic Hash function and symmetric key encryption, but the server can deceive the owner by using the previous metadata or responses due to lack of the randomness in the challenge. The number of updates and challenges is limited and fixed a priori. Also, one cannot perform block insertions anywhere. Based on this work, ChrisErway et al. [9] introduced two Dynamic PDP schemes with a Hash function tree to realize the $O(\log n)$ communication and computational costs for a file consisting of $n$ blocks. The basic scheme, called DPDP-I, remains the drawback of SPDP, and in the 'blockless' scheme, called DPDP-II, the data blocks $\{m_{i_j}\}_{j\in[1,t]}$ can be leaked by the response of challenge, $M = \sum_{j=1}^{t} a_j m_{i_j}$, where $a_j$ is a random value in the challenge. Furthermore, these schemes are not effective to hybrid clouds because the verification path of the challenge block cannot be stored completely in a cloud.

Juels and Kaliski [12] presented a POR scheme which relies largely on preprocessing steps the client conducts before sending a file to CSP. Unfortunately, these operations prevent any efficient extension to update data. Shacham and Waters [16] proposed an improved version of this protocol called Compact POR, which uses homomorphic property to aggregate a proof into $O(1)$ authenticator value and $O(t)$ computation costs for $t$ challenge blocks, but their solution is also static and there exists the leakage of data blocks in the verification process. Wang et al. [17] presented a dynamic scheme with $O(\log n)$ costs by integrating the above CPOR scheme and Merkle Hash Tree (MHT) in DPDP. Furthermore, serveral POR schemes and models have been recently proposed including [6, 8]. Since the response of challenges has homomorphic property, the above schemes (especially CPOR schemes) can leverage the PDP construction on hybrid clouds. In this paper, we also focus on the homomorphic property of CPOR.

**Contributions.** In this paper, we focus on the construction of PDP scheme on hybrid clouds, supporting privacy

Table 1: Comparison of PDP schemes for a file consisting of $n$ blocks.

| Scheme | CSP comp. | Client Comp. | Comm. | Cloud | Frag. | Privacy | Dynamic Operations | | | Prob. of Detection |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | modify | insert | delete | |
| PDP[1] | $O(t)$ | $O(t)$ | $O(1)$ | single | | ✓ | | | | $1-(1-\rho)^t$ |
| SPDP[2] | $O(t)$ | $O(t)$ | $O(t)$ | single | ✓ | ✓ | ✓$^\sharp$ | | ✓$^\sharp$ | $1-(1-\rho)^{t\cdot s}$ |
| DPDP-I[9] | $O(t\log n)$ | $O(t\log n)$ | $O(t\log n)$ | single | | ✓ | ✓ | ✓ | ✓ | $1-(1-\rho)^t$ |
| DPDP-II[9] | $O(t\log n)$ | $O(t\log n)$ | $O(t\log n)$ | single | | | ✓ | ✓ | ✓ | $1-(1-\rho)^{\Omega(n)}$ |
| CPOR-I[16] | $O(t)$ | $O(t)$ | $O(1)$ | single* | | | | | | $1-(1-\rho)^t$ |
| CPOR-II[16] | $O(t+s)$ | $O(t+s)$ | $O(s)$ | single* | ✓ | | | | | $1-(1-\rho)^{t\cdot s}$ |
| IPDP | $O(t+s)$ | $O(t+s)$ | $O(s)$ | single | ✓ | ✓ | ✓ | ✓ | ✓ | $1-(1-\rho)^{t\cdot s}$ |
| CPDP | $O(t+cs)$ | $O(t+s)$ | $O(cs)$ | hybrid | ✓ | ✓ | ✓ | ✓ | ✓ | $1-\prod_{P_k\in\mathcal{P}}(1-\rho_k)^{t\cdot s\cdot r_k}$ |

$s$ is the number of sectors in each block, $t$ is the number of sampling blocks, $\mathcal{P}$ denotes a set of CSPs in hybrid clouds, where $\mathcal{P}=\{P_k\}$ and $|\mathcal{P}|=c$, $*$ indicates that the scheme could support hybrid cloud, $\sharp$ indicates that a operation is performed only a limited (pre-determined) number of times, $\rho$ and $\rho_k$ is the probability of block corruption in a single cloud or the $k$-th CSP in hybrid cloud $\mathcal{P}$, respectively, and $r_k$ is the proportion of data blocks in the $k$-th CSP.

protection and dynamic scalability. The main contributions of this work are summarized as follows:

- Our work is the first attempt to introduce a formal framework for Interactive Provable Data Possession (IPDP) and provides a practical Zero-Knowledge IPDP (ZK-IPDP) solution for private clouds to prevent data leakage in PDP verification;
- We also provide an effective construction of Cooperative Provable Data Possession (CPDP) using Homomorphic Verifiable Responses (HVR). This construction realizes zero-knowledge and transparent property for the clients to store and manage the resources on hybrid cloud.
- In addition, we present an efficient method for selecting the optimal parameter value to minimize computational overheads of CSP and clients' operations. Our experimental results also validate the effectiveness of the optimal value. This value ensures that the extra storage of tags does not exceed 1% in CSP.

Both schemes, ZK-IPDP and ZK-CPDP, use homomorphic property, on which the responses of the client's challenge computed from multiple CSPs can be combined into a single response as the final result of hybrid clouds. By using this mechanism, the client can be convinced of data possession without knowing what machines or in what geographical locations their files reside.

Let $n$ be the number of blocks and $s$ be the number of sectors in each block. The communication and computation complexity are $O(s)$ for verification process and dynamic data update in our schemes. Given the probability of sector corruption $\rho$ (or $\rho_k$ for $k$-th CSP in hybrid clouds $\mathcal{P}=\{P_k\}$), the detection probability $P$ is $1-(1-\rho)^{n\cdot s\cdot w}$ and $1-\prod_{P_k\in\mathcal{P}}(1-\rho_k)^{n\cdot s\cdot r_k\cdot w}$, where $w$ is the sampling probability in verification process, $r_k$ is the proportion of data blocks in $k$-th CSP (see Table 1). Given the fixed $\rho$ and $P$, we found that the sampling number of verification protocol $t=n\cdot w$ is independent of the file size. This means that the challenge message in verification protocol has a constant length for the files with different sizes. Furthermore, the number of sectors $s$ has an optimal value to minimize the computation costs of CSP and clients' operations. This value can support 16T-Bytes in NTFS.

We list the features of our PDP schemes (ZK-IPDP and ZK-CPDP) in Table 1. We also include a comparison of related techniques, such as, PDP [1], DPDP[9], and CPOR [16]. It clearly shows that our schemes not only support privacy protection and dynamic data operations, but also have the $O(1)$ computational and communication overheads, which is independent of the file size.

**Organization.** The rest of the paper is organized as follows. In Section 2, we describe some basic notations and common structure of PDP. In Section 3, we define the formal models of IPDP and CPDP. We introduce ZK-IPDP scheme on private clouds and ZK-CPDP scheme on hybrid clouds in Section 4 and Section 6, respectively. We describe the performance of our schemes and the results of experiments in Section 7 and Section 8 concludes this paper.

# 2 Preliminaries

## 2.1 Notations and Preliminaries

A homomorphism is a map $f:\mathbb{P}\to\mathbb{Q}$ between two groups such that $f(g_1\oplus g_2)=f(g_1)\otimes f(g_2)$ for all $g_1,g_2\in\mathbb{P}$, where $\oplus$ denotes the operation in $\mathbb{P}$ and $\otimes$ denotes the operation in $\mathbb{Q}$. This notation has been used to define a Homomorphic

Verifiable Tags (HVTs) in [1]: Given two values $\sigma_i$ and $\sigma_j$ for two message $m_i$ and $m_j$, anyone can combine them into a value $\sigma'$ corresponding to the sum of the message $m_i + m_j$.

When provable data possession is considered as a challenge-response protocol, we also extend this notation to introduce the concept of a Homomorphic Verifiable Responses (HVRs), which is used to integrate multiple responses from the different CSPs in cooperative PDP scheme, as follows:

**Definition 1** (homomorphic verifiable response)**.** *A response is called homomorphic verifiable response in PDP protocol, if given two responses $\theta_i$ and $\theta_j$ for two challenges $Q_i$ and $Q_j$ from two CSPs, there exists an efficient algorithm to combine them into a response $\theta$ corresponding to the sum of the challenges $Q_i \bigcup Q_j$.*

Let $\mathbb{H} = \{H_k\}$ be a keyed hash family of functions $H_k : \{0,1\}^n \to \{0,1\}^*$ index by $k \in \mathcal{K}$. We say that algorithm $\mathcal{A}$ has advantage $\epsilon$ in breaking the collision-resistance of $\mathcal{H}$ if

$$\Pr[\mathcal{A}(k) = (m_0, m_1) : m_0 \neq m_1, H_k(m_0) = H_k(m_1)] \geq \epsilon,$$

where the probability is over the random choice of $k \in \mathcal{K}$ and the random bits of $\mathcal{A}$.

**Definition 2** (collision-resistant hash)**.** *A hash family $\mathbb{H}$is $(t, \epsilon)$-collision-resistant if no $t$-time adversary has advantage at least $\epsilon$ in breaking the collision-resistance of $\mathbb{H}$.*

We set up our systems using bilinear pairings proposed by Boneh and Franklin [5]. Let $\mathbb{G}$ be two multiplicative groups using elliptic curve conventions with large prime order $p$. The function $e$ be a computable bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with the following properties: for any $G, H \in \mathbb{G}$ and all $a, b \in \mathbb{Z}_p$, we have 1) Bilinearity: $e([a]G, [b]H) = e(G, H)^{ab}$. 2) Non-degeneracy: $e(G, H) \neq 1$ unless $G$ or $H = 1$. 3) Computability: $e(G, H)$ is efficiently computable.

**Definition 3** (bilinear map group system)**.** *A bilinear map group system is a tuple $\mathbb{S} = \langle p, \mathbb{G}, \mathbb{G}_T, e\rangle$ composed of the objects as described above.*

Next, we recall two representative schemes in [16] to discuss the security of the existing schemes, as follows:

## 2.2 Public Blocked Scheme

The client breaks a (possibly encoded) file $F$ into $n$ blocks $m_1, \cdots, m_n \in \mathbb{Z}_p$ for some large prime $p$. Let $e : G \times G \to G_T$ be a computable bilinear map with group $G$'s support being $\mathbb{Z}_p$ and $H : \{0,1\}^* \to G$ be the BLS Hash function. A client's private key is $sk = x \in \mathbb{Z}_p$, and her public key is $pk = (v, u)$, where $v = g^x \in G$ and $g, u$ is two generators in $G$. the signature on block $i$ is $\sigma_i = [H(i)u^{m_i}]^x$. On receiving index-coefficient pair query $Q = \{(i, v_i)\}_{i \in I}$ for an index $I$, the server computes and sends back $\sigma' \leftarrow \prod_{(i,v_i)\in Q} \sigma_i^{v_i}$ and $\mu \leftarrow \sum_{(i,v_i)\in Q} v_i m_i$. The verification equation is

$$e(\sigma', g) = e(\prod\nolimits_{(i,v_i)\in Q} H(i)^{v_i} \cdot u^{\mu}, v).$$

The scheme is not secure due to the leakage of file information and the forging of tags, as follows:

**Attack 1.** *The adversary can get the file and tag information by running or wiretapping the $n$-times verification communication for a file with $n$ blocks.*

*Proof.* Let $n$ be the number of blocks in the attacked file and $\mu^{(k)} = \sum_{i=1}^{n} v_i \cdot m_i$ denote the response of the $k$-th user's challenge $Q^{(k)}$, where we fill $v_i = 0$ to extent the challenge coefficients, that is, $v_i = 0$ for any $(i, v_i) \notin Q$. Such that the adversary gets the responses $\{(\sigma^{(1)}, \mu^{(1)}), \cdots, (\sigma^{(1)}, \mu^{(n)})\}$ after he finishes $n$ times queries. These responses can generate the equations

$$\begin{cases} \mu^{(1)} &= v_1^{(1)} m_1 + \cdots + v_n^{(1)} m_n \\ \vdots & \qquad\qquad \vdots \\ \mu^{(n)} &= v_1^{(n)} m_1 + \cdots + v_n^{(n)} m_n \end{cases}$$

where, $v_i^{(k)}$ is known for all $i \in [1, n]$ and $k \in [1, n]$. The adversary can compute $f = (m_1, \cdots, m_n)$ by solving the equations. Similarly, the adversary can get all tags $\sigma_1, \cdots, \sigma_n$ by the equation system $\sigma^{(i)} = {\sigma_1}^{v_1^{(i)}} \cdot {\sigma_2}^{v_2^{(i)}} \cdot \cdots \cdot {\sigma_n}^{v_n^{(i)}}$ for $i \in [1, n]$. Note that, the above attacks are not based on any kind of assumption. □

**Attack 2.** *The server can deceive the client by forging the tag of data block if the client's private/public keys are reused for the different files, the client modifies the data in a file, or the client repeats to insert and delete data blocks.*

*Proof.* This attack can be occurred in a variety of cases, but they have a common feature that the same hash value $H(i)$ been used at least 2 times. For example, the adversary gets two data-tag pairs $(m_i, \sigma_i)$ and $(m_i', \sigma_i')$ with the same $H(i)$ from two file $F$ and $F'$, such that $\sigma_i = (H(i) \cdot u^{m_i})^x$, $\sigma_i' = (H(i) \cdot u^{m_i'})^x$. The adversary first computes $\sigma_i \cdot \sigma_i'^{-1} = u^{(m_i - m_i')x}$ and gets $u^x = (\sigma_i \cdot \sigma_i'^{-1})^{\frac{1}{m_i - m_i'}}$ by using extended Euclidean algorithm $\gcd(m_i - m_i', p)$. Further, the adversary can capture the $H(i)^x$ (or $H(k)^x$ for $\forall k \in [1, n]$) by $H(i)^x = \frac{\sigma_i}{(u^x)^{m_i}} = (\sigma_i'^{m_i} / \sigma_i^{m_i'})^{\frac{1}{m_i - m_i'}}$. Hence, for an arbitrary message $m_k^* \neq m_k$, the forged tag is generated by

$$\sigma_k^* = H(k)^x \cdot (u^x)^{m_k^*} = \sigma_k \cdot (\sigma_i \cdot \sigma_i'^{-1})^{\frac{m_k^* - m_k}{m_i - m_i'}}.$$

This means that the adversary can forge the data and tags at any position within the file. □

## 2.3 Public Fragmented Scheme

Given a file $F$, the client split $F$ into $n$ blocks $(m_1, \cdots, m_n)$ and each block $m_i$ is also split into $s$ sectors $(m_{i,1}, \cdots, m_{i,s}) \in \mathbb{Z}_P^s$ for some enough large $p$. Let $e : G \times G \to \mathbb{G}_T$ be a bilinear map, $g$ be a generator of $\mathbb{G}$, and $H : \{0,1\}^* \to G$ be the BLS hash. The secret key is $sk = x \in_R \mathbb{Z}_p$ and the public key is $pk = (g, v = g^x)$. The client chooses $s$ random $u_1, \cdots, u_s \in_R \mathbb{G}$ as the verification information $t = (F_n, u_1, \cdots, u_s)$, where $F_n$ is the file name.

