International Journal of Cooperative Information Systems
Vol. 21, No. 3 (2012) 165–197

DOI: 10.1142/S0218843012410018

World Scientific
www.worldscientific.com

SECURE COLLABORATIVE INTEGRITY VERIFICATION FOR HYBRID CLOUD ENVIRONMENTS*

YAN ZHU† and SHANBIAO WANG‡

Beijing Key Laboratory of Internet Security Technology, Peking University
Institute of Computer Science and Technology, Peking University
Beijing 100871, China
†*yan.zhu@pku.edu.cn*
‡*wangsb@pku.edu.cn*

HONGXIN HU

Department of Computer and Information Sciences
Delaware State University, Dover, DE 19901, USA
hhu@desu.edu

GAIL-JOON AHN

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, AZ 85281, USA
gahn@asu.edu

DI MA

Department of Computer and Information Science
University of Michigan-Dearborn, Dearborn, MI 48168, USA
dmadma@umd.umich.edu

Received 22 February 2012
Accepted 4 September 2012
Published 29 November 2012

A hybrid cloud is a cloud computing environment in which an organization provides and manages some internal resources and has others provided externally. However, this new environment could bring irretrievable losses to the clients due to a lack of integrity verification mechanism for distributed data outsourcing. To support scalable service and data migration, in this paper we address the construction of a collaborative integrity verification mechanism in hybrid clouds where we consider the existence of multiple cloud service providers to collaboratively store and maintain the clients' data. We propose a collaborative provable data possession scheme adopting the techniques of homomorphic verifiable responses and hash index hierarchy. In addition, we articulate the performance optimization mechanisms for our scheme and prove the security of our scheme based on multi-prover zero-knowledge proof system, which can satisfy the properties of completeness, knowledge soundness, and zero-knowledge. Our experiments also show that our proposed solution only incurs a small constant amount of communications overhead.

Keywords: Integrity verification; multi-prover; collaborative; hybrid clouds.

*This article is an extended version of the article originally published in CollaborateCom2011.[1]

1. Introduction

Cloud computing has become a faster profit growth point in recent years by providing a comparably low-cost, scalable, location-independent platform for clients' data. Although commercial cloud services have revolved around public clouds, the growing interest of building private cloud on open-source cloud computing tools allows local users to have a flexible and agile private infrastructure to run service workloads within their administrative domains. Private clouds are not exclusive for being public clouds, and they can also support a *hybrid* cloud model by supplementing a local infrastructure with computing capacity from an external public cloud. By using virtual infrastructure management (VIM),[2] a hybrid cloud can allow remote access to its resources over the Internet via remote interfaces, such as the Web services interfaces that Amazon EC2 uses.

Usually, a hybrid cloud is a cloud computing environment in which an organization provides and manages some internal resources as well as external resources. From the viewpoint of internal resource management, a hybrid cloud is different from a *multiple* cloud environment, which usually refers to an open cloud architecture consisting of multiple public clouds. Therefore, a hybrid cloud puts more emphasis on a cloud aggregation platform including private clouds and public clouds, combining the features of availability, scalability, and low cost from public clouds, and security from private clouds. For example, as shown in Fig. 1, an organization, considered as a hybrid cloud, uses public cloud services, such as Zoho and Amazon's EC2, for general computing purposes while storing customers' data within its own data centers, such as university health science center (Private Cloud I) and medicinal research institute (Private Cloud II).[3] Obviously, it is more secure to store customers' data in private clouds.

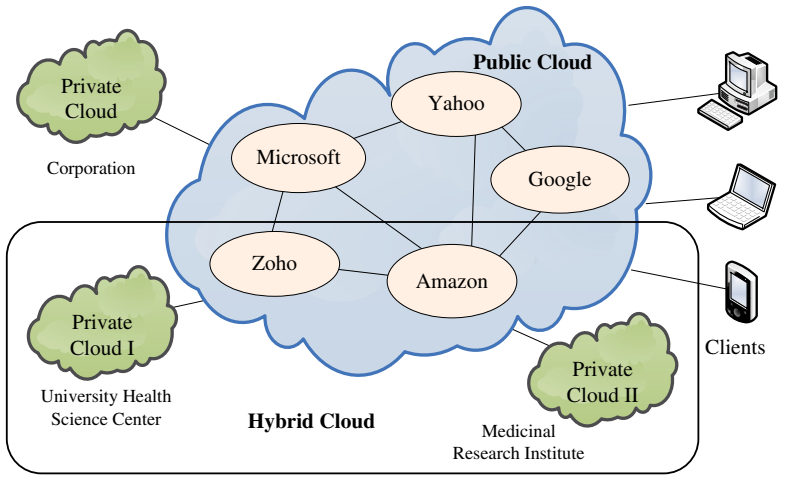


Fig. 1. Types of cloud computing: private cloud, public cloud and hybrid cloud.

As cloud computing has been rapidly adopted, the hybrid model will be more prevalent for a number of reasons[2]: to provide clients with the same features found in commercial public clouds; to provide a uniform and homogeneous view of virtualized resources; to support configurable resource allocation policies to meet an organization's specific goals (high availability, server consolidation to minimize power usage, and so on); and to meet an organization's changing resource needs. In particular, hybrid cloud also fulfills an organization's security requirements, that is, sensitive data is entirely controlled and retained by the enterprise.

With the growing popularity of clouds, the tools and technologies for hybrid clouds are emerging recently, such as Platform VM Orchestrator,[a] VMware vSphere,[b] and Ovirt.[c] They help users construct a comparably low-cost, scalable, location-independent platform for managing clients' data. However, if such an important platform is vulnerable to security attacks, it would bring irretrievable losses to the clients. For example, confidential data in an enterprise's private cloud may be illegally accessed by using remote interfaces in hybrid cloud, or relevant data and archives are lost or tampered with when they are transferred into an uncertain storage pool outside the enterprise by an application in a hybrid cloud. Therefore, it is indispensable for cloud service providers (CSPs) to provide secure management techniques to ensure their storage services.

Provable data possession (PDP)[4] is a probabilistic proof technique for a storage provider to prove that clients' data remains intact. In other words, clients can fully recover their data and have confidence to use the recovered data. This creates strong demand for seeking an effective solution to check if their data has been tampered with or deleted without downloading the latest version of data. Various PDP schemes have been recently proposed, such as Scalable PDP[5] and Dynamic PDP,[6] to work in a publicly verifiable way so that users can employ their verification protocol to prove the availability of the stored data. However, these schemes focus on the PDP issues at untrusted servers (public clouds), and are not applicable for a hybrid cloud environment due to the lack of support for heterogeneous multi-cloud storages, as well as a privacy protection mechanism (see Sec. 2.3 for details).

In this paper, we address the problem of provable data possession in hybrid clouds. By discussing the characteristics of hybrid clouds and analyzing security drawbacks of some existing schemes, we indicate our main research objectives in three aspects: high security, verification transparency and high performance. On this basis, we first propose a verification framework for hybrid clouds along with the description of three techniques adopted in our approach: (i) fragment structure, (ii) hash index hierarchy (HIH), and (iii) homomorphic verifiable response (HVR). We argue that it is possible to construct a collaborative PDP (CPDP) scheme

[a] www.platform.com/Products/platform-vm-orchestrator
[b] www.vmware.com/products/vsphere
[c] http://ovirt.org

without compromising data privacy based on modern cryptographic techniques, such as multi-prover zero-knowledge proof system (MP-ZKP).[7]

We then provide an effective construction of CPDP using homomorphic verifiable responses and hash index hierarchy. This construction realizes security against data leakage attacks and tag forging attacks, considering transparent property for clients to store and manage resources in hybrid clouds. This construction uses homomorphic property, on which the responses of clients' challenges computed from multiple CSPs can be combined into a single response as the final result of hybrid clouds. By using such a mechanism, clients can be convinced of data possession without knowing geographical locations where their files reside. In addition, a new hash index hierarchy is proposed to realize the client-oriented transparency measures to store and manage clients' resources in hybrid clouds.

We also evaluate the performance of our CPDP scheme. First, we provide a brief security analysis of our scheme. We also analyze the performance of probabilistic queries for detecting abnormal situations in a timely manner. This probabilistic method also has the inherent benefit in reducing the computation and communication overheads. In addition, we prove the security of our scheme based on multi-prover zero-knowledge proof system, which can satisfy the properties of completeness, knowledge soundness and zero-knowledge. In practical applications, our optimization algorithm also provides an adaptive parameter selection for different sizes of files (or clusters), which could ensure that the extra storage is optimal for the verification process.

The rest of the paper is organized as follows. In Sec. 2, we address our research motivation and research objectives. Section 3 discusses our framework and model of integrity verification in hybrid clouds. We describe background techniques adopted in our practical construction in Sec. 4. Section 5 describes the security and performance analysis of our solution. We address the related work in Secs. 6 and 7 concludes this paper and discusses the future work.

2. Motivation and Objectives

In this section, we give an overview of our motivation and research objectives in constructing collaborative PDP. Our motivation is based on following challenging questions that need to be addressed in secure cloud storage, which also help us define our objectives in this paper.

2.1. *Why need integrity checking of data in clouds?*

Cloud storage service has become a new profit growth point by providing a comparably low-cost, scalable, location-independent platform for managing clients' data. However, data security is critical to such convenient storage services due to the following reasons: cloud infrastructures are much more powerful and reliable than personal computing devices but they are still facing all kinds of internal and external threats; for the benefits of their own business, there exist various motivations

for cloud service providers to behave unfaithfully toward cloud users; furthermore, the behaviors of CSPs may not be known by cloud users, thus a dispute may be raised even if this dispute may result from users' own improper operations. Therefore, it is necessary to explore various mechanisms for integrity, availability, and confidentiality of data stored in clouds.