For each $i \in [1, n]$, the tag at the $i$-th block is $\sigma_i = (H(Fn||i) \cdot \prod_{j=1}^s u_j^{m_{i,j}})^x$. On receiving query $Q = \{(i, v_i)\}_{i \in I}$ for an index set $I$, the server computes and sends back $\sigma' \leftarrow \prod_{(i,v_i) \in Q} \sigma_i^{v_i}$ and $\mu = (\mu_1, \cdots, \mu_s)$, where $\mu_j \leftarrow \sum_{(i,v_i) \in Q} v_i m_{i,j}$. The verification equation is

$$e(\sigma', g) = e(\prod\nolimits_{(i,v_i) \in Q} H(Fn||i)^{v_i} \cdot \prod\nolimits_{j=1}^s u_j^{\mu_j}, v).$$

This scheme is also not secure due to the leakage of file information and the forging of tags, as follows:

**Attack 3.** *The adversary can get the file and tag information by running or wiretapping the $n$-times verification communication for a file with $n \times s$ sectors.*

*Proof.* The proof is similar to that of Theorem 1. Let $s$ be the number of sectors. Given $n$ times challenges $(Q^{(1)}, \cdots, Q^{(n)})$ and their the results $((\sigma'^{(1)}, \mu^{(1)}), \cdots, (\sigma'^{(n)}, \mu^{(n)}))$, $\mu^{(k)} = (\mu_1^{(k)}, \cdots, \mu_s^{(k)})$ and $Q^{(k)} = \{(i, v_i)\}_{i \in I}$, the adversary can solve the system of equations, $\mu_i^{(k)} = m_{1,i} \cdot v_1^{(k)} + \cdots + m_{n,i} \cdot v_n^{(k)}$ for $k \in [1, n]$, to reach $\{m_{1,i}, \cdots, m_{n,i}\}$. After $s$ times solving these equations ($i \in [1, s]$), the adversary can obtain the whole file, $F = \{m_{i,j}\}_{j \in [1,s]}^{i \in [1,n]}$. Similarly, the adversary can get all tags $\sigma_1, \cdots, \sigma_n$ by using $\sigma'^{(1)}, \cdots, \sigma'^{(n)}$. □

**Attack 4.** *Let $s$ be the number of sectors in each blocks. The server can deceive the client by forging the tag of data block if the client's private/public keys and the file name are reused for 2 different files with the number of blocks $n \geq 2s$, the client modifies at least $s$ data blocks in a file, or the client repeats at least $s$ times to insert and delete data blocks.*

*Proof.* The proof is similar to that of Theorem 2. Assume two file $F$ and $F'$ have the same file name $Fn$. The adversary choices $2s$ different blocks randomly from the same position in two files, without loss of generality, $(m_1, \cdots, m_{2s})$ and $(m_1', \cdots, m_{2s}')$, such that $\sigma_i = (H(Fn, i) \cdot \prod_{j=1}^s u_j^{m_{i,j}})^x$, $\sigma_i' = (H(Fn, i) \cdot \prod_{j=1}^s u_j'^{m_{i,j}'})^x$ for $i \in [1, 2s]$. The adversary computes $\Delta_1, \cdots, \Delta_{2s}$ by using $\Delta_i = \sigma_i \cdot \sigma_i'^{-1} = \prod_{j=1}^s (u_j^{m_{i,j}} \cdot (u'_j{}^{m_{i,j}'})^{-1})^x$. These values can generate the following system of equations

$$(\Delta_1, \cdots, \Delta_{2s})^T = M \cdot (u_1^x, \cdots, u_s^x, u_1'^x, \cdots, u_s'^x)^T.$$

where, $M$ denotes a $2s \times 2s$ matrix as

$$M = \begin{pmatrix} m_{1,1} & \cdots & m_{1,s} & -m_{1,1}' & \cdots & -m_{1,s}' \\ \vdots & & \vdots & \vdots & & \vdots \\ m_{2s,1} & \cdots & m_{2n,s} & -m_{2s,1}' & \cdots & -m_{2s,s}' \end{pmatrix}$$

Let $D = M^{-1} = (d_{i,j})_{2s \times 2s}$. The adversary can compute $u_i^x = \prod_{j=1}^{2s} (\frac{\sigma_j}{\sigma_j'})^{d_{i,j}}$ and $u_i'^x = \prod_{j=1}^{2n} (\frac{\sigma_j}{\sigma_j'})^{d_{s+i,j}}$ for $i \in [1, s]$. Such that $H(Fn, k)^x = \sigma_k / \prod_{j=1}^s (u_j^x)^{m_{k,j}}$ for $k \in [1, n]$. Hence, for any message $m_k^* \neq m_k$, the forged tag is $\sigma_k^* = H(Fn, k)^x \cdot \prod_{j=1}^s (u_j^x)^{m_{k,j}^*} = \sigma_k \cdot \prod_{j=1}^s (u_j^x)^{m_{k,j}^* - m_{k,j}}$. □

## 2.4 Common Fragmented PDP

We start by recalling the schemes in [1, 16] and can find that these schemes have the common framework and characters (showed in Figure 3): 1) the file is split into $n \times s$ sectors and each block ($s$ sectors) corresponds to a tag, so that the storage of signature tags can be reduced with increase of $s$; 2) the verifier can verify the integrity of file in random sampling approach, which is of utmost importance for large or huge files; 3) these schemes rely on homomorphic properties to aggregate the data and tags into a constant size response, which minimizes network communication. The signals used in Figure 3 are explained in Table 2.

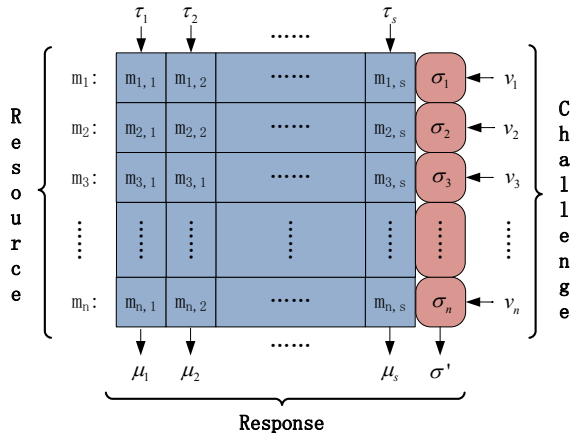


Figure 3: The framework of PDP model.

The above model, considered to be common representative for some existing schemes, is readily converted to MAC-based, ECC or RSA schemes. These schemes, built from BLS signatures and secure in random oracle model, have the shortest query and response with public verifiability. However, the above attacks show that these schemes have still some security problems in privacy and dynamic aspects:

- The validation process leads to the leakage of file and tag information, so these schemes cannot be used in private cloud or privacy preserving applications;
- The dynamic operations (insert, delete, or modify) can incur a risk to the tag forgery in untrusted servers, so these schemes is merely used to backup or archive data.

# 3 Definition of PDP Models

In this section, we present the precise definition and security models of our schemes (IPDP and CPDP). For clarity, we list some used signals in Table 2.

Table 2: The signal and its explanation.

| Sig. | Repression |
|---|---|
| $n$ | the number of blocks in a file; |
| $s$ | the number of sectors in each block; |
| $t$ | the number of index coefficient pairs in a query; |
| $c$ | the number of clouds to store a file; |
| $F$ | the file with $n \times s$ sectors, i.e., $F = \{m_{i,j}\}_{j\in[1,s]}^{i\in[1,n]}$; |
| $\sigma$ | the set of tags, i.e., $\sigma = \{\sigma_i\}_{i\in[1,n]}$; |
| $Q$ | the set of index-coefficient pairs, i.e., $Q = \{(i, v_i)\}$; |
| $\theta$ | the response for the challenge $Q$. |

### 3.1 Interactive Provable Data Possession

We present the definition of Interactive Provable Data Possession (IPDP) based on interactive proof system:

**Definition 4** (Interactive-PDP)**.** *An interactive provable data possession scheme $\mathcal{S}$ is a collection of two algorithms and an interactive proof system, $\mathcal{S} = (\mathcal{K}, \mathcal{T}, \mathcal{P})$:*

$\mathcal{K}eyGen(1^s)$**:** *takes a security parameter $s$ as input, and returns a secret key $sk$ or a public-secret keypair $(pk, sk)$;*

$\mathcal{T}agGen(sk, F)$**:** *takes as inputs the secret key $sk$ and a file $F$, and returns the triples $(\zeta, \psi, \sigma)$, where $\zeta$ denotes the secret of tags, $\psi$ is the set of public verification parameters $u$ and index information $\chi$, i.e., $\psi = (u, \chi)$; $\sigma$ denotes the set of verification tags;*

$\mathcal{P}roof(P, V)$**:** *is a protocol of proof of possession between prover ($P$) and verifier ($V$). At the end of the protocol run, $V$ returns $\{0|1\}$, where 1 means the file is being correct stored on the server. It includes two cases:*

- *$\langle P(F, \sigma), V(sk, \zeta)\rangle$ is a private proof, where $P$ takes as input a file $F$ and a set of tags $\sigma$, and $V$ takes as input a secret key $sk$ and a secret of tags $\zeta$;*
- *$\langle P(F, \sigma), V\rangle(pk, \psi)$ is a public proof, where $P$ takes as input a file $F$ and a set of tags $\sigma$, and a public key $pk$ and a set of public parameters $\psi$ is the common input between $P$ and $V$.*

*where, $P(x)$ denotes the subject $P$ holds the secret $x$ and $\langle P, V\rangle(x)$ denotes both parties $P$ and $V$ share a common data $x$ in a protocol.*

This model is a more general PDP model than the existing PDP models. As the verification process considered as an interactive protocol, this definition does not limit to the specific steps of verification, including scale, sequence, and the number of moves in protocol, so it can provide greater convenience for the construction of protocol. Further, this paper will consider only the construction of public proof protocol.

### 3.2 Security Requirements

According to the standard definition of interactive proof system proposed by Bellare and Goldreich [10], the PDP protocol $\mathcal{P}roof(P, V)$ has two requirements:

**Definition 5.** *A pair of interactive machines $(P, V)$ is called an available provable data possession for a file $F$ if $P$ is a (unbounded) probabilistic algorithm, $V$ is a deterministic polynomial-time algorithm, and the following conditions hold for some polynomial $p_1(\cdot), p_2(\cdot)$, and all $s \in \mathbb{N}$:*

- *Completeness: For every $\sigma \in TagGen(sk, F)$,*

$$\Pr[\langle P(F, \sigma), V\rangle(pk, \psi) = 1] \geq 1 - 1/p_1(\kappa); \tag{1}$$

- *Soundness: For every $\sigma^* \notin TagGen(sk, F)$, every interactive machine $P^*$,*

$$\Pr[\langle P^*(F, \sigma^*), V\rangle(pk, \psi) = 1] \leq 1/p_2(\kappa); \tag{2}$$

*where, $p_1(\cdot)$ and $p_2(\cdot)$ are two polynomial*[1]*, and $\kappa$ is a security parameter used in $KeyGen(1^\kappa)$.*

Here, the knowledge soundness can be regarded as a stricter notion of unforgeability for the file block tags. This definition means that the prover can forge file tags by means of a knowledge Extractor [7] if soundness property does not hold. For a private cloud, we are more concerned about the disclosure of private information in the verification process. It is easy to find that data blocks and their tags could be obtained by the verifier in some existing schemes. To solve this problem, we introduce Zero-Knowledge property into IPDP system, as follows:

**Definition 6** (Zero-knowledge)**.** *A proof system for provable data possession problem is computational zero knowledge if there exists a probabilistic polynomial-time algorithm $S^*$ (call a simulator) such that for every probabilistic poly-nomial-time algorithm $D$, for every polynomial $p(\cdot)$, and for all sufficiently large $s$, it holds that*

$$\left| \begin{array}{c} \Pr[D(pk, \psi, S^*(pk, \psi)) = 1] - \\ \Pr[D(pk, \psi, \langle P(F, \sigma), V^*\rangle(pk, \psi)) = 1] \end{array} \right| \leq 1/p(s),$$

*where, $S^*(pk, \psi)$ denotes the output of simulator $S$. That is, for all $\sigma \in TagGen(sk, F)$, the ensembles $S^{\mathcal{O}(F)}(pk, \psi)$ and $\langle P(F, \sigma), V^*\rangle(pk, \psi)$* [2] *are computationally indistinguishable.*

[1]The function $1/p_1(\kappa)$ is called the completeness error, and the function $1/p_2(\kappa)$ is called the soundness error. For non-triviality, we require $1/p_1(\kappa) + 1/p_2(\kappa) \leq 1 - 1/poly(\kappa)$.

[2]The output of the interactive machine $V^*$ after interacting with $P(F, \sigma)$ on common input $(pk, \psi)$.

Actually, zero-knowledge is a property that captures $P$'s robustness against attempts to gain knowledge by interacting with it. For the PDP scheme, we use the zero-knowledge property to the security of data blocks and signature tags.

**Definition 7** (ZK-IPDP). *A IPDP is said as Zero-Knowledge Interactive Provable Data Possession (ZK-IPDP) if the completeness, knowledge soundness, and zero-knowledge property hold.*

### 3.3 Dynamic Provable Data Possession

In order to satisfy the requirement of dynamic scenario, we introduce a dynamic PDP definition on the above definition:

**Definition 8** (Dynamic PDP). *Given a provable data possession $\mathcal{S} = (\mathcal{K}, \mathcal{T}, \mathcal{P})$, it is called as dynamic PDP if this scheme support the following algorithms:*

$\mathcal{U}pdate(sk, \psi, m_i')$**:** *is an algorithm run by the client to update the block of file $m_i$ at the index $i$ by using $sk$, and it returns a new verification metadata $(\psi', \sigma')$;*

$\mathcal{D}elete(sk, \psi, m_i)$**:** *is an algorithm run by the client to delete the block $m_i$ of file at the index $i$ by using $sk$, and it returns a new verification metadata $(\psi')$;*

$\mathcal{I}nsert(sk, \psi, m_i)$**:** *is an algorithm run by the client to insert the block of file $m_i$ at the index $i$ by using $sk$, and it returns a new verification metadata $(\psi', \sigma')$.*

Since the dynamic operations may raise security issues, we state the security for a DPDP scheme using the following game that captures the advantage of the adversary from dynamic operations, as follows:

**Setup:** The challenger runs KeyGen. It gives the adversary the resulting $pk$ and keeps $sk$ to itself;

**Learning:** The adversary issues signature queries $m_1, \cdots, m_q$. To each query $m_i$ the challenger responds by running $Sign$ to generate a signature $\sigma_i$ of $m_i$ and sending $\sigma_i$ to the adversary. These queries may be asked adaptively so that each query $m_i$ may depend on the replies to $m_1, \cdots, m_{i-1}$.

**Output:** Finally the adversary outputs a pair $(m^*, \sigma^*)$. The adversary wins if $\sigma^*$ is a valid signature of $m^*$ according to Verify and $(m^*, \sigma^*)$ is not among the pairs $(m_i, \sigma_i)$ generated during the query phase.

### 3.4 Cooperative Provable Data Possession

In order to prove the integrity of data stored in hybrid clouds, we define a framework for Cooperative Provable Data Possession (CPDP) based on the above-mentioned IPDP:

**Definition 9** (Cooperative-PDP). *A cooperative provable data possession scheme $\mathcal{S}'$ is a collection of two algorithms and an interactive proof system, $\mathcal{S}' = (\mathcal{K}, \mathcal{T}, \mathcal{P})$:*

$\mathcal{K}eyGen(1^\kappa)$**:** *takes a security parameter $\kappa$ as input, and returns a secret key $sk$ or a public-secret keypair $(pk, sk)$;*

$\mathcal{T}agGen(sk, F, \mathcal{P})$**:** *takes as inputs a secret key $sk$, a file $F$, and a set of cloud storage providers $\mathcal{P} = \{P_k\}$, and returns the triples $(\zeta, \psi, \sigma)$, where $\zeta$ is the secret of tags, $\psi = (u, \mathcal{H})$ is a set of verification parameters $u$ and an index hierarchy $\mathcal{H}$ for $F$, $\sigma = \{\sigma^{(k)}\}_{P_k \in \mathcal{P}}$ denotes a set of all tags, $\sigma^{(k)}$ is the tags of the fraction $F^{(k)}$ of $F$ in $P_k$;*

$\mathcal{P}roof(\mathcal{P}, V)$**:** *is a protocol of proof of data possession between the CSPs ($\mathcal{P} = \{P_k\}$) and a verifier ($V$). At the end of the protocol run, $V$ returns a bit $\{0|1\}$ denoting false and true. It includes two cases:*

- *$\langle \sum_{P_k \in \mathcal{P}} P_k(F^{(k)}, \sigma^{(k)}), V(sk, \zeta) \rangle$ is a private proof, where each $P_k$ takes as input a fraction of file $F^{(k)}$ and a set of tags $\sigma^{(k)}$, and $V$ takes as input a secret key $sk$ and a secret of tags $\zeta$;*
- *$\langle \sum_{P_k \in \mathcal{P}} P_k(F^{(k)}, \sigma^{(k)}), V \rangle (pk, \psi)$ is a public proof, where each $P_k$ takes as input a file $F^{(k)}$ and a set of tags $\sigma^{(k)}$, and a public key $pk$ and a set of public parameters $\psi$ is the common input between $P$ and $V$.*

*where, $\sum_{P_k \in \mathcal{P}}$ denotes the cooperative computing in $P_k \in \mathcal{P}$.*

To realize the CPDP, a trivial way is to check the data stored in each cloud one by one. In this way the client needs greater costs of communication and computation. It is obviously unreasonable that these costs are paid out by the clients. Moreover, this way break the advantage of cloud storage: cloud storage should work fully transparent to the end-user and support dynamic scalability.

# 4 ZK-IPDP for Private Cloud

We propose a Zero-Knowledge IPDP scheme to enhance the security of data in private cloud. This scheme is a 3-move interactive proof system, which provides zero-knowledge proof to ensure the confidentiality of secret data and the undeceivability of invalid tags.

## 4.1 Proposed Construction

In our construction, the verification protocol has 3-move structure: commitment, challenge and response, which be showed in Figure 4. This protocol is similar to Schnorr's $\Sigma$ protocol [15], which is a zero-knowledge proof system. By using this property, we ensure the verification process does not reveal anything other than the veracity of the statement of data integrity in a private cloud.

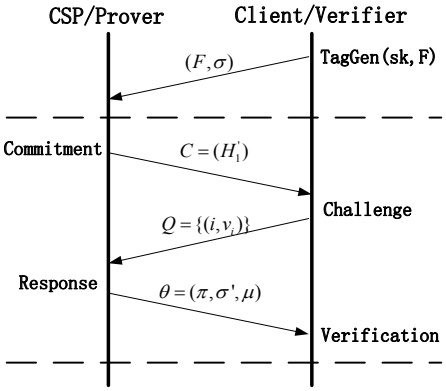


Figure 4: The framework of IPDP model.

We present our IPDP construction in Figure 5. In our scheme, each client holds a secret key $sk$, which can be used to generate the tags of many files. Each processed file will produce a public verification parameter $\psi = (u, \chi)$, where $u = (\xi^{(1)}, u_1, \cdots, u_s)$, $\chi = \{\chi_i\}_{i\in[1,n]}$ is the index table. We define $\chi_i = (B_i||V_i||R_i)$, where $B_i$ is the sequence number of block, $R_i$ is the version number of updates for this block, and $R_i$ is a random integer to avoid collision. The value $\xi^{(1)}$ can be considered as the signature of the secret $\tau_1, \cdots, \tau_s$. Note that, it must assure that the $\psi$'s are different for all processed files. Moreover, it is clear that our scheme admits short responses in verification protocol.

In order to prevent the leakage of the stored data and tags in the verification process, the secret data $\{m_{i,j}\}$ are protected by a random $\lambda_j \in \mathbb{Z}_p$ and the tags $\{\sigma_i\}$ are randomized by a $\gamma \in \mathbb{Z}_p$. Furthermore, the values $\{\lambda_j\}$ and $\gamma$ are protected by the simple commitment methods, i.e., $H_1^{\gamma}$, $H_2^{\gamma}$ and $u_i^{\lambda_i} \in \mathbb{G}$, to avoid the adversary from gaining them.

## 4.2 Security Proof of Construction

The above scheme is an efficient interactive proof system: 1) Completeness: for every available tag $\sigma \in TagGen(sk, F)$ and a random challenge $Q = (i, v_i)_{i\in I}$, the completeness of protocol can be elaborated as follows:

$$\begin{aligned} e(\sigma', h) &= e(\prod_{(i,v_i)\in Q} (\xi_i^{(2)})^{v_i}, h)^{\gamma\cdot\alpha} \cdot e(g,h)^{\gamma\cdot\beta\sum_{(i,v_i)\in Q}\sum_{j=1}^{s}\tau_j\cdot v_i\cdot m_{i,j}} \\ &= e(\prod_{(i,v_i)\in Q} (\xi_i^{(2)})^{v_i}, h^{\alpha\cdot\gamma}) \cdot e(\prod_{j=1}^{s} u_j^{\gamma\cdot\sum_{(i,v_i)\in Q} v_i\cdot m_{i,j}}, h^{\beta}) \\ &= e(\prod_{(i,v_i)\in Q} (\xi_i^{(2)})^{v_i}, h^{\alpha\cdot\gamma}) \cdot \prod_{j=1}^{s} e(u_j^{\mu_j-\lambda_j}, h^{\beta}). \end{aligned}$$

There exists a trivial solution when $v_i = 0$ for all $i \in I$. In this case, the above equation could not determine whether the processed file is available due to $\sigma' = 1$, $\mu_j = \lambda_j$, and $\pi_j = u_j^{\mu_j}$. Hence, the completeness of protocol holds

$$\Pr[\langle P(F,\sigma), V\rangle(pk,\psi) = 1] \geq 1 - 1/p^t, \tag{4}$$

**KeyGen($1^\kappa$):** Let $\mathbb{S} = (p, \mathbb{G}, \mathbb{G}_T, e)$ be a bilinear map group system with randomly selected generators $g, h \in_R \mathbb{G}$, where $\mathbb{G}$, $\mathbb{G}_T$ are two group of large prime order $p$, $|p| = O(\kappa)$. Generate a collision-resistant hash function $H_k(\cdot)$ and chooses a random $\alpha, \beta \in_R \mathbb{Z}_p$ and computes $H_1 = h^\alpha$ and $H_2 = h^\beta \in \mathbb{G}$. Thus, the secret key is $sk = (\alpha, \beta)$ and the public key is $pk = (g, h, H_1, H_2)$.

**TagGen($sk, F$):** Splits the file $F$ into $n \times s$ sectors $F = \{m_{i,j}\} \in \mathbb{Z}_p^{n \times s}$. Chooses $s$ random $\tau_1, \cdots, \tau_s \in \mathbb{Z}_p$ as the secret of this file and computes $u_i = g^{\tau_i} \in \mathbb{G}$ for $i \in [1, s]$ and $\xi^{(1)} = H_\xi(\text{“}Fn\text{”})$, where $\xi = \sum_{i=1}^{s} \tau_i$ and $Fn$ is the file name. Builds an index table $\chi = \{\chi_i\}_{i=1}^{n}$ and fills out the item $\chi_i = (B_i = i, V_i = 1, R_i \in_R \{0,1\}^*)^\dagger$ in $\chi$ for $i \in [1, n]$, then calculates its tag as

$$\sigma_i \quad \leftarrow \quad (\xi_i^{(2)})^\alpha \cdot g^{\sum_{j=1}^{s} \tau_j \cdot m_{i,j} \cdot \beta} \in \mathbb{G}.$$

where $\xi_i^{(2)} = H_{\xi^{(1)}}(\chi_i)$ and $i \in [1, n]$. Finally, sets $u = (\xi^{(1)}, u_1, \cdots, u_s)$ and outputs $\zeta = (\tau_1, \cdots, \tau_s)$, $\psi = (u, \chi)$ to TTP, and $\sigma = (\sigma_1, \cdots, \sigma_n)$ to CSP.

**Proof($P, V$):** This is a 3-move protocol between Prover (CSP) and Verifier (client) with the common input $(pk, \psi)$, which is stored in TTP, as follows:

- **Commitment**$(P \to V)$: $P$ chooses a random $\gamma \in \mathbb{Z}_p$, and sends its commitment $C = (H_1')$ to $V$, where $H_1' = H_1^\gamma$;
- **Challenge**$(P \leftarrow V)$: $V$ chooses a random challenge set $I$ of $t$ indexes along with $t$ random coefficients $v_i \in \mathbb{Z}_p$. Let $Q$ be the set $\{(i, v_i)\}_{i \in I}$ of challenge index coefficient pairs. $V$ sends $Q$ to $P$;
- **Response**$(P \to V)$: $P$ chooses $s$ integers $\lambda_j$ for $j \in [1, s]$ at random, and calculates the response $\theta, \mu$ as

$$\sigma' \leftarrow \prod_{(i,v_i) \in Q} \sigma_i^{\gamma \cdot v_i}, \quad \mu_j \leftarrow \lambda_j + \gamma \cdot \sum_{(i,v_i) \in Q} v_i \cdot m_{i,j},$$

where $\mu = \{\mu_j\}_{j \in [1,s]}$. Let $\pi_j \leftarrow u_j^{\lambda_j} \in \mathbb{G}_1$ for $j \in [1, s]$ and $\pi = \{\pi_j\}_{j \in [1,s]}$, $P$ sends $\theta = (\pi, \sigma', \mu)$ to $V$;

**Verification**: Now the verifier $V$ can check that the response was correctly formed by checking that

$$e(\sigma', h) \stackrel{?}{=} e(\prod_{(i,v_i) \in Q} (\xi_i^{(2)})^{v_i}, H_1') \cdot e(\prod_{j=1}^{s} \frac{u_j^{\mu_j}}{\pi_j}, H_2). \tag{3}$$

Figure 5: The Zero-Knowledge IPDP scheme.

where $t$ is the number of index coefficient pairs in $Q$.