Data integrity verification is one of the most basic and critical techniques in outsourced cloud storage. In cryptography, message authentication code (MAC) and digital signature can be applied to verify data integrity and ownership. For example, to sign data D, a data owner first produces a fix length string $H = Hash(D)$ through a hash function, and then signs it to compute a data tag $S = Sign_{sk}(H)$ by his/her private key. To verify, both the data and the tag (D, S) will be sent to the verifier, and then the verifier checks that $Verify_{pk}(D, S) = True/False$ by using a verification algorithm, where (pk, sk) is a key pair of the data owner. But this process is not suitable for outsourced cloud storage due to the following reasons:

- The verification process requires to send back original data to the verifier. It is only fit for short data, but it is obviously not possible for large amounts of data due to higher transmission overheads; and
- The verification process cannot achieve integrity verification for a part of data, but it is not fit for distributed cloud storage in which outsourced data may be distributed to different physical storage devices or CSPs.[8]

Therefore, it is crucial for CSPs to offer an efficient verification mechanism for solving the above-mentioned problems. This will bring the following advantages: verification without download and verification for partial data.

2.2. *Why need a new mechanism for ensuring data security in hybrid clouds?*

As mentioned previously, a hybrid cloud is an environment consisting of multiple public clouds and private clouds. A hybrid cloud typically needs at least one private cloud. The reason for the existence of private cloud is that it offers the greatest level of security and control for enterprises' sensitive data, hence security risks are less as compared to those stored in public clouds. Ideally, a hybrid approach allows a business to take advantage of the scalability and cost-effectiveness of a public cloud environment, without exposing mission-critical applications and data to third-party vulnerabilities.

In hybrid clouds, one of the core design principles is dynamic scalability, which guarantees cloud storage services to handle growing amounts of application data in a flexible manner. By employing the virtualization technique, such as VIM, hybrid clouds can effectively provide dynamic scalability of service and data migration. This kind of scalability is also particularly important for large enterprises taking into account their existing substantial investments of infrastructure. Furthermore, many organizations would prefer to keep sensitive data under their own control,

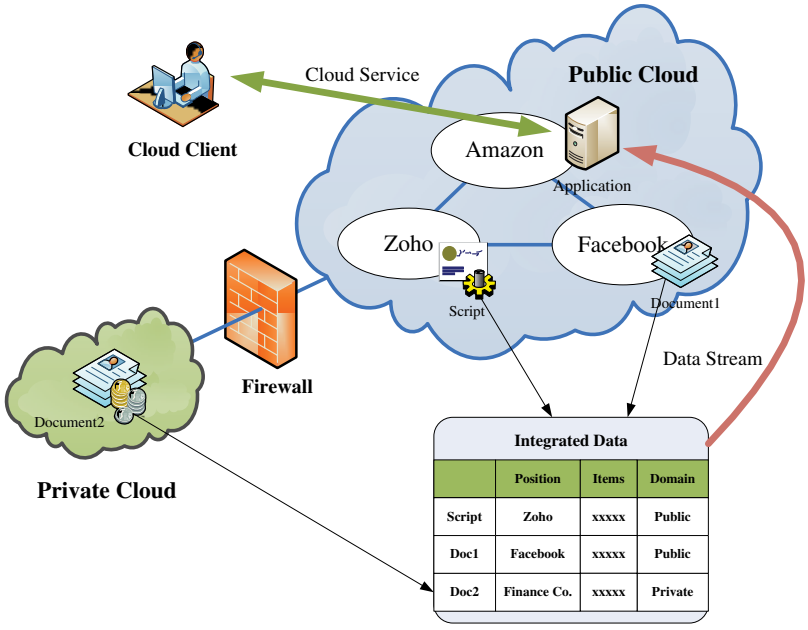


Fig. 2. The cloud application for integrated data in hybrid clouds.

ensuring security requirements. For example, a client might integrate data and scripts from multiple private or public providers into a large-size archive or a complete application setting; or a cloud service might capture data from other services in private clouds, but application scripts, intermediate data and results are executed and stored in public clouds.[9,10]

We show such an example of collaborative computing in Fig. 2. Assume that a cloud service, like a Web-based application on service-oriented architecture (SOA), is constructed on Amazon's EC2. The running of this service depends on the scripts from Zoho, the documents from Facebook and Finance Co. In this example, all of these integrated data, like documents and scripts, is called an *application setting*. From a practical standpoint, this kind of application setting can be either a large file, a database, or a set of files in an application system including softwares, Web pages, snapshots, and so on. Obviously, this collaborative paradigm is more vulnerable to various security attacks. For instance, an attacker can modify application softwares or scripts, tamper with application data, or load a trojan into a snapshot of virtual machine (VM) to compromise cloud applications. In order to mitigate these security risks, the safest way is to verify the data integrity of application setting before the application service is executed or the outsourced data is downloaded. In addition, an easily overlooked problem is to avoid disclosure of sensitive data in the verification process itself when this process is considered as public Daemons (like httpd, inetd, and fmd).

2.3. *Are existing PDP schemes efficient for hybrid clouds?*

As mentioned above, traditional cryptographic technologies based on Hash functions and signature schemes[11,12] cannot support the verification of data integrity and availability of outsourced data without a local copy of data. It is evidently impractical to download the whole data to verify data validation due to the expensiveness of communication, especially, for large-size files. Recently, several PDP schemes are proposed to address this issue. Essentially, the PDP is an interactive proof system between a CSP and a client because the client makes a false/true decision for data possession without downloading data.

Existing PDP schemes mainly focus on integrity verification issues at untrusted stores in single clouds, but they are not suitable for a hybrid cloud environment since they were originally constructed based on a two-party interactive proof system (TP-IPS). In cryptography, there exist some quite subtle differences between TP-IPS and multi-party interactive proof system (MP-IPS). These differences restrict the application of existing PDP scheme in hybrid clouds. For example, existing PDP schemes usually employ three-move interactive mode: *Commitment*, *Challenge*, and *Response*. In the first step, the CSPs need to select some random variables and sent their commitments into the client. However, such random selections of variables are independent, thus we cannot simply combine them into PDP scheme for a hybrid cloud. Although it is an alternative way for multiple CSPs to generate random variables by using Multi-Party Random Coin Tosses and Byzantine Agreement, this requests a complex protocol to ensure the randomness and consistency of results (among multiple parties). Likewise, some similar problems must be solved in other steps.

On the other hand. If we do not consider the relationship among data stored in deferent CSPs, existing PDP schemes can be used in a trivial way to realize data integrity verification in a hybrid cloud. That is, a verifier (a client) must invoke them repeatedly to check the integrity of data stored in each single cloud. This means that the verifier must know the exact position of each data block in hybrid clouds. Moreover, this process will consume higher communication bandwidth and computation overheads at both the verifier side and the cloud sides.

In response to actual characteristics of storage services in a hybrid cloud, the concerns to improve the existing PDP schemes are mainly from three aspects:

- How to design a more efficient PDP model for hybrid clouds to reduce storage and network overheads;
- How to deal with heterogeneous storage in a hybrid cloud and enhance the transparency of verification activities for the verifier; and
- How to provide an efficient sampling policy to help provide a more cost-effective verification service.

Solving these problems will help improve the quality of PDP services, which can not only timely detect abnormality, but also take up less resources, or rationally

allocate resources. Hence, a new PDP scheme is desirable to accommodate these application requirements from hybrid clouds.

2.4. *Are existing PDP schemes secure enough for hybrid cloud environments?*

In hybrid clouds, a collaborative work model provides some mutual channels among individual clouds. This kind of channels will no doubt increase the possibility of malicious attacks. In particular, they also provide a possibility of unauthorized access to data in a private cloud. This should be a deadly threat for hybrid cloud environments because most of existing PDP schemes ignore the leakage problem of verified data via the interactive process of the verification protocol in a PDP scheme. Thus, when a public verification service does not have a strong security mechanism to data protection, a malicious attacker could easily exploit such a service to obtain private data. Such an attack is extremely dangerous to the confidential data of an enterprise.

Even though existing PDP schemes have addressed various aspects such as public verifiability,[4] dynamics,[6] scalability,[5] and privacy preservation,[13] we still need a careful consideration to the following attacks.

- *Data leakage attack*: Through the interfaces of public clouds, various applications in hybrid clouds are allowed to access data in private clouds, so a PDP service (considered as a *Daemon*) undoubtedly provides a covert channel to access the secret data in private clouds. Therefore, if a PDP scheme cannot resist against the data leakage attacks, an adversary can easily obtain the entire data through an interactive proof process. For instance, Attacks 1 and 3 described in Appendixes A and B demonstrate that a verifier can get the stored data after running or wire-tapping sufficient verification communications. It is obvious that such attacks could significantly impact the privacy of outsourced data in clouds.
- *Tag forgery attack*: In hybrid clouds, an untrusted CSP has more opportunities to induce a forgery attack, in which the CSP can cheat a verifier by generating a valid tag for the tampered data. For example, Attacks 2 and 4 given in Appendixes A and B show that a successful forgery attack can occur only if one of the following cases is happened:
  (a) Clients modify data blocks in a file;
  (b) Clients insert and delete blocks repeatedly in a file;
  (c) Clients reuse the same file name to store multiple different files.

2.5. *Our objectives*

To summarize the above discussions, we believe that it is very essential to develop an efficient collaborative method for data integrity verification in hybrid cloud environments. Furthermore, from the above-mentioned challenges, our objectives for

checking integrity of outsourced data in hybrid clouds are as follows:

- *Usability aspect*: In the way of collaboration, a client should utilize the integrity check in a distributed cloud storage system. Our scheme should conceal the details of the storage to reduce the burden on clients;
- *Security aspect*: Our scheme should provide adequate security features to resist some existing attacks, especially data leakage attack and tag forgery attack; and
- *Performance aspect*: Our scheme should have the lower communication and computation overheads than non-cooperative solutions.

3. Framework and Model

In this section, we present our verification framework for hybrid clouds and a formal definition of collaborative PDP. In order to guarantee security, we also define a security model based on zero-knowledge interactive proof system.

3.1. *Verification framework for hybrid clouds*

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 the aspects of security, bandwidth and usability. To solve this problem, we consider a hybrid cloud storage service as illustrated in Fig. 3.