2) Soundness: For every tag $\sigma^* \notin TagGen(sk, F)$, we assume that there exists an interactive machine $P^*$ can pass verification with any verifier $V^*$ with noticeable probability. In order to prove the nonexistence of $P^*$, to the contrary, we make use of $P^*$ to construct a knowledge extractor $\mathcal{M}$, which gets the common input $(pk, \psi)$ and rewindable black-box access to the prover $P^*$ and attempts to break the computation Diffie-Hellman (CDH) assumption in $\mathbb{G}$: given $G, G_1 = G^a, G_2 = G^b \in_R \mathbb{G}$, output $G^{ab} \in \mathbb{G}$. We have the following theorem:

**Lemma 1.** *Our IPDP scheme has $(t, \epsilon')$ knowledge soundness in random oracle and rewindable knowledge extractor model assuming the $(t, \epsilon)$-computation Diffie-Hellman (CDH) assumption holds in the group $\mathbb{G}$ for $\epsilon' \geq \epsilon$.*

The proof of this Lemma is described in Appendix A. It is showed that there exists a polynomial $p(\cdot)$ and all sufficiently large $\kappa$'s, the following condition holds:

$$Pr[\langle P^*(F, \sigma^*), V^* \rangle(pk, \psi) = 1] \leq 1/p(\kappa). \tag{5}$$

Thus, the completeness and soundness are established.

**Lemma 2.** *The verification protocol $(P, V)$ is a computational zero-knowledge system in our IPDP scheme.*

By using the standard simulator $S^*(pk, \psi)$, the proof of this theorem is described in Appendix B. According to Lemma 1 and 2, we have the following theorem:

**Theorem 1.** *Under CDH assumption, our IPDP scheme is a zero-knowledge provable data possession (called as ZK-IPDP) in random oracle and rewindable extractor model.*

# 5 Implementation of Dynamic Operations

To support dynamic data operations, it is necessary for the clients to employ an index table $\chi$ to record the realtime status of the stored files. This kind of index structure is also able to generate the value of Hash function $H_{\xi^{(1)}}(\chi_i)$ in our ZK-IPDP scheme. Some schemes in a dynamic scenario are insecure due to replay attack on the same Hash values. To solve this problem, a simple index table $\chi = \{\chi_i\}$ used in the above construction can be described in Table 3, which includes four columns: No. denotes the real number $i$ of data block $m_i$, $B_i$ is the original number of block, $V_i$ stores the version number of updates for this block, and $R_i$ is a random integer to avoid collision.

Table 3: The index table with random values.

| No. | $B_i$ | $V_i$ | $R_i$ | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | ← Used to head |
| 1 | 1 | 2 | $r'_1$ | ← Update |
| 2 | 2 | 1 | $r_2$ | |
| 3 | 4 | 1 | $r_3$ | ← Delete |
| 4 | 5 | 1 | $r_5$ | |
| 5 | 5 | 2 | $r'_5$ | ← Insert |
| ⋮ | ⋮ | ⋮ | ⋮ | |
| n | n | 1 | $r_n$ | |
| n+1 | n+1 | 1 | $r_{n+1}$ | ← Append |

In order to ensure the security, we require that each $\chi_i$ = "$B_i||V_i||R_i$" is unique in this table. Although the same values of "$B_i||V_i$" may be produced by repeating the insert and delete operations, the random $R_i$ can avoid this collision. An alteration method is to generate an updated random value by $R'_i \leftarrow H_{R_i}(\sum_{j=1}^{s} m'_{i,j})$, where the initial value is $R_i \leftarrow H_{\xi^{(1)}}(\sum_{j=1}^{s} m_{i,j})$ and $m_i = \{m_{i,j}\}$ denotes the $i$-th data block. We show a simple example to describe the changes of different operations in Table 3. According to the structure of the index table, we propose a simple method to provide dynamic data modification in Figure 6.

**Fact 1.** *There no exists two same records $\chi_i$ = "$B_i||V_i||R_i$" in the index table $\chi$, that is, $\chi_i \neq \chi_j$ and $i \neq j$, if $R_i \neq R'_j$ for $\forall i, j$.*

The other method is to use the value of $R_i$ to record the maximum of $V_k$ and $R_k$ in all records $\chi_k$ with $B_k = B_i$, define the initial value $R_i \leftarrow 1$ and its updated value $R'_i \leftarrow \max_{B_i=B_k}\{V_k, R_k\} + 1$. In this case, the Hash value is defined as $H_{\xi^{(1)}}(\chi_i)$, where $\chi_i$ = ("$B_i||V_i$"). We show an example to describe the changes of different operations in Table 4. Assume that the size of each record $\chi_i$ is 32-Bits and the size of each block is $4K$-Bytes, in which $|B_i|$ = 11-Bits, $|V_i| = |R_i|$ = 10-Bits. Such that the maximum number of data blocks is about $n = 2^{21}$ and the maximum length of file is about $8T$-Bytes in this case. It is more efficient than the above case in which the random value $R_i$ must been enough length to resilient Birthday attack.

**Fact 2.** *There no exists two same records $\chi_i$ = "$B_i||V_i$" in the index table $\chi$, that is, $\chi_i \neq \chi_j$ and $i \neq j$, if any operations update the value of $R_k$ in all records $\{\chi_k\}_{B_k=B_i}$ with the same $B_i$.*

Note that, all tags and the index table should be renewed if only any item in $\chi$ is full. Of course, we can replace the sequent lists by dynamically linked lists to improve the efficiency of updating index table.

**Theorem 2.** *Under the CDH assumption in a cyclic group $\mathbb{G} \in \mathbb{S}$, the ZK-IPDP scheme is a DPDP scheme to resist the attack of forging the tags for dynamic data operations with random oracle model if each $\chi_i$ is unique in $\chi$.*

The proof of this theorem is described in Appendix C. It is easy to prove the each $\chi_i$ is unique in $\chi$ in the above algorithms. Further, we omit the discuss of the head and tail index items in $\chi$ and they is easy to implementation.

**Update**$(sk, \psi, m_i')$**:** modifies the version number by $V_i \leftarrow \max_{B_i=B_j}\{V_j\}+1$ and chooses a new $R_i$ in $\chi_i \in \chi$ to get a new $\psi'$; computes the new hash $\xi_i^{(2)} = H_{\xi^{(1)}}(\text{“}B_i||V_i||R_i\text{”})$; by using $sk$, computes $\sigma_i' = (\xi^{(2)})^{\alpha} \cdot (\prod_{j=1}^{s} u_j^{m_{i,j}'})^{\beta}$, where $u = \{u_j\} \in \psi$, finally outputs $(\psi', \sigma_i', m_i')$.

**Delete**$(sk, \psi, m_i)$**:** computes the original $\sigma_i$ by $m_i$ and computes the new hash $\xi_i^{(2)} = H_{\xi^{(1)}}(\text{“}B_i||0||R_i\text{”})$ and $\sigma_i' = (\xi_i^{(2)})^{\alpha}$ by $sk$; delete $i$-th record to get a new $\psi'$; finally outputs $(\psi', \sigma_i, \sigma_i')$.

**Insert**$(sk, \psi, m_i')$**:** inserts a new record in $i$-th position of the index table $\chi \in \psi$, and the other records move backward in order; modifies $B_i \leftarrow B_{i-1}$, $V_i \leftarrow \max_{B_i=B_j}\{V_j\}+1$, and a random $R_i$ in $\chi_i \in \chi$ to get a new $\psi'$; computes the new hash $\xi_i^{(2)} = H_{\xi^{(1)}}(\text{“}B_i||V_i||R_i\text{”})$ and $\sigma_i' = (\xi^{(2)})^{\alpha} \cdot (\prod_{j=1}^{s} u_j^{m_{i,j}'})^{\beta}$, where $u = \{u_j\} \in \psi$, finally outputs $(\psi', \sigma_i', m_i')$.

The client send the above result to cloud store provider $P$ via secret channel. For Update or Insert operations, $P$ must check the following equation for $(\psi', \sigma_i', m_i')$:

$$e(\sigma_i', h) \stackrel{?}{=} e(\xi_i^{(2)}, H_1) \cdot e(\prod_{j=1}^{s} u_j^{m_{i,j}'}, H_2).$$

For Delete operation, $P$ must check whether $\sigma_i$ is equal to the stored $\sigma_i$ and $e(\sigma_i', h) \stackrel{?}{=} e(H_{\xi^{(1)}}(\text{“}B_i||0||R_i\text{”}), H_1)$. Furthermore, $TTP$ must replace $\psi$ by the new $\psi'$ and check the completeness of $\chi \in \psi$.

Figure 6: The dynamic scheme for ZK-IPDP.

Table 4: The index table with max function.

| No. | $B_i$ | $V_i$ | $R_i$ | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | ← Used to head |
| 1 | 1 | 2 | 2 | ← Update |
| 2 | 2 | 1 | 1 | |
| 3 | 4 | 1 | 1 | ← Delete |
| 4 | 5 | 1 | 2 | |
| 5 | 5 | 2 | 2 | ← Insert |
| ⋮ | ⋮ | ⋮ | ⋮ | |
| n | n | 1 | 1 | |
| n+1 | n+1 | 1 | 1 | ← Append |

# 6 ZK-CPDP for Hybrid Cloud

Based on the above ZK-PDP protocol, we propose a new Cooperative Provable Data Possession scheme for hybrid cloud, which can support the user's dynamic behavior and private protection. Concretely speaking, this scheme can meet the following requirements:

- Conceal the details of the storage, which the users do not need to know in the verification process;
- Ensure that the verification does not reveal any information, it is of particular importance for sensitive data;

## 6.1 Protocol Settings and Index Hierarchy

A representative architecture for data storage on hybrid cloud is illustrated in Figure 7. This architecture is a hierarchical structure $\mathcal{H}$ on three layers to represent the relationship among all blocks for stored resources. Three layers can be described as follows:

- First-Layer (*Express Layer*): offer an abstract representation of the stored resources;
- Second-Layer (*Service Layer*): immediately offer and manage cloud storage services;

- Third-Layer (*Storage Layer*): practicably realize data storage on many physical devices;

This kind of architecture is a nature representation of file storage. We make use of this simple hierarchy to organize multiple CSP services, which involves private clouds or public clouds, by shading the differences between these clouds. In Figure 7, the resources in Express Layer are split and stored into three CSPs , that have different colors, in Service Layer. In turn, each CSP fragmented and stored the assigned data into the storage servers in Storage Layer. We make use of colors to distinguish different CSPs, and the denotation of Storage Layer is the same as in Figure 3. Moreover, we follow the logical order of the data blocks to organize the Storage Layer. This architecture could provide some special functions for data storage and management. For example, there may exist overlap among data blocks (as shown in dashed line) and skipping (as shown on a non-continuous color). But these functions would increase the complexity of storage management.

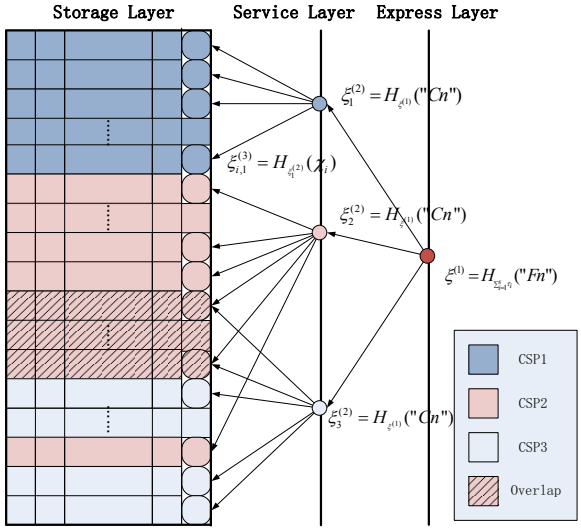


Figure 7: The architecture of CPDP model.

We make use of this architecture to construct a new Index Hierarchy $\mathcal{H}$, which is used to replace the hash function in the ZK-IPDP schemes, as follows:

- Express layer: given $s$ random $\{\tau_i\}_{i=1}^{s}$ and the file name $Fn$, sets $\xi^{(1)} = H_{\sum_{i=1}^{s}\tau_i}(\text{“}Fn\text{”})$ and makes it public for verification but makes $\{\tau_i\}_{i=1}^{s}$ secret;
- Service layer: given the $\xi^{(1)}$ and the cloud name $Cn$, sets $\xi_k^{(2)} = H_{\xi^{(1)}}(\text{“}Cn\text{”})$;
- Storage layer: given the $\xi^{(2)}$, a block number $i$, and its index record $\chi_i = \text{“}B_i||V_i||R_i\text{”}$, sets $\xi_{i,j}^{(3)} = H_{\xi_k^{(2)}}(\chi_i)$.

To meet this change, the index table $\chi$ in the ZK-IPDP scheme needs to increase a new column $C_i$ to record the serial number of CSP, that stores the $i$-th block. By using this structure, it is obvious that our CPDP scheme can also support dynamic data operations.

## 6.2 Cooperative Provable Data Possession

According to the above architecture, four different network entities can be identified as follows: the verifier (V), trusted third party (TTP), the organizer (O), and some cloud service providers in hybrid cloud $\mathcal{P} = \{P_i\}_{i\in[1,c]}$. The organizer is an entity that directly contacts with the verifier, moreover it can initiate and organize the verification process. Often, the organizer is an independent server or a certain CSP in $\mathcal{P}$. In our scheme, the verification is performed by a 5-move interactive proof protocol showed in Figure 8, as follows: 1) the organizer initiates the protocol and sends the commitment to the verifier; 2) the verifier returns a challenge set of random index-coefficient pairs $Q$ to the organizer; 3) the organizer relays them into each $P_i$ in $C$ according to the exact position of each data block; 4) each $P_i$ returns its response of challenge to the organizer; 5) the organizer synthesizes a final response from these responses and sends it to the verifier. The above process would guarantee that the verifier accesses files without knowing on what CSPs or in what geographical locations their files reside.

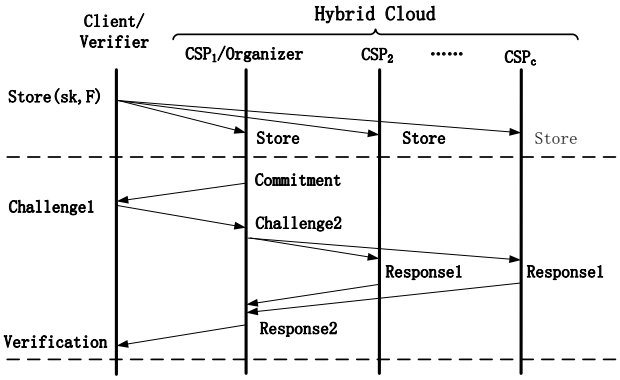


Figure 8: The framework of CPDP model.

According to the above framework of protocol, we propose the first CPDP construction for the hybrid clouds based on the above-mentioned ZK-IPDP scheme, which is showed in Figure 9. In contrast to single (private or public) cloud surrounding, the protocol considers the interaction of a verifier with more than one CSP in hybrid clouds, in which CSP need not interact with each other during the verification process, but an organizer is used to organize and manage all CSPs. For this idea, our construction differs from the ZK-IPDP scheme in two respects:

1) In the commitment stage, the organizer generates a random $\gamma \in_R \mathbb{Z}_p$ and returns it to the verifier, but the CSPs do not obtain the value of $\gamma$. So that it requires the organizer can compute the final $\sigma'$ by using this $\gamma$ and the $\sigma'_k$ received from CSPs. In order to ensure the security of data tags, we use the ElGamal encryption to encrypt the combination of tags $\prod_{(i,v_i)\in Q_k} \sigma_i^{v_i}$, that is, for $sk = s \in \mathbb{Z}_p$ and $pk = (g, S = g^s) \in \mathbb{G}^2$, the cipher of the message $m$ is $\mathcal{C} = (\mathcal{C}_1 = g^r, \mathcal{C}_2 = m \cdot S^r)$ and its decryption is preformed by $m = \mathcal{C}_2 \cdot \mathcal{C}_1^{-s}$. Thus we have the following equation

$$\begin{aligned} \sigma' &= (\prod_{P_k\in\mathcal{P}} \frac{\sigma'_k}{R^s})^\gamma = (\prod_{P_k\in\mathcal{P}} \frac{S^{r_k} \cdot \prod_{(i,v_i)\in Q_k} \sigma_i^{v_i}}{R_k^s})^\gamma \\ &= (\prod_{P_k\in\mathcal{P}} \prod_{(i,v_i)\in Q_k} \sigma_i^{v_i})^\gamma = \prod_{(i,v_i)\in Q} \sigma_i^{v_i\cdot\gamma}. \end{aligned}$$

2) Because of the homomorphic property, the responses computed from the CSPs in hybrid clouds can be combined into a single final response, as follows: given a set of $\theta_k = (\pi_k, \sigma'_k, \mu_k, R_k)$ received from $P_k$. Let $\lambda_j = \sum_{P_k\in\mathcal{P}} \lambda_{j,k}$, the organizer can compute the following equation:

$$\begin{aligned} \mu_j &= \sum\nolimits_{P_k\in\mathcal{P}} \mu_{j,k} = \sum\nolimits_{P_k\in\mathcal{P}} (\lambda_{j,k} + \sum\nolimits_{(i,v_i)\in Q_k} v_i \cdot m_{i,j}) \\ &= \sum\nolimits_{P_k\in\mathcal{P}} \lambda_{j,k} + \sum\nolimits_{P_k\in\mathcal{P}} \sum\nolimits_{(i,v_i)\in Q_k} v_i \cdot m_{i,j} \\ &= \lambda_j + \sum\nolimits_{(i,v_i)\in Q} v_i \cdot m_{i,j}. \end{aligned}$$

The corresponding commitment of $\lambda_j$ is also computed by

$$\pi_j = \prod\nolimits_{P_k\in\mathcal{P}} \pi_{j,k} = \prod\nolimits_{P_k\in\mathcal{P}} u_j^{\lambda_{j,k}} = u_j^{\sum_{P_k\in\mathcal{P}} \lambda_{j,k}} = u_j^{\lambda_j}.$$

It is obvious that the final response $\theta$ received by the verifier in CPDP is the same as that in the ZK-IPDP scheme in terms of the above analysis. This means that the verifier cannot feel the difference between them. Similarly to implementation of dynamic operations in the ZK-IPDP scheme, this construction can also support the probable updates to the stored data in hybrid clouds.

Two response algorithms, Response1 and Response2, comprise a homomorphic verifiable responses: Given two responses $\theta_i$ and $\theta_j$ for two challenges $Q_i$ and $Q_j$ from two CSPs, i.e., $\theta_i = Response1(Q_i, \{m_k\}_{k\in I_i}, \{\sigma_k\}_{k\in I_i})$, there exists an efficient algorithm to combine them into a response $\theta$ corresponding to the sum of the challenges $Q_i \bigcup Q_j$, i.e.,

$$\begin{aligned} \theta &= Response2(\theta_i, \theta_j) \\ &= Response1(Q_i \bigcup Q_j, \{m_k\}_{k\in I_i \bigcup I_j}, \{\sigma_k\}_{k\in I_i \bigcup I_j}). \end{aligned}$$

**KeyGen($1^\kappa$):** Let $\mathbb{S} = (p, \mathbb{G}, \mathbb{G}_T, e)$ be a bilinear map group system with randomly selected generators $g, h \in \mathbb{G}$, where $\mathbb{G}, \mathbb{G}_T$ are two bilinear group of large prime order $p$, $|p| = O(\kappa)$. Makes a hash function $H_k(\cdot)$ public.

1. For a CSP, chooses a random $s \in_R \mathbb{Z}_p$ and computes $S = g^s \in \mathbb{G}$. Thus, $sk_p = s$ and $pk_p = (g, S)$.
2. For a user, chooses two random $\alpha, \beta \in_R \mathbb{Z}_p$ and sets $sk_u = (\alpha, \beta)$ and $pk_u = (g, h, H_1 = h^\alpha, H_2 = h^\beta)$.

**TagGen($sk, F, \mathcal{P}$):** Splits the file $F$ into $n \times s$ sectors $\{m_{i,j}\}_{i\in[1,n],j\in[1,s]} \in \mathbb{Z}_p^{n\times s}$. Chooses $s$ random $\tau_1, \cdots, \tau_s \in \mathbb{Z}_p$ as the secret of this file and computes $u_i = g^{\tau_i} \in \mathbb{G}$ for $i \in [1, s]$. Constructs the index table $\chi = \{\chi_i\}_{i=1}^n$ and fills out out the record $\chi_i = (C_i = k, B_i = i, V_i = 1, R_i \in_R \{0,1\}^*)$ in $\chi$ for $i \in [1, n]$, then calculates the tag for each block $m_i$ as

$$\xi^{(1)} = H_{\sum_{i=1}^s \tau_i}(\text{“}Fn\text{”}), \quad \xi_k^{(2)} = H_{\xi^{(1)}}(\text{“}Cn\text{”}), \quad \xi_{i,k}^{(3)} = H_{\xi_k^{(2)}}(\chi_i), \quad \sigma_{i,k} \leftarrow (\xi_{i,k}^{(3)})^\alpha \cdot (\prod_{j=1}^s u_j^{m_{i,j}})^\beta \in \mathbb{G},$$

where $Fn$ is the file name and $Cn$ is the CSP name. And then outputs $\zeta = (\tau_1, \cdots, \tau_s)$ to the file owner, $\psi = ((u_1, \cdots, u_s), \xi^{(1)})$ to TTP, and $\sigma_k = \{\sigma_{i,j}\}_{\forall j=k}$ to $P_k \in \mathcal{P}$.

**Proof($\mathcal{P}, V$):** Let $O$ be an organizer in hybrid cloud $\mathcal{P} = \{P_i\}_{i\in[1,c]}$. This is a 5-move protocol between Provers ($\mathcal{P}$) and Verifier ($V$) with the common input $(pk, \psi)$, which is stored in TTP. The protocol can be described as:

1. **Commitment**($O \to V$): $O$ chooses a random $\gamma \in \mathbb{Z}_p$ and generates $H_1' = H_1^\gamma$, $H_2' = H_2^\gamma$, sends $c = (H_1', H_2')$ to $V$;
2. **Challenge1**($O \leftarrow V$): $V$ chooses a set of challenge index coefficient pairs $Q = \{(i, v_i)\}_{i\in I}$ and sends $Q$ to $O$;
3. **Challenge2**($\mathcal{P} \leftarrow O$): $O$ forwards $Q_k = \{(i, v_i)\}_{m_i\in P_k} \in Q$ along to each $P_k$ in $\mathcal{P}$;
4. **Response1**($\mathcal{P} \to O$): $P_k$ chooses a random $r_k \in \mathbb{Z}_p$ and $s$ random $\lambda_{j,k} \in \mathbb{Z}_p$ for $j \in [1, s]$, and calculates the response

$$\sigma_k' \leftarrow S^{r_k} \cdot \prod_{(i,v_i)\in Q_k} \sigma_i^{v_i} \in \mathbb{G}, \qquad \mu_{j,k} \leftarrow \lambda_{j,k} + \sum_{(i,v_i)\in Q_k} v_i \cdot m_{i,j} \in \mathbb{Z}_p, \qquad \pi_{j,k} \leftarrow u_j^{\lambda_{j,k}} \in \mathbb{G},$$

   where $\mu_k = \{\mu_{j,k}\}_{j\in[1,s]}$ and $\pi_k = \{\pi_{j,k}\}_{j\in[1,s]}$. Let $R_k \leftarrow g^{r_k} \in \mathbb{G}$, each $P_k$ sends $\theta_k = (\pi_k, \sigma_k', \mu_k, R_k)$ to $O$;
5. **Response2**($O \to V$): After receiving all responses from $\{P_i\}_{i\in[1,c]}$, $O$ aggregates $\{\theta_k\}_{P_k\in\mathcal{P}}$ into a common $\theta$ as

$$\sigma' \leftarrow (\prod_{P_k\in\mathcal{P}} \sigma_k' \cdot R_k^{-s})^\gamma, \qquad \mu_j \leftarrow \sum_{P_k\in\mathcal{P}} \mu_{j,k}, \qquad \pi_j \leftarrow \prod_{P_k\in\mathcal{P}} \pi_{j,k}.$$

   Let $\mu = \{\mu_j\}_{j\in[1,s]}$ and $\pi = \{\pi_i\}_{i\in[1,s]}$. $O$ sends $\theta = (\pi, \sigma', \mu)$ to $V$.

**Verification**: Now the verifier $V$ can check that the response was correctly formed by checking that

$$e(\sigma', h) \stackrel{?}{=} e(\prod_{(i,v_i)\in Q} H_{\xi_k^{(2)}}(\chi_i)^{v_i}, H_1') \cdot e(\prod_{j=1}^s u_j^{\mu_j} \cdot {\pi_j}^{-1}, H_2').$$

Figure 9: Cooperative Provable Data Possession for Hybrid Cloud.

More importantly, the HVR is a pair of values $\theta = (\pi, \sigma, \mu)$, which have a constant size even for the different challenges.

We give a brief security analysis of this construction. This construction is in essence a Multi-Prover Zero-Knowledge Proof (MPZK) protocol, which can be considered as an extension of the notion of an interactive proof system. Since this construction is directly derived from the ZK-IPDP scheme, zero-knowledge and soundness properties are still remained in this construction as follows:

**Theorem 3.** *Under the CDH assumption, our CPDP construction is a zero-knowledge cooperative provable data possession (called as ZK-CPDP) in random oracle and rewindable extractor model.*

# 7 Performances and System

## 7.1 Performances Analysis

We first analyze the computation cost of IPDP and CPDP schemes. For clearance, we give Table 5 to present them. In this table, we use $[E]$ to denote the computation costs of an exponent operation in $\mathbb{G}$, namely, $g^x$, where $x$ is a positive integer in $\mathbb{Z}_p$ and $g \in \mathbb{G}$ or $\mathbb{G}_T$. We neglect the computation costs of algebraic operations and simple modular arithmetic operations because they run fast enough [3]. The more complex operation are the computation of a bilinear map $e(\cdot,\cdot)$ between two elliptic points (denoted as $[B]$).

Table 5: The performance analysis for our schemes.

| | IPDP | CPDP |
|---|---|---|
| KeyGen | $2[E]$ | $3[E]$ |
| TagGen | $(2n+s)[E]$ | $(2n+s)[E]$ |
| Proof(P) | $(t+s+1)[E]$ | $(t+cs+2c+3)[E]$ |
| Proof(V) | $3[B]+(t+s)[E]$ | $3[B]+(t+s)[E]$ |
| Update | $s+1[E]$ $[3[B]+s[E]]$ | $s+1[E]$ $[3[B]+s[E]]$ |
| Delete | $s+2[E]$ $[2[B]]$ | $s+2[E]$ $[2[B]]$ |
| Insert | $s+1[E]$ $[3[B]+s[E]]$ | $s+1[E]$ $[3[B]+s[E]]$ |

Secondly, we analyze the storage and communication cost of our schemes. We define the bilinear pairing takes the form $e : E(\mathbb{F}_{p^m}) \times E(\mathbb{F}_{p^{km}}) \to \mathbb{F}^*_{p^{km}}$ (we give here the definition from [4, 11]), where $p$ is a prime, $m$ is a positive integer, and $k$ is the embedding degree (or security multiplier). In this case, we utilize asymmetric pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ to replace symmetric pairing in the original schemes.