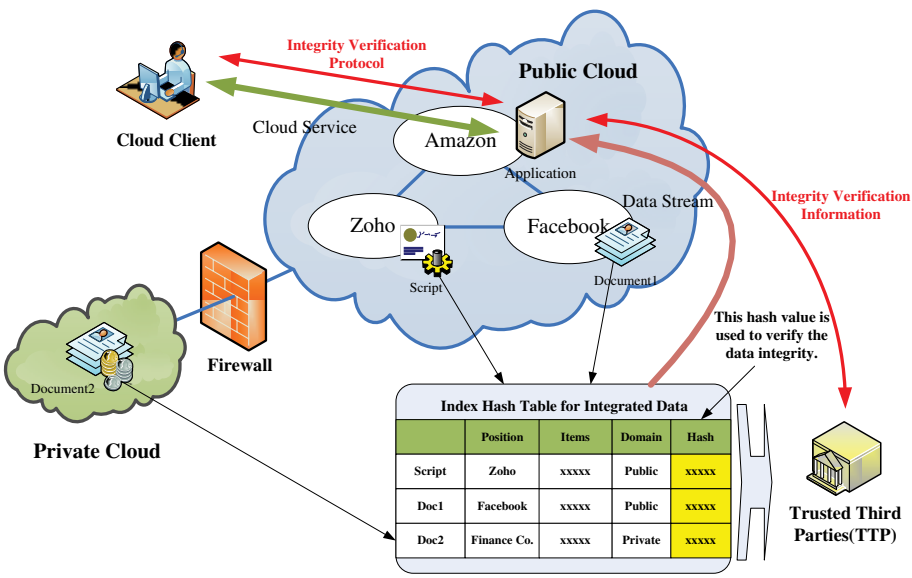


Fig. 3. Verification architecture for data integrity in hybrid clouds.

In this architecture, we consider a data storage service in a hybrid cloud involving three different entities:

- Granted clients, who have large amount of data stored in a hybrid cloud and have the right to access and manipulate these stored data;
- Cloud service providers (CSPs), who work together to provide data storage service and have enough storage spaces and computation resources;
- Trusted third party (TTP), who is trusted to store verification parameters, including index-hash table for integrated data (see Fig. 3), and offers the query services for these parameters.

In Fig. 3, we present an index hash table to manage the application data aggregated from multiple CSPs in a hybrid cloud. Moreover, some basic data items, such as data block position, access domain, and hash value, should be added to this table. In our scheme, one of the most important items is a cryptographic hash value, which is used to compress the record itself and supports the data integrity verification in collaborative PDP services. More importantly, this table is also used to solve the heterogeneous storage problem (see Sec. 4.1).

To support this architecture, a cloud storage provider also needs to add corresponding modules to implement collaborative PDP services. For example, OpenNebula is an open source, virtual infrastructure manager that is integrated with multiple virtual machine managers, transfer managers, and external cloud providers. In Fig. 4, we describe such a cloud computing platform based on OpenNebula architecture,[2] in which a service module of collaborative PDP is added into cloud computing management platform (CCMP). This module is able to respond

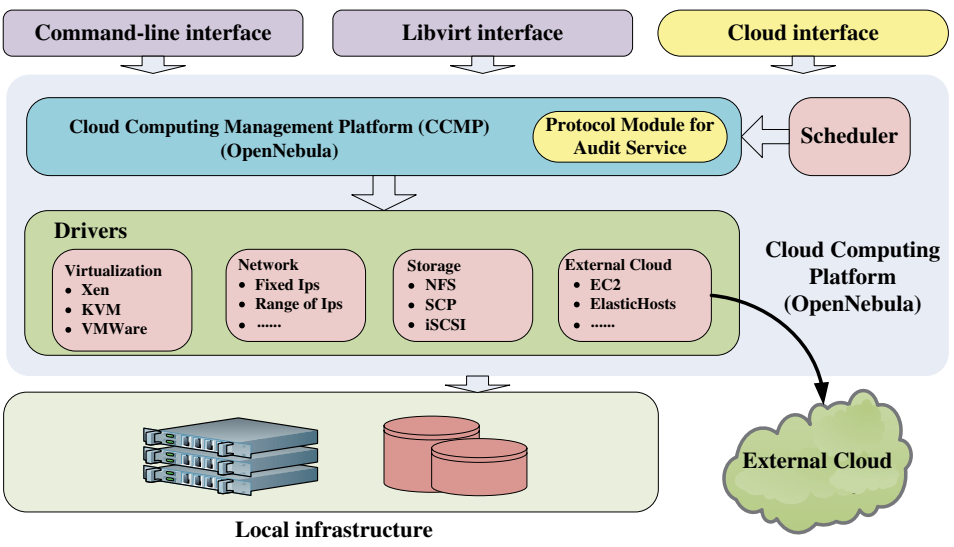


Fig. 4. Cloud computing platform for CPDP service based on OpenNebula architecture.

to the PDP requests of TTP through cloud interfaces. In addition, a hash index hierarchy (HIH), which is described in detail in Sec. 4.1, is used to provide a uniform and homogeneous view of virtualized resources in virtualization components. For the sake of clarity, we use yellow color to indicate the changes from the original OpenNebula architecture.

In this architecture, we consider the existence of multiple CSPs to collaboratively store and maintain clients' data. Moreover, a collaborative PDP is used to verify the integrity and availability of their stored data in CSPs. The verification flowchart is described as follows: First, the client (data owner) uses the secret key to pre-process a file, which consists of a collection of n blocks, generates a set of public verification information that is stored in TTP, transmits the file and some verification tags to CSPs, and may delete its local copy; then, by using a verification protocol for collaborative PDP, the clients can issue a challenge for one CSP to check the integrity and availability of outsourced data in terms of public verification information stored in TTP.

3.2. *Definition of collaborative PDP*

In order to prove the integrity of data stored in hybrid clouds, we define a framework for collaborative provable data possession (CPDP) based on multi-prover interactive proof system (MP-IPS)[14,15]:

Definition 3.1 (Collaborative-PDP). A collaborative provable data possession scheme $\mathcal{S}$ is a collection of two algorithms and a multi-prover interactive proof system, $\mathcal{S} = (\mathcal{K}, \mathcal{T}, \mathcal{P})$:

(i) $\mathcal{K}eyGen(1^{\kappa})$: takes a security parameter κ as input, and returns a secret key sk or a public-secret keypair (pk, sk);
(ii) $\mathcal{T}agGen(sk, F, \mathcal{P})$: 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 (ζ, ψ, σ), where ζ 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;
(iii) $\mathcal{P}roof(\mathcal{P}, V)$: is a protocol of proof of data possession between the CSPs ($\mathcal{P} = \{P_k\}$) and a verifier (V), that is, $\langle \sum_{P_k \in \mathcal{P}} P_k(F^{(k)}, \sigma^{(k)}), V \rangle (pk, \psi)$, 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 ψ is the common input between P and V. At the end of the protocol running, V returns a bit $\{0 \mid 1\}$ denoting false and true.

where, $\sum_{P_k \in \mathcal{P}}$ denotes the collaborative 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. However, it would cause significant cost growth in terms of communication and computation overheads. It is obviously unreasonable to adopt such a primitive

Table 1. 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. $F = \{m_{i,j}\}_{j\in[1,s]}^{i\in[1,n]}$;
σ	the set of tags, i.e. $\sigma = \{\sigma_i\}_{i\in[1,n]}$;
Q	the set of index-coefficient pairs, i.e. $Q = \{(i, v_i)\}$;
θ	the response for the challenge Q.

approach that diminishes the advantages of cloud storage: scaling arbitrarily up and down on-demand.[16] For the sake of clarity, we list some used signals in Table 1.

3.3. *Security model for collaborative PDP*

In cryptography, the CPDP scheme is a multi-prover interactive proof system (MP-IPS) in nature. According to the security definition of MP-IPS, we require that the CPDP scheme satisfies the following security requirements:

Definition 3.2. A pair of interactive machines $(\sum_{P_k\in\mathcal{P}} P_k, V)$ is called an available provable data possession for a file F if $\mathcal{P} = \{P_k\}$ is a collection of (unbounded) probabilistic algorithms, V is a deterministic polynomial-time algorithm, and the following conditions hold for some polynomial $p_1(\cdot), p_2(\cdot)$, and all $s \in \mathbb{N}$:

(i) Completeness: For every $\sigma \in \mathit{TagGen}(sk, F)$,

$$\Pr\left[\left\langle \sum_{P_k\in\mathcal{P}} P_k(F^{(k)}, \sigma^{(k)}), V \right\rangle (pk, \psi) = 1\right] \geq 1 - 1/p_1(\kappa); \tag{1}$$

(ii) Soundness: For every $\sigma^* \notin \mathit{TagGen}(sk, F)$, every interactive machine $P_k^* \in \mathcal{P}$,

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

where, $p_1(\cdot)$ and $p_2(\cdot)$ denote two polynomials,[d] and κ is a security parameter used in $\mathit{KeyGen}(1^\kappa)$.

The standard definition of proofs of knowledge was proposed by Bellare and Goldreich.[14] Here, the knowledge soundness could be regarded as the stricter definition of security of tag information, e.g. the forging tag attack. This means that the prover can forge file tags by means of a knowledge extractor $\mathcal{M}$ if soundness property does not hold.

[d]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 $1/p_1(\kappa) + 1/p_2(\kappa) \leq 1 - 1/\mathit{poly}(\kappa)$.

For a private cloud, we concerned more 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. In order to solve this problem, we introduce Zero-Knowledge notion into the CPDP scheme, as follows:

Definition 3.3 (Zero-knowledge). An interactive 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 polynomial-time algorithm D, for every polynomial $p(\cdot)$, and for all sufficiently large s, it holds that

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

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 $\langle\sum_{P_k\in\mathcal{P}}P_k(F^{(k)},\sigma^{(k)}),V^*\rangle(pk,\psi)$[e] are computationally indistinguishable.

Actually, zero-knowledge is a property that captures $\mathcal{P}$'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.

4. Main Techniques

In this section, we propose a new cooperative provable data possession scheme for hybrid clouds. Concretely speaking, this scheme can meet the following requirements: conceal the details of the storage, which is not necessary to be known by users in the verification process; and ensure that the verification does not reveal any information, which is of particular importance for sensitive data. We set up our system using bilinear pairings proposed by Boneh and Franklin.[17] We define this kind of bilinear pairings in the following bilinear map group system:

Definition 4.1 (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 following objects: $\mathbb{G}$ and $\mathbb{G}_T$ are two multiplicative groups using elliptic curve conventions with a large prime order p. The function e is 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

- Bilinearity: $e([a]G,[b]H)=e(G,H)^{ab}$;
- Non-degeneracy: $e(G,H)\neq 1$ unless G or $H=1$; and
- Computability: $e(G,H)$ is efficiently computable.