Without loss of generality, let the security parameter $\kappa$ be 80-bits, we need the elliptic curve domain parameters over $\mathbb{F}_p$ with $|p| = 160$-bits and $m = 1$ in our experiments. This means that the length of integer is $l_0 = 2\kappa$ in $\mathbb{Z}_p$. Similarly, we have $l_1 = 4\kappa$ in $\mathbb{G}_1$, $l_2 = 24\kappa$ in $\mathbb{G}_2$, and $l_T = 24\kappa$ in $\mathbb{G}_\mathbb{T}$ for the embedding degree $k = 6$. Based on these definitions, we decribe storage or communication costs in Table 6.

Table 6: The storage/communication overhead.

| Algorithm | | IPDP | CPDP |
|---|---|---|---|
| KeyGen | Client | $2l_0$ | $2l_0$ |
| | CSP | | $l_0$ |
| TagGen | CSP | $nsl_0 + nl_1$ | $(nsl_0 + nl_1)/c$ |
| | TTP | $sl_1 + (n+1)l_0$ | $sl_1 + (n+1)l_0$ |
| Proof | Commit | $2l_2$ | $2l_2$ |
| | Challenge | $2tl_0$ | $2tl_0$ |
| | | | $2tl_0/c$ |
| | Response | $sl_0 + (s+1)l_1$ | $(sl_0 + (s+2)l_1)c$ |
| | | | $sl_0 + (s+1)l_1$ |

Hence, for IPDP scheme, the storage overhead of a file with $size(f) = 1M$-Bytes is $store(f) = n \cdot s \cdot l_0 + n \cdot l_1 = 1.04M$-Bytes for $n = 10^3$ and $s = 50$. The storage overhead of its index table $\chi$ is $n \cdot l_0 = 20K$-Bytes. We define the overhead rate as $\lambda = \frac{store(f)}{size(f)} - 1 = \frac{l_1}{s \cdot l_0}$ and it should therefore be kept as low as possible in order to minimize storage in cloud storage providers. It is obvious that a higher $s$ means more lower storage. Furthermore, in the verification protocol, the communication overhead of challenge is $2t \cdot l_0 = 40 \cdot t$-Bytes in terms of $t$, but its response have a constant-size communication overhead $s \cdot l_0 + (s+1) \cdot l_1 \approx 3K$-Bytes for the different-size files.

## 7.2 Optimization of Parameters

**Theorem 4.** *Given a file with $sz = n \cdot s$ sectors and the probability $\rho$ of sector corruption, the detection probability of IPDP has $P(sz, \rho, \omega) \geq 1 - (1-\rho)^{sz \cdot \omega}$, where $\omega$ denotes the sampling probability in the verification protocol. For a hybrid cloud $\mathcal{P}$, the detection probability of CPDP has $P(sz, \{\rho_k, r_k\}_{P_k \in \mathcal{P}}, \omega) \geq 1 - \prod_{P_k \in \mathcal{P}} (1-\rho_k)^{sz \cdot r_k \cdot \omega}$, where $r_k$ denotes the proportion of data blocks in the $k$-th CSP, $\rho_k$ denotes the probability of file corruption in the $k$-th CSP.*

*Proof.* For a uniform random verification in the IPDP scheme, $P_b \geq 1-(1-\rho)^s$ is the probability of block corruption with $s$ sectors. For a challenge with $t = n \cdot \omega$ index-coefficient pairs, the verifier can detect block errors with probability $P \geq 1-(1-P_b)^t \geq 1-((1-\rho)^s)^{n\cdot\omega} = 1-(1-\rho)^{sz\cdot\omega}$, where $sz = n \cdot s$.

In the same way, the verifier can detect this server misbehavior with probability

$$\begin{aligned} P(sz, \{\rho_k, r_k\}_{P_k\in\mathcal{P}}, \omega) &\geq 1 - \prod_{P_k\in\mathcal{P}} ((1-\rho_k)^s)^{n\cdot r_k\cdot\omega} \\ &= 1 - \prod_{P_k\in\mathcal{P}} (1-\rho_k)^{sz\cdot r_k\cdot\omega}, \end{aligned}$$

where $r_k \cdot \omega$ denotes the possible number of blocks queried by the verifier in the $k$-th CSP. □

**Theorem 5.** *Given the detection probability $P$ and the probability of sector corruption $\rho$, the optimal value of $s$ can be computed by*

$$\min_{s\in\mathbb{N}} \left\{ \frac{\log(1-P)}{\log(1-\rho)} \frac{a}{s} + b \cdot s \right\},$$

*where $a \cdot t + b \cdot s$ denotes the computational cost of verification protocol in IPDP, and $a, b \in \mathbb{R}$.*

*Proof.* Let $sz = n \cdot s = size(f)/l_0$. According to Theorem 4, the sampling probability holds

$$w \geq \frac{\log(1-P)}{sz \cdot \log(1-\rho)} = \frac{\log(1-P)}{n \cdot s \cdot \log(1-\rho)}.$$

In order to minimize the computational cost, we have

$$\begin{aligned} \min_{s\in\mathbb{N}} \{a \cdot t + b \cdot s\} &= \min_{s\in\mathbb{N}} \{a \cdot n \cdot w + b \cdot s\} \\ &\geq \min_{s\in\mathbb{N}} \left\{ \frac{\log(1-P)}{\log(1-\rho)} \frac{a}{s} + b \cdot s \right\}. \end{aligned}$$

Since $\frac{a}{s}$ is a monotone decreasing function and $b \cdot s$ is a monotone increasing function for $s > 0$, there exists an optimal value of $s \in \mathbb{N}$ in the above equation. □

Remark. The optimal value of $s$ is unrelated to a certain file from Theorem 4.

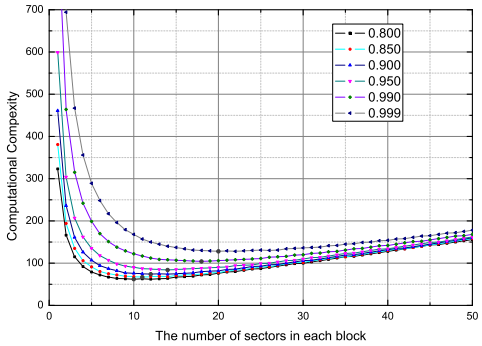


Figure 10: The relationship between computational costs and the number of sectors in each block.

For instance, we assume the probability of sector corruption is the constant value $\rho = 0.01$. We set the detection probability $P$ ranged from 0.8 to 1, e.g., $P = \{0.8, 0.85, 0.9, 0.95, 0.99, 0.999\}$. In terms of Table 5, the computational cost of the verifier can be simplified to $t + s$, so that we can observe the computational costs under the different $s$ and $P$ in Figure 10 by using Theorem 5. When $s$ is less than the optimal value, the computational cost decreases evidently with increase of $s$, and then it raises when $s$ is more than the optimal value.

More accurately, we show the influence of parameters, $sz \cdot w$, $s$, and $t$, under the different detection probabilities in Table 7. It is easy to see that the computational costs raise with increase of $P$. Moreover, we can make sure the sampling number of challenge by the following Lemma:

Table 7: The influence of parameters under the different detection probabilities $P$ ($\rho = 0.01$).

| P | 0.8 | 0.85 | 0.9 | 0.95 | 0.99 | 0.999 |
|---|---|---|---|---|---|---|
| $sz \cdot w$ | 160.14 | 188.76 | 229.11 | 298.07 | 458.21 | 687.31 |
| $s$ | 12 | 14 | 14 | 14 | 18 | 25 |
| $t$ | 14 | 14 | 17 | 22 | 26 | 28 |

**Lemma 3.** *Given the detection probability $P$, the probability of sector corruption $\rho$, and the number of sectors in each block $s$, the sampling number of verification protocol is a constant $t = n \cdot w \geq \frac{\log(1-P)}{s \cdot \log(1-\rho)}$ for the different files.*

Finally, we observe the change of $s$ under the different $\rho$ and $P$. The experimental results are showed in Table 8. It is obvious that the optimal value of $s$ raises with increase of $P$ and with decrease of $\rho$. We choose the optimal value of $s$ on the basis of practical settings and system requisition. For NTFS format, we suggest that the value of $s$ is 200 and the size of block is 4K-Bytes, which is the same as the default size of cluster when the file size is less than 16TB in NTFS. In this case, the value of $s$ ensures that the extra storage doesn't exceed 1% in storage servers.

Table 8: The influence of parameter $s$ under the different probabilities of sector corruption $\rho$ and the different detection probabilities $P$.

| | 0.1 | 0.01 | 0.001 | 0.0001 |
|---|---|---|---|---|
| 0.8 | 3 | 12 | 37 | 118 |
| 0.85 | 3 | 14 | 40 | 136 |
| 0.9 | 4 | 14 | 44 | 150 |
| 0.95 | 4 | 14 | 53 | 166 |
| 0.99 | 6 | 18 | 61 | 207 |
| 0.999 | 7 | 25 | 79 | 249 |

**Theorem 6.** *Given a detection probability $P$ and hybrid cloud $\mathcal{P} = \{P_k\}$, the optimal value of $s$ can be computed by*

$$\min_{s \in \mathbb{N}} \left\{ \frac{\log(1-P)}{\sum_{P_k \in \mathcal{P}} r_k \cdot \log(1-\rho_k)} \cdot \frac{a}{s} + b \cdot s \right\}, \tag{6}$$

*where $r_k$ denotes the proportion of data blocks in the $k$-th CSP, $\rho_k$ denotes the probability of file corruption in the $k$-th CSP, $a \cdot t + b \cdot s$ denotes the computational cost of verification protocol in CPDP, and $a, b \in \mathbb{R}$.*

### 7.3 Implementation and Experimental Results

We have implemented Hierarchy PDP scheme in order to test the effect of dispersal secret data on private cloud and hybrid cloud. The code is written in C++ and experiments were run on an Intel Core 2 processor running at 2.16 GHz. All cryptographic operations utilize the QT and bilinear cryptographic library.

We evaluate the performance of our ZK-IPDP and ZK-CPDP schemes in terms of computational overhead, due to these two schemes have constant-size communication overhead. For sake of comparison, our experiments use the same scenario as the above analysis, where a fix-size file is used to generate the tags and prove possession under the different $s$. For a 150K-Bytes file, the computational overheads of the verification protocol are showed in Figure 11(a) when the value of $s$ is ranged from 1 to 50 and the size of sector is 20-Bytes. It is obvious that the experiment curve is consistent with the above analysis in Figure 10. The optimal values of $s$ are consistent with the above analysis. The computational overheads of the tag generation are also showed in Figure 11(b) in the same case. The results indicate that the overhead is reduced when the values of $s$ is increased. Our demo software is showed in Figure 12 and is downloaded in our web (http://sefcom.asu.edu/cloud/release.rar).

## 8 Conclusions

In this paper, we addressed the construction of PDP scheme on hybrid clouds. Based on interactive zero-knowledge proof and multi-prover zero-knowledge proof, we proposed an interactive PDP (IPDP) scheme to support zero-knowledge

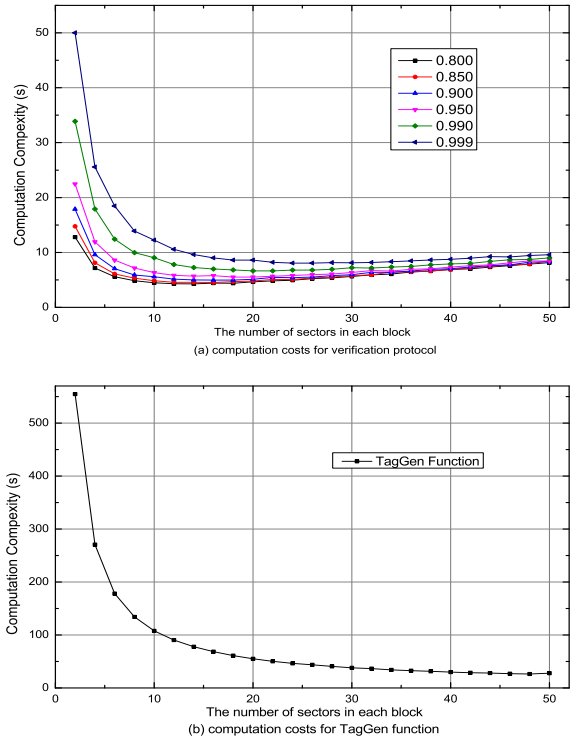


Figure 11: The experiment results of the different $s$ for a 150K-Bytes file ($\rho = 0.01$ and $P = 0.99$).

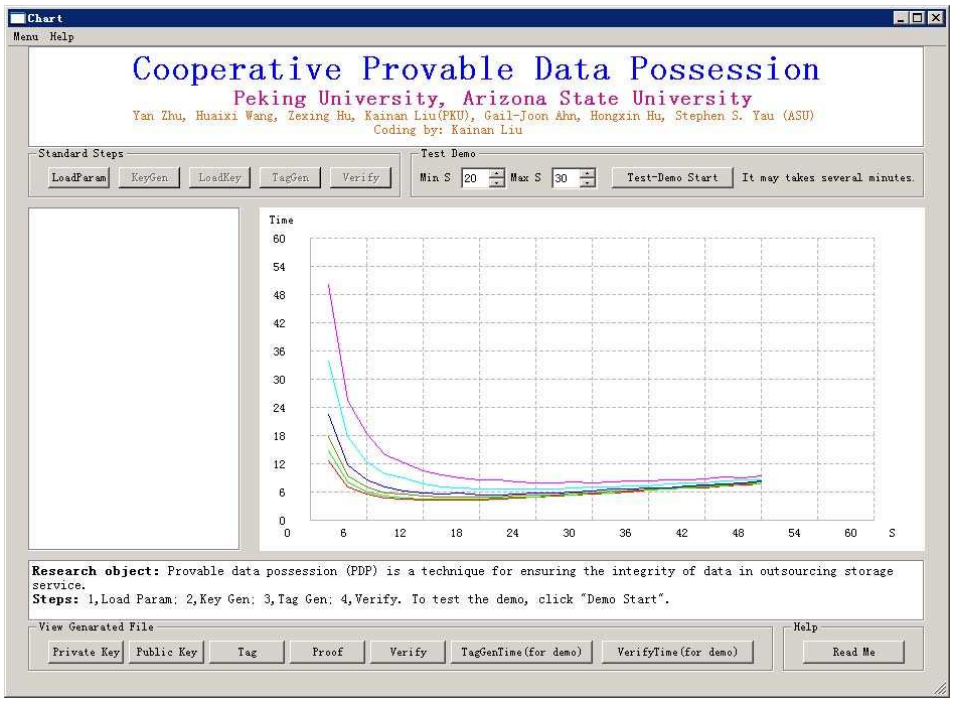


Figure 12: The demo software for our schemes (download it in our web).

property and dynamic scalability, as well as a cooperative PDP (CPDP) solution on hybrid clouds using three-layered index hierarchy. Our experiments showed that our schemes require a small, constant amount of overhead, which minimizes computation and communication complexity while presenting an efficient method for selecting the optimal

number of sectors in each block to minimize the computation costs of clients and storage service providers.