[e]The output of the interactive machine V^* after interacting with $\sum_{P_k\in\mathcal{P}}P_k(F^{(k)},\sigma^{(k)})$, on common input (pk,ψ).

In the remainder of this section, we first introduce the execution environment of our scheme, including hash index hierarchy and outsourced data storage structure. Then, we give our CPDP scheme in details, and discuss the validity of the scheme.

4.1. *Hash index hierarchy for collaborative PDP*

As a virtualization approach, we introduce a simple index-hash table to record the status and change of file blocks, as well as generate the Hash value of block in the verification process (see an example in Fig. 3). The structure of our index-hash table is similar to that of file block allocation table in file systems. The index-hash table consists of serial number, block number, version number, random integer, and so on. Different from the common index table, we must assure that all records in this kind of table differ from one another to prevent the forgery of data blocks and tags. In practical applications, it should been constructed into the virtualization infrastructure of cloud-based storage service.[2] In addition, the index-hash table can be used to solve heterogeneous storage problem because it is irrelevant to the type of storage systems in a hybrid cloud.

As an extension of name space in data clustering,[18] a representative architecture for data storage in hybrid clouds is illustrated as follows: 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; and
- **Third-Layer (*Storage Layer*):** practicably realize data storage on many physical devices.

This kind of architecture is a nature representation of file storage. We utilize this simple hierarchy to organize multiple CSP services, including private clouds or public clouds. In practical applications, this architecture can be constructed into a virtualization infrastructure of cloud-based storage service. In Fig. 5, we show an example of mapping three-layer structure into Hadoop distributed file system (HDFS),[19] which is a distributed, scalable, and portable file system. HDFS' architecture is composed of NameNode and DataNode, where NameNode maps a file name to a set of indexes of blocks and DataNode indeed stores data blocks. Obviously, the Express layer corresponds to the NameNode and the Service layer corresponds to secondary NameNode in each CSP. Finally, the Storage layer can be used to represent the structure of DataNode. Hence, based on this architecture, it becomes possible for the CPDP scheme to support actual storage systems. Also, it lays a foundation for replacing the common checksum algorithm with the CPDP scheme to implement the data integrity verification. It is worth pointing out that

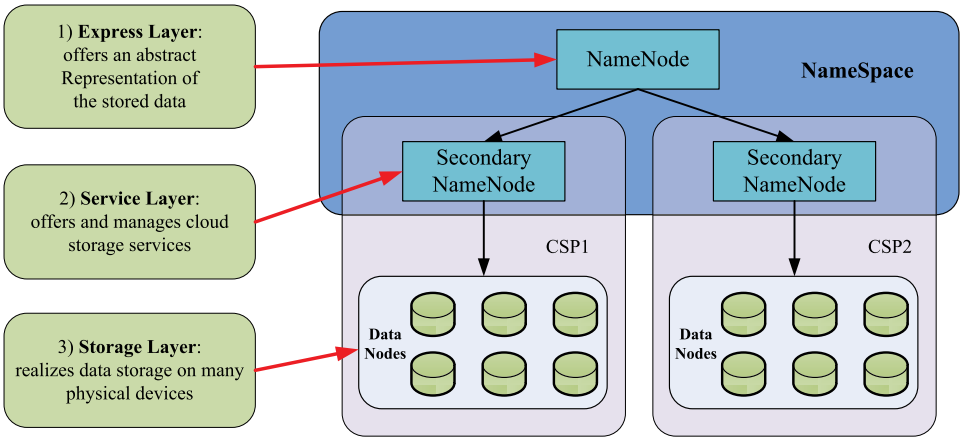


Fig. 5. The three-layer hierarchical structure over name space and data nodes.

our architecture is adaptable for the other cloud storage systems, such as Google File System (GFS),[20] Amazon S3 File System, and CloudStore.

Based on this hierarchical structure, we define a new index-hash table which is a data structure that uses a hash function to map data identifying values, known as indexes (e.g. file name, CSP server name, block number, and version number), to some fix-size (unique) feature values. These feature values, used in our CPDP construction, are generated by a collision-resistant hash function (also fixed-length compression function). Let $\mathbb{H} = \{H_k\}$ be a family of hash functions $H_k : \{0,1\}^n \rightarrow \{0,1\}^*$ indexed by $k \in \mathcal{K}$, where $\mathcal{K}$ is the key space. We have the following definition:

Definition 4.2 (Collision-Resistant Hash). A hash family $\mathbb{H}$ is (t, ϵ)-collision-resistant if no t-time adversary has advantage at least ϵ in breaking collision-resistance of $\mathbb{H}$, that is, for every probabilistic polynomial-time algorithm $\mathcal{A}$,

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

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

Given a hash function $H_k(\cdot)$, we make use of this architecture to construct a Hash Index Hierarchy $\mathcal{H}$, which is used to replace the common hash function in PDP scheme, 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_k, sets $\xi_k^{(2)} = H_{\xi^{(1)}}(\text{“}CN_k\text{”})$;
- *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 $\xi_{i,k}^{(3)} = H_{\xi_k^{(2)}}(\chi_i)$,[f] where B_i is the sequence number of block, R_i

[f]The index record is used to support dynamic data operations.

is the version number of updates for this block, and R_i is a random integer to avoid collision.

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

The above structure can be readily adopted into MAC-based, ECC or RSA schemes.[4,21] These schemes, built from collision-resistance signatures and the random oracle model, have the shortest query and response with public verifiability. They have some common characters to implement the CPDP framework in hybrid clouds: (i) 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; (ii) the verifier can verify the integrity of file in random sampling approach, which is of utmost importance for large or huge files; (iii) these schemes rely on homomorphic properties to aggregate the data and tags into a constant size response, which minimizes network communication; and (iv) the hierarchy structure provides a virtualization manner to conceal the storage details of multiple CSPs.

4.2. *Outsourced data storage structure*

In the storage layer, we define a common fragment structure that provides probabilistic verification of data integrity for outsourced storage. The fragment structure is a data structure that maintains a set of block-tag pairs, allowing searches, checks and updates in $O(1)$ time. An instance for this structure which is used in this scheme is showed in the storage layer: an outsourced file F is split into n blocks $\{m_1, m_2, \ldots, m_n\}$, and each block m_i is split into s sectors $\{m_{i,1}, m_{i,2}, \ldots, m_{i,s}\}$. The fragment structure consists of n block-tag pair (m_i, σ_i), where σ_i is a signature tag of block m_i generated by some owner's secrets $\tau = (\tau_1, \tau_2, \ldots, \tau_s)$. The multi-sector strategy is to improve the storage and verification efficiency. For example, for block length $|m_i|$ and signature length $|\sigma_i|$, the rate of signature-message is $\frac{|\sigma_i|}{|m_i|} = \frac{|\sigma_i|}{s|m_{i,j}|}$ if each block m_i is split into s sectors. It is obvious that as s becomes larger, higher encoding efficiency (rate) is achieved for the storage, communication, and verification. In addition, the data owner can choose different secrets τ for each file from the security point of view, but we expect that our CPDP scheme is secure enough even if the data owner only relies on such a secret τ to sign all his/her files.

Based on this fragment structure and hash index hierarchy, we propose a construction of CPDP scheme with $KeyGen$ and $TagGen$ algorithms, as follows:

- **KeyGen(1^κ):** 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 groups of a large prime order p, $|p| = O(\kappa)$. Makes a hash function $H_k(\cdot)$ public.
  (a) For a CSP $P_k \in \mathcal{P}$, chooses a random number $\delta_k \in_R \mathbb{Z}_p$ and computes $S = g^{\delta_k} \in \mathbb{G}$. Thus, $sk_k = \delta_k$ and $pk_k = (g, S_k)$.

(b) For a cloud client (data owner), chooses two random numbers $\alpha, \beta \in_R \mathbb{Z}_p$ 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 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 $\tau_1, \ldots, \tau_s \in \mathbb{Z}_p$ 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 the record χ_i[g] in χ for $i \in [1, n]$, then calculates the tag for each block m_i as

$$\begin{cases} \xi^{(1)} \leftarrow H_{\sum_{i=1}^s \tau_i}(\text{“}FN\text{”}), & \xi_k^{(2)} \leftarrow H_{\xi^{(1)}}(\text{“}CN_k\text{”}), \\ \xi_{i,k}^{(3)} \leftarrow H_{\xi_k^{(2)}}(\chi_i), & \sigma_{i,k} \leftarrow (\xi_{i,k}^{(3)})^\alpha \cdot \left(\prod_{j=1}^s u_j^{m_{i,j}}\right)^\beta, \end{cases}$$

where F_n is the file name and C_k is the CSP name of $P_k \in \mathcal{P}$. And then stores $\psi = (u, \xi^{(1)}, \chi)$ into TTP, and $\sigma_k = \{(m_i, \sigma_{i,j})\}_{\forall j=k}$ to $P_k \in \mathcal{P}$, where $u = (u_1, \ldots, u_s)$. Finally, the data owner saves the secret $\zeta = (\tau_1, \ldots, \tau_s)$.

Note that, each cloud service provider P_k in hybrid cloud is assigned a public–private key pair (pk_k, sk_k) for any $P_k \in \mathcal{P}$ in order to achieve identity authentication and secure communication.

In order to check data integrity, the fragment structure implements probabilistic verification, as follows: given a random chosen challenge (or query) $Q = \{(i, v_i)\}_{i\in_R I}$, where I is a subset of the block indices and v_i is a random coefficient. There exists an efficient algorithm to produce a constant-size response $(\mu_1, \mu_2, \ldots, \mu_s, \sigma')$, where μ_i comes from all $\{m_{k,i}, v_k\}_{k\in I}$ and σ' is from all $\{\sigma_k, v_k\}_{k\in I}$. This kind of probabilistic verification property will bring convenience to the development of our CPDP verification protocol.