# References


[1] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. X. Song. Provable data possession at untrusted stores. In *ACM Conference on Computer and Communications Security*, pages 598–609, 2007.

[2] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks, SecureComm*, pages 1–10, 2008.

[3] P. S. L. M. Barreto, S. D. Galbraith, C. O'Eigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. *Des. Codes Cryptography*, 42(3):239–271, 2007.

[4] J.-L. Beuchat, N. Brisebarre, J. Detrey, and E. Okamoto. Arithmetic operators for pairing-based cryptography. In *CHES*, pages 239–255, 2007.

[5] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology (CRYPTO'2001)*, volume 2139 of LNCS, pages 213–229, 2001.

[6] K. D. Bowers, A. Juels, and A. Oprea. Hail: a high-availability and integrity layer for cloud storage. In *ACM Conference on Computer and Communications Security*, pages 187–198, 2009.

[7] R. Cramer, I. Damgård, and P. D. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In *Public Key Cryptography*, pages 354–373, 2000.

[8] Y. Dodis, S. P. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *TCC*, pages 109–127, 2009.

[9] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. In *ACM Conference on Computer and Communications Security*, pages 213–222, 2009.

[10] O. Goldreich. *Foundations of Cryptography: Basic Tools*, volume Basic Tools. Cambridge University Press, 2001.

[11] H. Hu, L. Hu, and D. Feng. On a class of pseudorandom sequences from elliptic curves over finite fields. *IEEE Transactions on Information Theory*, 53(7):2598–2605, 2007.

[12] A. Juels and B. S. K. Jr. Pors: proofs of retrievability for large files. In *ACM Conference on Computer and Communications Security*, pages 584–597, 2007.

[13] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta. On availability of intermediate data in cloud computations. In *Proc. 12th Usenix Workshop on Hot Topics in Operating Systems (HotOS XII)*, 2009.

[14] S. Pallickara, J. Ekanayake, and G. Fox. Granules: A lightweight, streaming runtime for cloud computing with support, for map-reduce. In *CLUSTER*, pages 1–10, 2009.

[15] C.-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[16] H. Shacham and B. Waters. Compact proofs of retrievability. In *ASIACRYPT*, pages 90–107, 2008.

[17] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *ESORICS*, pages 355–370, 2009.

# A Proof of Knowledge Soundness

*Proof.* For some unavailable tags $\{\sigma^*\} \notin TagGen(sk, F)$, we assume that there exists an interactive machine $P^*$ can pass verification with noticeable probability, that is, there exists a polynomial $p(\cdot)$ and all sufficiently large $\kappa$'s,

$$\Pr[\langle P^*(F, \{\sigma^*\}), V\rangle(pk, \psi) = 1] \geq 1/p(\kappa). \tag{7}$$

Using $P^*$, we build a probabilistic algorithm $\mathcal{M}$ (called the knowledge Extractor) that breaks the Computational Diffie-Hellman $CDH$ problem in a cyclic group $\mathbb{G} \in \mathbb{S}$ of order $p$. That is, given $G, G_1, G_2 \in_R \mathbb{G}$, output $G^{ab} \in \mathbb{G}$, where $G_1 = G^a$, $G_2 = G^b$. The algorithm $\mathcal{M}$ is constructed by interacting with $P^*$ as follows:

**Setup:** $\mathcal{M}$ chooses a random $\gamma \in_R \mathbb{Z}_p$ and sets $g = G$, $h = G^\gamma$, $H_1 = G_1^\gamma$, $H_2 = G_2^\gamma$ as the public key $pk = (g, h, H_1, H_2)$, which is sent to $P^*$;

**Learning:** given a file $F = \{m_{i,j}\}_{j\in[1,s]}^{i\in[1,n]}$, $\mathcal{M}$ first chooses $s$ random $\tau_i \in_R \mathbb{Z}_p$ and $u_i = G_2^{\tau_i}$ for $i \in [1, s]$. Secondly, $\mathcal{M}$ assigns the indexes $1, \cdots, n$ into two sets $T = \{t_1, \cdots, t_{\frac{n}{2}}\}$ and $T' = \{t'_1, \cdots, t'_{\frac{n}{2}}\}$. Let $m_{t_i,j} \neq m_{t'_i,j}$ for all $i \in [1, n/2]$ and $j \in [1, s]$. Then, $\mathcal{M}$ builds an index table $\chi$ and $\xi^{(1)}$ in terms of the original scheme and generates the tag of each block, as follows:

- For each $t_i \in T$, $\mathcal{M}$ chooses $r_i \in_R \mathbb{Z}_p$ and sets $\xi_{t_i}^{(2)} = H_{\xi^{(1)}}(\chi_{t_i}) = G^{r_i}$ and $\sigma_{t_i} = G_1^{r_i} \cdot G_2^{\sum_{j=1}^s \tau_j \cdot m_{t_i,j}}$.
- For each $t'_i \in T'$, $\mathcal{M}$ uses $r_i$ and two random $r'_i, \zeta_i \in_R \mathbb{Z}_p$ to sets $\xi_{t'_i}^{(2)} = H_{\xi^{(1)}}(\chi_{t'_i}) = G^{r_i} \cdot G_2^{r'_i}$ and $\sigma_{t'_i} = G_1^{\zeta_i} \cdot G_2^{\sum_{j=1}^s \tau_j \cdot m_{t'_i,j}}$.

$\mathcal{M}$ checks whether $e(\sigma_{t'_i}, h) \stackrel{?}{=} e(\xi_{t'_i}^{(2)}, H_1) \cdot e(\prod_{j=1}^s u_j^{m_{t'_i,j}}, H_2)$ for all $t'_i \in T'$. If the result is true, then outputs $G^{ab} = G_2^a = (G^{\zeta_i} \cdot G_1^{r_i})^{(r'_i)^{-1}}$, otherwise $\mathcal{M}$ sends $(F, \sigma^* = \{\sigma_i\}_{i=1}^n)$ and $\psi = (u = \{u_i\}, \chi, \xi^{(1)})$ to $P^*$. At any time, $P^*$ can query the hash function $H_{\xi^{(1)}}(\chi_k)$, $\mathcal{M}$ responds with $\xi_{t_i}^{(2)}$ or $\xi_{t'_i}^{(2)}$ with consistency, where $k = t_i$ or $t'_i$.

**Output:** $\mathcal{M}$ chooses an index set $I \subset [1, \frac{n}{2}]$ and two subset $I_1$ and $I_2$, where $I = I_1 \bigcup I_2$, $|I_2| > 0$. $\mathcal{M}$ constructs the challenges $\{v_i\}_{i\in I}$ and all $v_i \neq 0$. Then $\mathcal{M}$ simulates $V$ to run an interactive $\langle P^*, \mathcal{M}\rangle$ as follows:

- Commitment. $\mathcal{M}$ receives $H'_1$ from $P^*$;
- Challenge. $\mathcal{M}$ sends the challenge $Q_1 = \{(t_i, v_i)\}_{i\in I}$ to $P^*$;
- Response. $\mathcal{M}$ receives $(\sigma', \{\mu'_j, \pi'_j\}_{j=1}^s)$ from $P^*$.

$\mathcal{M}$ checks whether or not these responses is an effective result by Equation 3. If it is true, then $\mathcal{M}$ completes a rewindable access to the prover $P^*$ as follows:

- Commitment. $\mathcal{M}$ receives $H''_1$ from $P^*$;
- Challenge. $\mathcal{M}$ sends the following challenge to $P^*$,

$$Q_2 = \{(t_i, v_i)\}_{i\in I_1} \bigcup \{(t'_i, v_i)\}_{i\in I_2};$$

- Response. $\mathcal{M}$ receives $(\sigma'', \{\mu''_j, \pi''_j\}_{j=1}^s)$ or a special halting-symbol from $P^*$.

If the response is not a halting-symbol, then $\mathcal{M}$ checks whether the response is effective by Equation 3, $H'_1 \stackrel{?}{=} H''_1$, for $\forall j \in [1, s]$, $\pi'_j \stackrel{?}{=} \pi''_j$. If they are true, then $\mathcal{M}$ outputs

$$G^{ab} = G_2^a = (\sigma''\sigma'^{-\psi} G_1^{r\cdot(\psi-1)\sum_{i\in I} r_i v_i})^{\frac{1}{r\cdot\sum_{i\in I_2} r'_i \cdot v_i}}. \tag{8}$$

where $r = (\mu''_j - \mu'_j) \cdot (\sum_{i\in I_2} v_i \cdot (m_{t'_i,j} - m_{t_i,j}))^{-1}$ and

$$\psi = \frac{\sum_{i\in I_1}\sum_{j=1}^s \tau_j m_{t_i,j} v_i + \sum_{i\in I_2}\sum_{j=1}^s \tau_j m_{t'_i,j} v_i}{\sum_{i\in I}\sum_{j=1}^s \tau_j m_{t_i,j} v_i}$$

for any $j \in [1, s]$, $H'_1 = H_1^r$ and $\psi \neq 1$.

It is obvious that we set $\alpha = a$ and $\beta = b$ in the above construction. Since the tags $\sigma_{t_i}$ are available in $\forall t_i \in T$, the responses in the first interactive satisfies the equation:

$$\begin{aligned} e(\sigma', h) &= e(\prod_{i\in I}(\xi_{t_i}^{(2)})^{v_i}, H_1') \cdot e(\prod_{j=1}^{s} u_j^{\mu_j'} \cdot \pi_j'^{-1}, H_2) \\ &= e(G^{\sum_{i\in I} r_i \cdot v_i}, H_1') \cdot e(\prod_{j=1}^{s} u_j^{\mu_j'} \cdot \pi_j'^{-1}, H_2). \end{aligned}$$

However, the $\sigma_{t_i'}$ are unavailable in $\forall t_i' \in T'$. In the second interaction, we require that $\mathcal{M}$ can rewind the prover $P^*$, i.e., the chosen parameters are the same in two protocol executions [7, 10]. In the above construction, this property ensures $H_1' = H_1'' = H_1^r$ and for all $j \in [1, s]$, $\pi_j' = \pi_j''$. So we have $\mu_j'' - \mu_j' = r \cdot \sum_{i\in I} v_i \cdot (m_{t_i',j} - m_{t_i,j}) = r \cdot \sum_{i\in I_2} v_i \cdot (m_{t_i',j} - m_{t_i,j})$, $\prod_{j=1}^{s} u_j^{\mu_j'} \pi_j'^{-1} = G_2^{\sum_{i\in I}\sum_{j=1}^{s} \tau_j m_{t_i,j} v_i}$, and $\prod_{j=1}^{s} u_j^{\mu_j''} \pi_j''^{-1} = G_2^{\sum_{i\in I_1}\sum_{j=1}^{s} \tau_j m_{t_i,j} v_i} \cdot G_2^{\sum_{i\in I_2}\sum_{j=1}^{s} \tau_j m_{t_i',j} v_i}$. In terms of the responses, we hold

$$\begin{aligned} & e(\prod_{i\in I_1}(\xi_{t_i}^{(2)})^{v_i} \cdot \prod_{i\in I_2}(\xi_{t_i'}^{(2)})^{v_i}, H_1'') \cdot e(\prod_{j=1}^{s} u_j^{\mu_j''} \cdot (\pi_j'')^{-1}, H_2) \\ = \; & e(\prod_{i\in I_1}(G^{r_i})^{v_i} \cdot \prod_{i\in I_2}(G^{r_i} \cdot G_2^{r_i'})^{v_i}, H_1'') \cdot e(\prod_{j=1}^{s} \frac{u_j^{\mu_j''}}{\pi_j''}, H_2) \\ = \; & e(\prod_{i\in I} G^{r_i \cdot v_i}, H_1') \cdot e(\prod_{i\in I_2} G_2^{r_i' \cdot v_i}, H_1') \cdot e(\prod_{j=1}^{s} \frac{u_j^{\mu_j'}}{\pi_j'}, H_2)^{\psi} \\ = \; & e(\sigma'^{\psi}, h) \cdot e(G_2^{\sum_{i\in I_2} r_i' v_i} \cdot G^{(1-\psi)\sum_{i\in I} r_i v_i}, H_1') = e(\sigma'', h) \end{aligned}$$

Thus, we have the equation $e(\sigma''/\sigma'^{\psi}, h) = e(G_2^{\sum_{i\in I_2} r_i' \cdot v_i} \cdot G^{(1-\psi)\sum_{i\in I} r_i v_i}, H_1')$ and $H_1' = h^{ar}$, the Equation 8 holds. Furthermore, we have

$$\begin{aligned} & \Pr[\mathcal{M}(CDH(G, G^a, G^b)) = G^{ab}] \\ \geq \; & \Pr[\langle P^*(F, \{\sigma^*\}), \mathcal{M}\rangle(pk, \psi) = 1] \geq 1/p(\kappa). \end{aligned}$$

It follows that $\mathcal{M}$ can solve the given $\epsilon$-CDH challenge with advantage at least $\epsilon$, as required. This completes the proof of Theorem. □

# B Proof of Zero-knowledge

*Proof.* In order to prove the theorem, we construct a protocol $\mathcal{S}'$ [3], which is the same as the original protocol $\mathcal{S}$, with one difference: we replace $\pi_i = u_i^{\lambda_i}$ with $\Lambda_i = e(u_i, H_2')^{\lambda_i}$, such that, $\Lambda_i = e(\pi_i, H_2)$, and the equation of verification is charged as

$$\begin{aligned} e(\sigma', h) &\stackrel{?}{=} e(\prod_{(i,v_i)\in Q} H_{\xi^{(1)}}(\chi_i)^{v_i}, H_1') \cdot \frac{e(\prod_{j=1}^{s} u_j^{\mu_j}, H_2)}{e(\prod_{j=1}^{s} \pi_j, H_2)} \\ &= e(\prod_{(i,v_i)\in Q} H_{\xi^{(1)}}(\chi_i)^{v_i}, H_1') \cdot \frac{e(\prod_{j=1}^{s} u_j^{\mu_j}, H_2)}{\prod_{j=1}^{s} \Lambda_j} \end{aligned}$$

For the protocol $\mathcal{S}'$, we construct a machine $S$, which is called a simulator for the interaction of $V$ with $P$. Given the public key $pk = (g, h, H_1, H_2)$, for a file $F$, a public verification information $\psi = (u_1, \cdots, u_s)$, and a index set $I$ ($t = |I|$), the simulator $S^F(pk, \psi)$ executes the following:

1. Chooses a random $\sigma' \in_R \mathbb{G}_1$ and computes $e(\sigma', h)$;
2. Chooses $t$ random coefficients $\{v_i\}_{i\in I} \in_R \mathbb{Z}_p^t$ and a random $\gamma \in_R \mathbb{Z}_p$ to compute $H_1' \leftarrow H_1^{\gamma}$ and

$$A_1 = e(\prod_{i\in I} H_{\xi^{(1)}}(\chi_i)^{v_i}, H_1');$$

[3] We do not adopt this scheme as a result of the high computational overhead of bilinear map.

3. Chooses $s$ random $\{\mu_i\} \in_R \mathbb{Z}_p^s$ to $A_2 = e(\prod_{j=1}^{s} u_j^{\mu_j}, H_2)$;

4. Chooses random $\{\pi_i\}_{i\in[1,s-1]} \in_R \mathbb{G}^{s-1}$ to calculate

$$\begin{cases} \Lambda_i & \leftarrow & e(\pi_i, H_2') & i \in [1, s-1] \\ \Lambda_s & \leftarrow & A_1 \cdot A_2 \cdot \left(e(\sigma', h) \cdot \prod_{i=1}^{s-1} \Lambda_i\right)^{-1} & \end{cases}.$$

Finally, the output of simulator $S$ is $S^F(pk, \psi) = (C, Q, \theta) = (\overline{H_1'}, \{(i, \overline{v_i})\}, (\overline{\Lambda}, \overline{\sigma'}, \overline{\mu}))$, which is an available verification for Equation (3). Let $View^F(\langle P(F, \sigma), V^* \rangle(pk, \psi)) = (H_1', \{(i, v_i)\}, (\Lambda, \sigma', \mu))$ denote the output of the interactive machine $V^*$ after interacting with the interactive machine $P$ on common input $(pk, \psi)$. In fact, every pair of variables is identically distributed in two ensembles, for example, $\overline{H_1'}, \{(i, \overline{v_i})\}$ and $H_1', \{(i, v_i)\}$ are identically distributed due to $\gamma, \{v_i\} \in_R \mathbb{Z}_p$, as well as $(\overline{\Lambda}, \overline{\sigma'}, \overline{\mu})$ and $(\Lambda, \sigma', \mu)$ is identically distributed due to $\overline{\sigma'} \in_R \mathbb{G}$, $\lambda_j \in_R \mathbb{Z}_p$ and $u_j \leftarrow \lambda_j + \sum_{i\in I} v_i \cdot m_{i,j}$ for $i \in [1, s]$, $\{\overline{\pi_i}\}_{i\in[1,s-1]} \in_R \mathbb{G}$, and the distribution of $\overline{\Lambda}_s$ is decided on the randomized assignment of the above variables. Hence, the ensembles $S^F(pk, \psi)$ and $View^F(\langle P(F, \sigma), V^* \rangle(pk, \psi))$ is computationally indistinguishable, thus for every probabilistic polynomial-time algorithm $D$, for every polynomial $p(\cdot)$, and for all sufficiently large $\kappa$, it holds that

$$\left| \begin{array}{c} \Pr[D\left(pk, \psi, S^F(pk, \psi)\right) = 1] - \\ \Pr[D\left(pk, \psi, View^F(P(F, \sigma), V^*)(pk, \psi)\right)] = 1 \end{array} \right| \leq 1/p(\kappa).$$

The fact that such simulators exist means that $V^*$ does not gain any knowledge from $P$ since the same output could be generated without any access to $P$. That is, the $\mathcal{S}'$ is a zero-knowledge protocol. Further, assume there exists no leakage of information by replace $\pi_i$ with $\Lambda_i$, such that $\mathcal{S}$ is a zero-knowledge protocol. □

# C Proof of Security against Forging Attack

*Proof.* Assume that an adversary $\mathcal{A}$ breaks the DPDP scheme $\mathcal{S}$ to forge an available tag, that is, the adversary $\mathcal{A}$ has advantage $\epsilon$ in forging the tags in $\mathcal{S}$ after $\mathcal{A}$ executes a series of dynamic data operations: given the valid $pk, \psi, \sigma$ for a file $F$ and a DPDP Oracle $S(\zeta)$ with the secret of tags $\zeta$,

$$\Pr\left[ \begin{array}{c} \mathcal{A}^{\mathcal{S}(\zeta)}(pk, \psi) = ((F, \sigma), (m^*, \sigma^*)) \\ \wedge \langle P(m^*, \sigma^*), V \rangle(pk, \psi)) = 1 \wedge m^* \notin F \end{array} \right] \geq \epsilon.$$

where $(F, \sigma)$ is the new file and tags after dynamic data operations and $(m^*, \sigma^*)$ is a forged data block and tag. Using $\mathcal{A}$, we build an algorithm $\mathcal{B}$ that breaks the Computational Diffie-Hellman (CDH) problem in a cyclic group $\mathbb{G} \in \mathbb{S}$ of order $p$. That is, given $G, G_1, G_2 \in_R \mathbb{G}$, output $G^{ab} \in \mathbb{G}$, where $G_1 = G^a$, $G_2 = G^b$. The algorithm $\mathcal{B}$ is constructed by interacting with $\mathcal{A}$ as follows:

**Setup:** $\mathcal{B}$ generates the public key $pk$ and $\psi$ as follows:

- chooses two random $\lambda, \gamma \in_R \mathbb{Z}_p$ and sets $g \leftarrow G_1$, $h \leftarrow G^{\lambda}$, $H_1 \leftarrow G_2^{\lambda}$, $H_2 \leftarrow G^{\gamma\lambda}$;
- chooses $s$ random $\tau_1, \cdots, \tau_s \in_R \mathbb{Z}_p$ and computes $u_i \leftarrow G_1^{\tau_i}$ for $i \in [1, s]$;
- builds an empty index set $L$ and an empty hash table $T$, in which each record $T_i$ involves a status $M_i$ (the initial value is $NULL$) and a random $X_i$ (the initial value is 0);
- provides $pk \leftarrow (g, h, H_1, H_2)$ and $\psi \leftarrow (u_1, \cdots, u_s)$ to $\mathcal{A}$.

**Learning:** $\mathcal{A}$ issues up to $t$ queries for dynamic data operations and $s$ hash queries, $\mathcal{B}$ responds to each query as follows:

- for an update query $(i, m_i)$: given any message $m_i = \{m_{i,j}\}_{j\in[1,s]}$, $\mathcal{B}$ searches the index table $\chi$ at the index $i$, if there exists the record $\chi_i$, then $\mathcal{B}$ sets the value of $V_i \leftarrow \max_{B_i=B_j}\{V_j\} + 1$ and chooses a random $R_i$ in $\chi_i$. $\mathcal{B}$ chooses a new random $r_i \in_R \mathbb{Z}_p$ and computes

$$H_{\xi^{(1)}}(\chi_i) \leftarrow G^{r_i}, \quad \sigma_i \leftarrow G_2^{r_i} \cdot G_1^{\sum_{j=1}^{s} \tau_j \cdot m_{i,j} \cdot \gamma}.$$

  Finally, $\mathcal{B}$ stores $(True, r_i)$ in $T_i$, sets $i \in L$, and sends $(m_i, \sigma_i)$ to $\mathcal{A}$;

- for a delete query $(i)$: given an index $i$, $\mathcal{B}$ sets $\chi_i =$ "$B_i||0||R_i$" chooses a new random $r_i \in_R \mathbb{Z}_p$ and computes $H_{\xi^{(1)}}(\chi_i) \leftarrow G^{r_i}$, $\sigma_i \leftarrow G_2^{r_i}$, delete this record and sends $\sigma_i$ to $\mathcal{A}$;
- for an insert query $(i, m_i)$: given any message $m_i = \{m_{i,j}\}_{j\in[1,s]}$, $\mathcal{B}$ searches the index table $\chi$ at the index $i$, if there exists the record $\chi_i$, then $\mathcal{B}$ moves the $No. \geq i$ records backward in order and sets the value of $B_i \leftarrow B_{i-1}$, $V_i \leftarrow \max_{B_i=B_j}\{V_j\} + 1$ and chooses a random $R_i$ in $\chi_i$. $\mathcal{B}$ chooses a new random $r_i \in_R \mathbb{Z}_p$ and computes

$$H_{\xi^{(1)}}(\chi_i) \leftarrow G^{r_i}, \quad \sigma_i \leftarrow G_2^{r_i} \cdot G_1^{\sum_{j=1}^{s} \tau_j \cdot m_{i,j} \cdot \gamma}.$$

  Finally, $\mathcal{B}$ stores $(True, r_i)$ in $T_i$, sets $i \in L$, and sends $(m_i, \sigma_i)$ to $\mathcal{A}$;
- for a hash query $(i)$: given an index $i$, if $i \in L$, then $\mathcal{B}$ sends $H_{\xi^{(1)}}(\chi_i) = G^{r_i}$ to $\mathcal{A}$ in terms of the stored $r_i$, else $\mathcal{B}$ chooses a new random $r_i \in_R \mathbb{Z}_p$, computes $H_{\xi^{(1)}}(\chi_i) = G_1^{r_i}$, and then increases a new record $T_i$ (sets $M_i = false$ and $X_i = r_i$) and sends $H_{\xi^{(1)}}(\chi_i) = G_1^{r_i}$ to $\mathcal{A}$.

**Output:** after $\mathcal{A}$ outputs a forgery $(m_i^*, \sigma_i^*)$, $\mathcal{B}$ checks whether the index $i$ belongs to $L$ and the following equation holds

$$e(\sigma_i^*, h) \stackrel{?}{=} e(G_1^{r_i}, H_1) \cdot e(\prod_{j=1}^{s} u_j^{m_{i,j}^*}, H_2), \tag{9}$$

which can replace the verification of $\langle P, V\rangle$. If $(m_i^*, \sigma_i^*)$ passes the above check, $\mathcal{B}$ outputs

$$G^{ab} \leftarrow \left(\frac{\sigma_i^*}{G_1^{\sum_{j=1}^{s} \tau_j \cdot m_{i,j}^* \cdot \gamma}}\right)^{r_i^{-1}}. \tag{10}$$

It is easy to discover that $\mathcal{B}$ defines $\alpha = b$ (unknown) and $\beta = \gamma$. As the hash function is a random Oracle manipulated by $\mathcal{B}$, $\mathcal{B}$ provides the different values in terms of the choices of $\mathcal{A}$, that is,

$$H_{\xi^{(1)}}(\chi_i) = \begin{cases} G^{r_i} & M_i = True \\ G_1^{r_i} & M_i = False \end{cases} \tag{11}$$

where $L$ denotes the set of all distributed tags. When $i \in L$, it is easy to prove the availability of all tags $\{\sigma_i\}_{i\in L}$ by using the following equation

$$e(\sigma_i, h) = e(G_2^{r_i}, G^{\lambda}) \cdot e(G_1^{\sum_{j=1}^{s} \tau_j \cdot m_{i,j} \cdot \gamma}, G^{\lambda}) = e(G^{r_i}, H_1) \cdot e(\prod_{j=1}^{s} u_j^{m_{i,j}}, H_2).$$

But for all $i \notin L$, $\mathcal{B}$ uses the hash value $G_1^{r_i}$ to solve the CDH problem:

$$\begin{aligned} e(\sigma_i^*, h) &= e(G_1^{r_i}, H_1) \cdot e(\prod_{j=1}^{s} u_i^{m_{i,j}^*}, H_2) \\ &= e(G_1^{r_i}, G_2^{\lambda}) \cdot e(G_1^{\sum_{j=1}^{s} \tau_j \cdot m_{i,j}^* \cdot \gamma}, G^{\gamma\cdot\lambda}) \\ &= e(G^{a\cdot b\cdot r_i}, G^{\lambda}) \cdot e(G_1^{\gamma \cdot \sum_{j=1}^{s} \tau_j \cdot m_{i,j}^* \cdot \gamma}, G^{\lambda}) = e(\sigma_i^*, G^{\lambda}) \end{aligned}$$

The probability that a given tag query causes $\mathcal{B}$ to abort is at most $q_h/p$ and therefore the probability that $\mathcal{B}$ aborts as a result of $\mathcal{A}$'s tag is at most $q_h q_t/p$. Thus, $\mathcal{B}$ can solve the given CDH challenge with advantage at least $\epsilon(1 - \frac{q_h q_t}{p})$, as required. This completes the proof of Theorem. □