4.3. *Collaborative provable data possession*

According to the above-mentioned architecture, four different network entities can be identified as follows: the verifier (V), the trusted third party (TTP), the organizer (O), and some cloud service providers (CSPs) in a 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 five-move interactive proof protocol showed in Fig. 6: (i) the organizer initiates the protocol and sends a commitment to the verifier; (ii) the verifier returns a challenge set of random index-coefficient pairs Q to the organizer; (iii) the organizer relays them into each P_i in $\mathcal{P}$ according to the exact position of each data block; (iv) each

[g]For $\chi_i =$ “B_i, V_i, R_i” in Sec. 4.1, we can set $\chi_i = (B_i = i, V_i = 1, R_i \in_R \{0, 1\}^*)$ at initial stage of CPDP scheme.

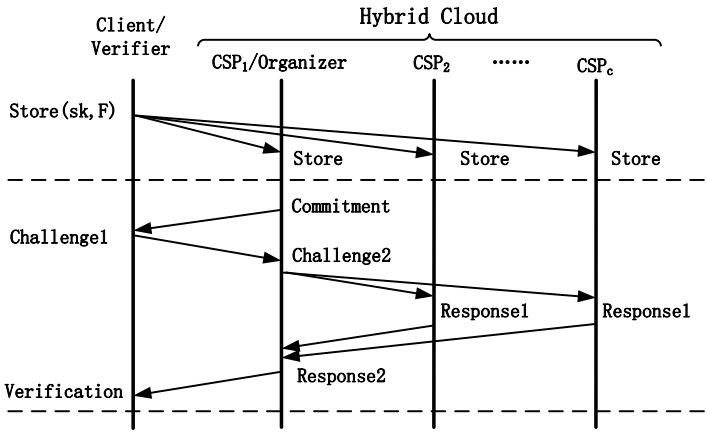


Fig. 6. The flowchart of verification process in our CPDP scheme.

P_i returns its response of challenge to the organizer; (v) 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 which CSPs or in what geographical locations their files reside.

We describe the details of verification protocol as follows:

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

  (a) **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;
  (b) **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;
  (c) **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 $\mathcal{P}$;
  (d) **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

$$\begin{cases} \sigma_k' \leftarrow S_k^{r_k} \cdot \prod\limits_{(i,v_i)\in Q_k} \sigma_{i,k}^{v_i} \in \mathbb{G}, \\ \mu_{j,k} \leftarrow \lambda_{j,k} + \sum\limits_{(i,v_i)\in Q_k} v_i \cdot m_{i,j} \in \mathbb{Z}, \\ \pi_{j,k} \leftarrow u_j^{\lambda_{j,k}} \in \mathbb{G}, \end{cases}$$

  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;

(e) **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 θ as

$$\sigma' \leftarrow \left(\prod_{P_k\in\mathcal{P}} \sigma'_k \cdot R_k^{-\delta_k}\right)^{\gamma}, \quad \mu'_j \leftarrow \sum_{P_k\in\mathcal{P}} \mu_{j,k}, \quad \pi'_j \leftarrow \prod_{P_k\in\mathcal{P}} \pi_{j,k}.$$

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: Finally, the verifier V can check that the response was correctly formed by checking that

$$e(\sigma', h) = e\left(\prod_{(i,v_i)\in Q} H_{\xi_k^{(2)}}(\chi_i)^{v_i}, H'_1\right) \cdot e\left(\prod_{j=1}^{s} u_j^{\mu'_j} \cdot {\pi'_j}^{-1}, H'_2\right).$$

Remark. Regardless of the length of verified files, the verifier executes the verification protocol with fix-length communication overheads (c, Q, θ) and constant computational overheads (just the last equation). Therefore, almost all computational tasks are completed by the organizer and the CSPs.

4.4. *Homomorphic verifiable response for collaborative PDP*

In this subsection, we describe the correctness of our proposed scheme based on mathematic homomorphism. 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 Homomorphic Verifiable Tags (HVTs)[4]:

Definition 4.3 (Homomorphic Verifiable Tags). Given two values σ_i and σ_j for two message m_i and m_j, anyone can combine them into a value σ' corresponding to the sum of the message $m_i + m_j$.

In our construction, this kind of homomorphism property is converted into more complex form because each block m_i is split into s sectors. That is, the sum of the message $m_i + m_j$ is replaced into $\sum_{e=1}^{s} \tau_e(m_{i,e} + m_{j,e})$, where $m_k = \{m_{k,1}, \ldots, m_{k,s}\}$ and $\tau_1, \ldots, \tau_s$ are some unknown secrets. Therefore, given the definition of signature $\sigma_{i,k} = (\xi_{i,k}^{(3)})^{\alpha} \cdot (\prod_{j=1}^{s} u_j^{m_{i,j}})^{\beta} = (\xi_{i,k}^{(3)})^{\alpha} \cdot (g^{\sum_{j=1}^{s} \tau_j m_{i,j}})^{\beta}$, the block's signature has the homomorphism property, that is,

$$\begin{aligned}\sigma_{i,k} \cdot \sigma_{j,k} &= (\xi_{i,k}^{(3)} \cdot \xi_{j,k}^{(3)})^{\alpha} \cdot \left(\prod_{e=1}^{s} u_e^{m_{i,e}+m_{j,e}}\right)^{\beta} \\ &= (\xi_{i,k}^{(3)} \cdot \xi_{j,k}^{(3)})^{\alpha} \cdot (g^{\sum_{e=1}^{s} \tau_e(m_{i,e}+m_{j,e})})^{\beta}.\end{aligned}$$

Similarly, for a certain CSP P_k, all block's signatures $\{\sigma_{i,k}\}$ can be aggregated into a signature by using the equation

$$\sigma'_k = S_k^{r_k} \cdot \prod_{(i,v_i)\in Q_k} \sigma_{i,k}^{v_i} = S_k^{r_k} \cdot \left(\prod_{(i,v_i)\in Q_k} \xi_{i,k}^{(3)v_i} \right)^{\alpha} \cdot (g^{\sum_{e=1}^{s} \tau_e \cdot \sum_{(i,v_i)\in Q_k} m_{i,e} v_i})^{\beta}.$$

When provable data possession is considered as a challenge–response protocol, we extend this notation to the concept of a homomorphic verifiable responses (HVRs), which is used to integrate multiple responses from the different CSPs in collaborative PDP scheme as follows:

Definition 4.4 (Homomorphic Verifiable Responses). A response is called HVRs in PDP protocol, if given two responses θ_i and θ_j for two challenges Q_i and Q_j from two CSPs, there exists an efficient algorithm to combine them into a response θ corresponding to the sum of the challenges $Q_i \bigcup Q_j$.

Homomorphic verifiable response is the key technique of collaborative PDP because it not only reduces the communication bandwidth, but also conceals the location of outsourced data in hybrid clouds. As we follow a similar method in the previous analysis, the responses (also considered as signatures), obtained from different CSPs, can be further aggregated into a response in terms of

$$\sigma' = \left(\prod_{P_k\in\mathcal{P}} \frac{\sigma'_k}{R_k^{\delta_k}} \right)^{\gamma} = \left(\prod_{P_k\in\mathcal{P}} \frac{S_k^{r_k} \cdot \prod_{(i,v_i)\in Q_k} \sigma_{i,k}^{v_i}}{R_k^{\delta_k}} \right)^{\gamma}$$

$$= \left(\prod_{P_k\in\mathcal{P}} \prod_{(i,v_i)\in Q_k} \sigma_{i,k}^{v_i} \right)^{\gamma} = \prod_{(i,v_i)\in Q} \sigma_i^{v_i\cdot\gamma},$$

where, $S_k^{r_k} = g^{\delta_k \cdot r_k} = R_k^{\delta_k}$. Similarly, for $j \in [1, s]$, the file blocks are also aggregated into s values $\{\mu'_1, \ldots, \mu'_s\}$, in which each value μ'_j can be defined as

$$\mu'_j = \sum_{P_k\in\mathcal{P}} \mu_{j,k} = \sum_{P_k\in\mathcal{P}} \left(\lambda_{j,k} + \sum_{(i,v_i)\in Q_k} v_i \cdot m_{i,j} \right)$$

$$= \sum_{P_k\in\mathcal{P}} \lambda_{j,k} + \sum_{P_k\in\mathcal{P}} \sum_{(i,v_i)\in Q_k} v_i \cdot m_{i,j}$$

$$= \lambda_j + \sum_{(i,v_i)\in Q} v_i \cdot m_{i,j},$$

where λ_j is an unknown secret value for the verifier, and all values $\{\lambda_1, \ldots, \lambda_s\}$ are aggregated into a commitment by using the equation

$$\pi'_j = \prod_{P_k\in\mathcal{P}} \pi_{j,k} = \prod_{P_k\in\mathcal{P}} u_j^{\lambda_{j,k}} = u_j^{\sum_{P_k\in\mathcal{P}} \lambda_{j,k}} = u_j^{\lambda_j}.$$

Finally, according to the above-mentioned responses, the final verification equation can be elaborated as follows:

$$
\begin{aligned}
e(\sigma', h) &= e\left(\prod_{(i,v_i)\in Q} (\xi_i^{(3)})^{v_i}, h\right)^{\gamma\cdot\alpha} \cdot e(g,h)^{\gamma\cdot\beta\sum_{(i,v_i)\in Q}\sum_{j=1}^{s}\tau_j\cdot v_i\cdot m_{i,j}} \\
&= e\left(\prod_{(i,v_i)\in Q} (\xi_i^{(3)})^{v_i}, h^{\alpha\cdot\gamma}\right) \cdot e\left(\prod_{j=1}^{s} u_j^{\gamma\cdot\sum_{(i,v_i)\in Q} v_i\cdot m_{i,j}}, h^{\beta}\right) \\
&= e\left(\prod_{(i,v_i)\in Q} (\xi_i^{(3)})^{v_i}, h^{\alpha\cdot\gamma}\right) \cdot \prod_{j=1}^{s} e(u_j^{\mu_j'-\lambda_j}, h^{\beta}) \\
&= e\left(\prod_{(i,v_i)\in Q} H_{\xi_k^{(2)}}(\chi_i)^{v_i}, H_1'\right) \cdot e\left(\prod_{j=1}^{s} u_j^{\mu_j'}\cdot {\pi_j'}^{-1}, H_2'\right).
\end{aligned}
$$

This equation means that our CPDP scheme is an efficient interactive proof system.

5. Security and Performance Analysis

In this section, we first give a brief security analysis of our CPDP construction. This analysis is directly derived from security requirements of multi-prover zero-knowledge proof system (MP-ZKPS). Next, we analyze the performance of CPDP scheme from the efficiency point of view of probabilistic verification. Finally, to validate the effectiveness of our scheme, we introduce a prototype of CPDP-based audit system and present experimental results.

5.1. *Security analysis for CPDP scheme*

The collaborate integrity verification for distrusted outsourced data, in essence, is a multi-prover interactive proof system (IPS), so that the correspondence construction should satisfy the security requirements of IPS. Moreover, in order to ensure the security of verified data, this kind of construction is also a multi-prover zero-knowledge proof (MP-ZKP) system,[7,15] which can be considered as an extension of the notion of an IPS. Roughly speaking, the scenario of MPZKP is that a polynomial-time bounded verifier interacts with several provers whose computational power is unlimited. Given an assertion L, such a system satisfies following three properties: (i) **Completeness:** whenever $x \in L$, there exists a strategy for provers that convinces the verifier that this is the case; (ii) **Soundness:** whenever $x \notin L$, whatever strategy the provers employ, they will not convince the verifier that $x \in L$; (iii) **Zero-knowledge:** no cheating verifier can learn anything other than the veracity of the statement. Since this construction is directly derived from MPZKP model, the soundness and zero-knowledge properties can protect our construction

from various attacks as follows:

- **Security for tag forging attack:** The soundness means that it is infeasible to fool the verifier into accepting false statements. It is also regarded as a stricter notion of unforgeability for the file tags. To be exact, soundness is defined as follows: for every "invalid" tag $\sigma^* \notin TagGen(sk, F)$, there does exist an interactive machine P^* which can pass verification with any verifier V^* with noticeable probability. In order to prove the non-existence of P^*, to the contrary, we can make use of P^* to construct a knowledge extractor $\mathcal{M}$, which gets the common input (pk, ψ) and rewindable black-box access to 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 $G^{ab} \in \mathbb{G}$. This means that the prover cannot forge the file tags or tamper with the data if soundness property holds.
- **Security for data leakage attack:** In order to protect the confidentiality of the checked data, we are more concerned about the leakage of private information in the verification process. In Sec. 2.4, we have shown 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 our construction. First, randomness is adopted into the CSP's response in order to resist Attack 2 and 4 in Appendixes A and B, i.e. the random integer $\lambda_{j,k}$ is adopted into the response $\mu_{j,k}$, i.e. $\mu_{j,k} = \lambda_{j,k} + \sum_{(i,v_i)\in Q_k} v_i \cdot m_{i,j}$. This means that the cheating verifier cannot obtain $m_{i,j}$ from $\mu_{j,k}$ because he does not know the random integer $\lambda_{j,k}$. At the same time, a random integer γ is also introduced to randomize the verification tag σ, i.e. $\sigma' \leftarrow (\prod_{P_k\in\mathcal{P}} \sigma'_k \cdot R_k^{-s})^{\gamma}$. Thus, the tag σ cannot reveal to the cheating verifier in terms of randomness.

  Based on this idea, we need to prove the following theorem according to the formal definition of zero-knowledge, in which every cheating verifier has some simulator that, given only the statement to be proven (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier. Actually, zero-knowledge is a property that captures (private or public) CSP's robustness against attempts to gain knowledge by interacting with it. For our construction, we make use of the zero-knowledge property to guarantee the security of data blocks and signature tags.

5.2. *Performance analysis of probabilistic verification*

In our construction, the integrity verification achieves the detection of CSP servers misbehavior in a random sampling mode (called probabilistic verification) in order to reduce the workload on the server. In the probabilistic verification of common PDP scheme (which involves one CSP), the detection probability P of disrupted blocks is an important parameter to guarantee that these blocks can be detected in time. Assume the CSP modifies e blocks out of the n-block file. The probability of disrupted blocks is $\rho_b = \frac{e}{n}$. Let t be the number of queried blocks for a challenge

in the verification protocol. We have detection probability

$$P(\rho_b, t) = 1 - \left(\frac{n-e}{n}\right)^t = 1 - (1-\rho_b)^t.$$

Hence, the number of queried blocks is $t = \frac{\log(1-P)}{\log(1-\rho_b)} \approx \frac{P \cdot n}{e}$ for a sufficiently large n.[h] This means that the number of queried blocks t is directly proportional to the total number of file blocks n for the constant P and e.

For a PDP scheme with *fragment structure*, given a file with $sz = n \cdot s$ sectors and the probability ρ of sector corruption, the detection probability of verification protocol has $P \geq 1 - (1-\rho)^{sz \cdot \omega}$, where ω denotes the sampling probability in the verification protocol. We can obtain this result as follows: because $\rho_b \geq 1 - (1-\rho)^s$ is the probability of block corruption with s sectors in common PDP scheme, the verifier can detect block errors with probability $P \geq 1 - (1-\rho_b)^t \geq 1 - ((1-\rho)^s)^{n \cdot \omega} = 1 - (1-\rho)^{sz \cdot \omega}$ for a challenge with $t = n \cdot \omega$ index-coefficient pairs.

Next, we extend the one-CSP PDP scheme into multi-CSPs CPDP scheme as follows: given a file with $sz = n \cdot s$ sectors and ω denotes the sampling probability in the verification protocol. For a hybrid cloud $\mathcal{P}$, the detection probability of CPDP scheme has

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

where r_k denotes the proportion of data blocks in the kth CSP, ρ_k denotes the probability of file corruption in the kth CSP, and $r_k \cdot \omega$ denotes the possible number of blocks queried by the verifier in the kth CSP. Furthermore, we observe the ratio of queried blocks in the total file blocks w under different detection probabilities. Based on above analysis, it is easy to find that this ratio holds the equation

$$w = \frac{\log(1-P)}{sz \cdot \sum_{P_k \in \mathcal{P}} r_k \cdot \log(1-\rho_k)}.$$

However, the estimation of w is not an accurate measurement.

In most cases, we adopt the probability of disrupted blocks to describe the possibility of data loss, damage, forgery or unauthorized changes. When this probability ρ_b is a constant probability, the verifier can detect severe misbehavior with a certain probability P by asking proof for a constant amount of blocks for PDP or $t = \frac{\log(1-P)}{s \cdot \sum_{P_k \in \mathcal{P}} r_k \cdot \log(1-\rho_k)}$ for CPDP, independently of the total number of file blocks.[4]

5.3. *CPDP for integrity audit services*

We apply our collaborative PDP scheme to construct an audit system architecture for outsourced data in hybrid clouds by replacing TTP with a third party auditor

[h] In terms of $(1 - \frac{e}{n})^t = 1 - \frac{e \cdot t}{n}$, we have $P = 1 - (1 - \frac{e \cdot t}{n}) = \frac{e \cdot t}{n}$.

(TPA) as shown in Fig. 3. In this architecture, data owners and granted clients need to dynamically interact with CSP to access or update their data for various application purposes. However, we neither assume that CSP is trusted to guarantee the security of stored data, nor assume that data owners have the ability to collect the evidence of the CSP's fault after errors have been found. Hence TPA, as a trust third party (TTP), is used to ensure the storage security of outsourced data. We assume the TPA is reliable and independent, and thus has no incentive to collude with either CSPs or users during the auditing process.

- TPA should be able to make regular checks on the integrity and availability of these delegated data at appropriate intervals;
- TPA should be able to organize, manage, and maintain the outsourced data instead of data owners, and support dynamic data operations for the granted user;
- TPA should be able to take the evidences for the disputes about the inconsistency of data in terms of authentic records for all data operations.

To enable privacy-preserving public auditing for cloud data storage under this architecture, our protocol design should achieve following security and performance guarantee:

- **Public auditability:** to allow TPA (or the other clients with help of TPA) to verify the correctness of the cloud data on demand without retrieving a copy of the whole data or introducing additional online burden to cloud users;
- **Verification correctness:** to ensure there exists no cheating CSP that can pass the audit from TPA without indeed storing users' data intact;
- **Verification transparency:** to enable TPA with secure and efficient auditing capability to cope with auditing delegations from possibly large number of different CSPs simultaneously;
- **Privacy-preserving:** to ensure that there exists no way for TPA to derive users' data from the information collected during the auditing process; and
- **Lightweight:** to allow TPA to perform auditing with minimum storage, communication and computation overhead, and to support batch auditing with a long enough period.

To validate the effectiveness and efficiency of our proposed approach, we have implemented a prototype of an audit system based on our proposed solution. We simulate the audit service and the storage service by using two local IBM servers with two Intel Core 2 processors at 2.16 GHz and 500M RAM running Windows Server 2003. These servers were connected via 250 MB/s of network bandwidth. Our audit scheme was also deployed in these servers. Using GMP and PBC libraries, we have implemented a cryptographic library upon which our scheme can be constructed. This C library contains approximately 5200 lines of codes and has been

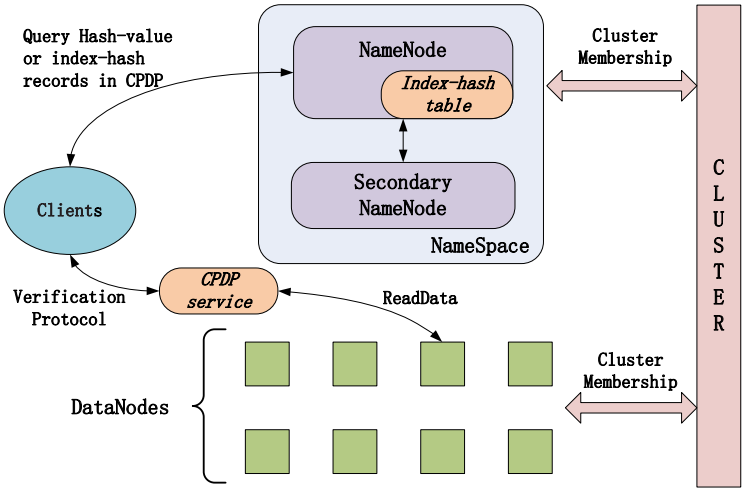


Fig. 7. An example of hash index hierarchy in Hadoop distributed file system (HDFS).

tested on Windows and Linux platforms. The elliptic curve utilized in the experiment is a MNT curve, with base field size of 160 bits and the embedding degree 6. The security level is chosen to be 80 bit, which means $|p| = 160$.

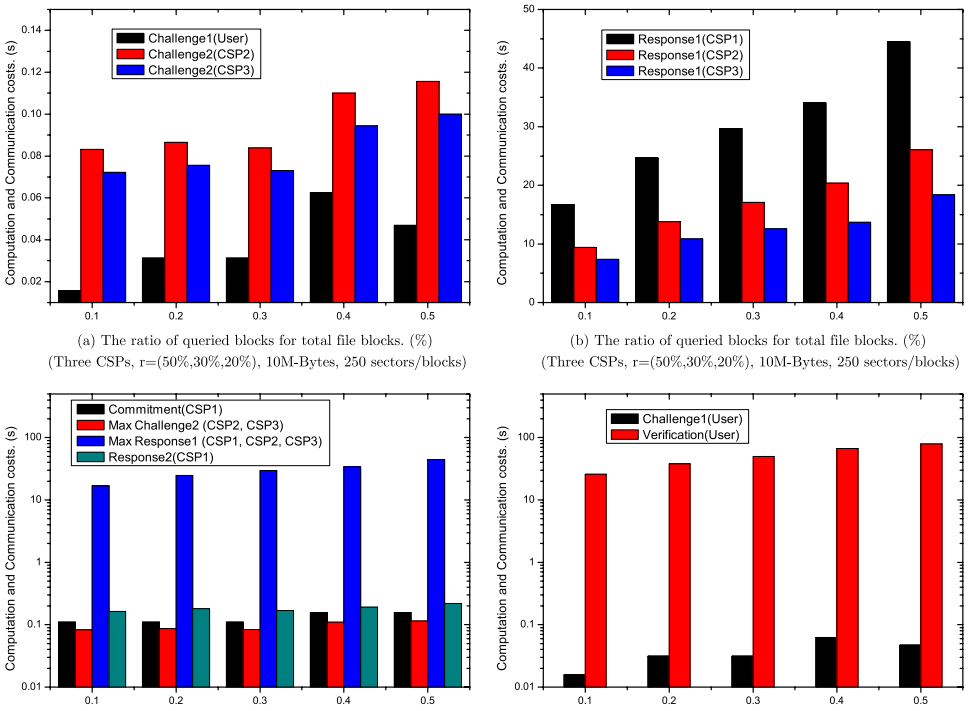
More importantly, we incorporated this prototype on CPDP scheme into a virtualization infrastructure of cloud-based storage service.[2] In Fig. 7, we show an example of Hadoop distributed file system (HDFS),[i] which a distributed, scalable, and portable file system.[19] HDFS' architecture is composed of NameNode and DataNode, where NameNode maps a file name to a set of indexes of blocks and DataNode indeed stores data blocks. To support the CPDP, the index-hash table and the metadata of NameNode should be integrated together to provide an enquiry service for the hash value $\xi_{i,k}^{(3)}$ or index-hash record χ_i. Based on the hash value, the clients can implement the verification protocol via CPDP services. Hence, it is easy to replace the checksum methods with the CPDP scheme for anomaly detection in current HDFS.

In our experiments, we maintained a simple global Index-Hash table to manage the whole file in a hybrid cloud. The Index-Hash table is defined as follows: $\chi_i =$ (BlkID: 16-bit, Pst: 20-bit, Property: 4-bit, Hash: 40-bit), where BlkID is used to store the order number of block, Pst is to store (CloudNo:4-bit, BlkNo:16-bit), Property is to store some property values each of which is denoted by one bit, and Hash is to store $\xi_{i,k}^{(3)}$ which is computed from a collision-resistant hash function. When the length of each block is 5K Bytes, the radio of storage is $10/5\text{K} = 0.2\%$. For a 10 MB file, the storage of table is 20K Bytes. In our experiments we do not

[i]Hadoop can enable applications to work with thousands of nodes and petabytes of data, and it has been adopted by currently mainstream cloud platforms from Apache, Google, Yahoo, Amazon, IBM and Sun.

attempt to change the structure of the HDFS, so a CPDP Daemon is constructed to access HDFS system by using (CloudNo, BlkNo), where BlkNo is a order number of blocks in terms of reordering of blocks in a CSP. In addition, the Index-Hash table is stored in a public computer and is accessed by all CSPs.

Our experiments were carried out in a hybrid cloud environment consisting of three CSPs: CSP1 CSP2, and CSP3, respectively. The CSP1 is considered as an organizer of CPDP protocol. The object in our experiments is a 10 MB file and 250 sectors per block, where the length of each block is 5K Bytes and the file includes 2000 blocks. In accordance with the proportion of 50%, 30%, 20%, we allocated the data blocks to three CSPs, respectively. The experimental results are shown in Fig. 8. The workflow of three CSPs is the same as that shown in Fig. 6, and we also used the same name to identify each stage of CPDP verification. After receiving the Commitment from CSP1, the user generated and sent a Challenge1 query $Q = \{(i, v_i)\}_{i \in I}$ into CSP1. CSP1 decomposed the query into three sub-queries, Challenge2(CSP1), Challenge2(CSP2), and Challenge2(CSP3). In Fig. 8(a), we show the overheads of computation and communication of Challenge1, Challenge2(CSP2), and Challenge2(CSP3). The costs of Challenge2(CSP2) and Challenge2(CSP3) are larger than that of Challenge1 because CSP1 need to search the



(a) The ratio of queried blocks for total file blocks. (%) (Three CSPs, r=(50%,30%,20%), 10M-Bytes, 250 sectors/blocks)

(b) The ratio of queried blocks for total file blocks. (%) (Three CSPs, r=(50%,30%,20%), 10M-Bytes, 250 sectors/blocks)

(c) The ratio of queried blocks for total file blocks. (%) (Three CSP, r=(50%,30%,20%), 10M-Bytes, 250 sectors/blocks)

(d) The ratio of queried blocks for total file blocks. (%) (Three CSP, r=(50%,30%,20%), 10M-Bytes, 250 sectors/blocks)

Fig. 8. Experimental results in a hybrid cloud environment.

index table for classifying sub-query $Q_k = \{(i, v_i)\}_{m_i \in P_k}$ and delivering them into other CSPs. We do not extract the time overhead of Challenge2(CSP1) because this process is included into the whole search process and does not have a delivering process. The time difference between Challenge2(CSP2) and Challenge2(CSP3) comes from the difference of communication cost and experiment equipments. Next, after receiving the Challenge2, each CSP would generate corresponding response, such Response1(CSP1), Response1(CSP2), and Response1(CSP3). In Fig. 8(b), we show the time overheads of computation and communication of these three responses. The overhead of Response1 is proportional to the number of queried blocks, which is related to the radio of queried blocks for total file blocks (x-axis) in our experiments. Note that, all Response1s have the same communication overhead because our CPDP has the constant-size Response1. After that, CSP1 aggregate all Response1s into Response2.

In Fig. 8(c), we show the result of comparison of Commitment, Challenge2(CSP2 and CSP3), Response1(CSP1, CSP2, CSP3) and Response2(CSP1), where Challenge2 is computed by making the maximum of Challenge2(CSP2) and Challenge2(CSP3), Response1 is processed by the same way. From this figure, we can see that the time overheads of Response1 are much larger than the other processes. Note that, all these processes are executed by three CSPs. Finally, the user performs the final verification (called Verification(User)). We show the overheads of Challenge(User) and Verification(User) in Fig. 8(d), all of which are executed by the user. From this figure, the time overheads of Verification(User) is similar to those of Response1(CSP1, CSP2, CSP3) in Fig. 8(c), and it is proportional to the number of queried blocks.

More importantly, from the above results we can notice that the processes of Challenge2 and Response1 are run in parallel, and around half of that time can be saved in this kind of parallel manner. Such a time saving is directly associated with the distribution of stored blocks in CSPs, as well as the distribution of queried blocks. Hence, our CPDP scheme provides an effective collaborative mechanism to reduce the total time overheads.

6. Related Work

Traditional cryptographic technologies for data integrity and availability, based on hash functions and signature schemes,[11,12] cannot work on the outsourced data without a local copy of data. Moreover, these traditional methods are not the practical solutions for data validation by downloading them due to the expensive communications, especially for large-size files. To check the availability and integrity of the stored data in cloud storage, researchers have proposed two basic approaches called Provable Data Possession (PDP)[4] and Proofs of Retrievability (POR).[22] Ateniese *et al.*[4] 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 cost. 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 dynamic scenarios because of the dependence on the index of blocks. Moreover, they do not fit for hybrid clouds due to the loss of homomorphism in the verification process.

Unfortunately, none of these schemes is aware of dynamic data operations such as query, insertion, modification, and deletion. To support dynamic data operations, Ateniese *et al.* have developed a dynamic PDP solution called Scalable PDP.[5] 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 in *a priori*. Also, one cannot perform block insertions anywhere. Based on this work, Erway *et al.*[6] 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, retains the drawback of Scalable PDP, 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. Juels and Kaliski[22] 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[21] 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 cost for t challenge blocks, but their solution is also static and exists the leakage of data blocks in the verification process. Wang *et al.*[13] presented a dynamic scheme with $O(\log n)$ cost by integrating the above CPOR scheme and Merkle Hash Tree (MHT) in DPDP. Furthermore, several POR schemes and models have been proposed recently including.[23,24] Since the response of challenges has homomorphic property, the above schemes (especially CPOR schemes) can leverage the PDP construction in hybrid clouds.

7. Conclusions and Future Work

In this paper, we addressed the construction of collaborative integrity verification mechanism for distributed data outsourcing in hybrid clouds. Based on the homomorphic verifiable responses and hash index hierarchy, we proposed a collaborative provable data possession scheme to support dynamic scalability on multiple (private and public) cloud storage providers. We showed that our scheme provided all security properties required by the zero-knowledge interactive proof system, so that it can resist various attacks even if it is deployed as a public verification service. Furthermore, our performance analysis indicated that our proposed solution only

incurs a small constant amount of communication and computational overheads, which are significantly lower than non-cooperative solutions. More importantly, our solution conceals the details of outsourced storage to reduce the burden on verifiers, and verifiers cannot even distinguish whether the verified data is in a hybrid cloud or a single cloud. Hence, our proposed method can be considered as a candidate technology to replace traditional hash method in cloud storage systems or distributed file systems.

As part of future work, we would extend our work to explore more effective and practical CPDP constructions. First, from our experiments we found that the performance of CPDP scheme, especially for large files, is seriously affected by the bilinear mapping operations due to its high complexity. To address this problem, RSA-based constructions may be adopted, but existing RSA-based schemes still have many restrictions on the system performance and security. Next, from a practical point of view, we still need to address several issues about integrating our CPDP scheme smoothly with existing systems, for example, how to match index-hash hierarchy with HDFSs two-layer name space, how to match index structure with cluster-network model, and how to dynamically update the CPDP parameters according to HDFS specific requirements. Finally, it is still a challenging problem for the generation of tags with the length irrelevant to the size of data blocks. We would explore a mechanism to provide the support of variable-length block verification.

Acknowledgments

The authors thank the reviewers for many helpful comments and suggestions. Many thanks to our collaborators, Yujing Han and Shinmin Chen, at Peking University for verifying our scheme by C++ Language. The work of Y. Zhu and S. Wang was supported by the National Natural Science Foundation of China (Project No. 61170264 and No. 10990011). This work of G.-J. Ahn and H. Hu was partially supported by the grants from US National Science Foundation (NSF-IIS-0900970 and NSF-CNS-0831360).

Appendix A. Attacks for Public Blocked Scheme

The client breaks a (possibly encoded) file F into n blocks $m_1, \ldots, m_n \in \mathbb{Z}_p$ 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

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

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

Theorem A.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 kth user's challenge $Q^{(k)}$, where we fill $v_i = 0$ to extend the challenge coefficients, that is, $v_i = 0$ for any $(i, v_i) \notin Q$. Such that the adversary gets the responses $\{(\sigma^{(1)}, \mu^{(1)}), \ldots, (\sigma^{(1)}, \mu^{(n)})\}$ after he finishes n times queries. These responses can generate the equations

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

where, $v_i^{(k)}$ is known for all $i \in [1, n]$ and $k \in [1, n]$. The adversary can compute $f = (m_1, \ldots, m_n)$ by solving the equations. Similarly, the adversary can get all tags $\sigma_1, \ldots, \sigma_n$ 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. □

Theorem A.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 occur in a variety of cases, but they have a common feature that the same hash value $H(i)$ been used at least two times. For example, the adversary gets two data-tag pairs (m_i, σ_i) and (m_i', σ_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. □

Appendix B. Attacks for Public Fragmented Scheme

Given a file F, the client split F into n blocks $(m_1, \ldots, m_n)$ and each block m_i is also split into s sectors $(m_{i,1}, \ldots, m_{i,s}) \in \mathbb{Z}_P^s$ 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 $u_1, \ldots, u_s \in_R \mathbb{G}$ as the verification information $t = (F_n, u_1, \ldots, u_s)$, where F_n is the file name.

For each $i \in [1, n]$, the tag at the ith 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 $\mu = (\mu_1, \ldots, \mu_s)$, where $\mu_j \leftarrow \sum_{(i,v_i) \in Q} v_i m_{i,j}$. The verification equation is

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

This scheme is not secure due to the leakage of outsourced data and the forging of tags, as follows:

Theorem B.1. *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 in Appendix A. Let s be the number of sectors. Given n times challenges $(Q^{(1)}, \ldots, Q^{(n)})$ and their the results $((\sigma'^{(1)}, \mu^{(1)}), \ldots, (\sigma'^{(n)}, \mu^{(n)}))$, $\mu^{(k)} = (\mu_1^{(k)}, \ldots, \mu_s^{(k)})$ 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 $\{m_{1,i}, \ldots, m_{n,i}\}$. 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 $\sigma_1, \ldots, \sigma_n$ by using $\sigma'^{(1)}, \ldots, \sigma'^{(n)}$. □

Theorem B.2. *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 two 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 in Appendix A. 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, $(m_1, \ldots, m_{2s})$ and $(m'_1, \ldots, m'_{2s})$, such that $\sigma_i = (H(Fn, i) \cdot \prod_{j=1}^{s} u_j^{m_{i,j}})^x$, $\sigma'_i = (H(Fn, i) \cdot \prod_{j=1}^{s} u'^{m'_{i,j}}_j)^x$ for $i \in [1, 2s]$. The adversary computes $\Delta_1, \ldots, \Delta_{2s}$ by using $\Delta_i = \sigma_i \cdot \sigma_i'^{-1} = \prod_{j=1}^{s} (u_j^{m_{i,j}} \cdot (u'^{m'_{i,j}}_j)^{-1})^x$. These values can generate the

following system of equations

$$(\Delta_1, \ldots, \Delta_{2s})^T = M \odot (u_1^x, \ldots, u_s^x, u_1'^x, \ldots, u_s'^x)^T,$$

where, $\odot$ denotes the operation $g^{ax+by} = (x, y) \odot (g^a, g^b)^T$, 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 $u'^x_i = \prod_{j=1}^{2n} (\frac{\sigma_j}{\sigma'_j})^{d_{s+i,j}}$ 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 $\sigma_k^* = H(Fn, k)^x \cdot \prod_{j=1}^{s} (u_j^x)^{m^*_{k,j}} = \sigma_k \cdot \prod_{j=1}^{s} (u_j^x)^{m^*_{k,j} - m_{k,j}}$. □

References

1. Y. Zhu, H. Hu, G. J. Ahn, Y. Han and S. Chen, Collaborative integrity verification in hybrid clouds, in *Proc. 7th Int. Conf. Collaborative Computing: Networking, Applications and Worksharing* (CollaborateCom, Orlando, FL, USA, 15–18 October, 2011), pp. 191–200.
2. B. Sotomayor, R. S. Montero, I. M. Llorente and I. T. Foster, Virtual infrastructure management in private and hybrid clouds, *IEEE Internet Comput.* **13**(5) (2009) 14–22.
3. A. Brown and J. S. Chase, Trusted platform-as-a-service: A foundation for trustworthy cloud-hosted applications, in *CCSW*, eds. C. Cachin and T. Ristenpart (ACM, 2011), pp. 15–20.
4. 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 Conf. Computer and Communications Security* (CCS, Alexandria, Virginia, USA, 28–31 October, 2007), pp. 598–609.
5. G. Ateniese, R. D. Pietro, L. V. Mancini and G. Tsudik, Scalable and efficient provable data possession, in *Proc. 4th Int. Conf. Security and Privacy in Communication Networks*, *SecureCommunication* (ACM, New York, NY, USA, 2008), pp. 1–10.
6. C. C. Erway, A. Küpçü, C. Papamanthou and R. Tamassia, Dynamic provable data possession, in *ACM Conf. Computer and Communications Security* (CCS, Chicago, IL, USA, 9–13 November, 2009), pp. 213–222.
7. L. Fortnow, J. Rompel and M. Sipser, On the power of multi-prover interactive protocols, in *Theoretical Computer Science* (1988), pp. 156–161.
8. J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson and A. Venkataramani, Consensus routing: The internet as a distributed system, in *NSDI'08: Proc. 5th USENIX Symp. Networked Systems Design and Implementation* (Berkeley, CA, USA, USENIX Association, 2008), pp. 351–364.
9. 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*) (Monte Verità, Switzerland, May 18–20, 2009), pp. 1–10.
10. S. Pallickara, J. Ekanayake and G. Fox, Granules: A lightweight, streaming runtime for cloud computing with support, for map-reduce, in *CLUSTER* (2009), pp. 1–10.

11. H.-C. Hsiao, Y.-H. Lin, A. Studer, C. Studer, K.-H. Wang, H. Kikuchi, A. Perrig, H.-M. Sun and B.-Y. Yang, A study of user-friendly hash comparison schemes, in *ACSAC* (2009), pp. 105–114.
12. A. R. Yumerefendi and J. S. Chase, Strong accountability for network storage, in *FAST* (USENIX, 2007), pp. 77–92.
13. 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* (2009), pp. 355–370.
14. O. Goldreich, *Foundations of Cryptography: Basic Tools* (Cambridge University Press, 2001), Vol. 1, Basic Techniques.
15. Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn and H. Hu, Zero-knowledge proofs of retrievability, *Science China*: *Information Sciences* (*Series F*) **54**(8) (2011) 1608–1617.
16. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica and M. Zaharia, Above the clouds: A Berkeley view of cloud computing, EECS Department, University of California, Berkeley, Technical Report UCB/EECS-2009-28, Feb (2009).
17. D. Boneh and M. Franklin, Identity-based encryption from the weil pairing, in *Advances in Cryptology (CRYPTO'2001)*, Vol. 2139 of LNCS (Springer, 2001), pp. 213–229.
18. V. Ramasubramanian and E. G. Sirer, The design and implementation of a next generation name service for the internet, in *SIGCOMM'04: Proc. 2004 Conf. Applications*, *Technologies*, *Architectures*, *and Protocols for Computer Communications* (ACM, New York, NY, USA, 2004), pp. 331–342.
19. A. Bialecki, M. Cafarella, D. Cutting and O. O'Malley, Hadoop: A framework for running applications on large clusters built of commodity hardware, Tech. Rep. (2005).
20. S. Ghemawat, H. Gobioff and S.-T. Leung, The google file system, in *SOSP'03: Proc. Nineteenth ACM Symp. Operating Systems Principles* (ACM, New York, NY, USA, 2003), pp. 29–43.
21. H. Shacham and B. Waters, Compact proofs of retrievability, in *ASIACRYPT* (Springer, 2008), pp. 90–107.
22. A. Juels and B. S. K. Jr., Pors: Proofs of retrievability for large files, in *ACM Conf. Computer and Communications Security* (CCS, Alexandria, Virginia, USA, 28–31 October, 2007), pp. 584–597.
23. K. D. Bowers, A. Juels and A. Oprea, Hail: A high-availability and integrity layer for cloud storage, in *ACM Conf. Computer and Communications Security* (CCS, Chicago, IL, USA, 9–13 November, 2009), pp. 187–198.
24. Y. Dodis, S. P. Vadhan and D. Wichs, Proofs of retrievability via hardness amplification, in *TCC* (Springer, 2009), pp. 109–127.