Collaborative Integrity Verification in Hybrid Clouds

Yan Zhu[1,2], Hongxin Hu[3], Gail-Joon Ahn[3], Yujing Han[1], Shimin Chen[1,2]
[1]Institute of Computer Science and Technology, Peking University, Beijing 100871, China
[2]Beijing Key Laboratory of Internet Security Technology, Peking University, Beijing 100871, China
[3]Laboratory of Security Engineering for Future Computing (SEFCOM), Arizona State University, Tempe, AZ 85287, USA
Email: {yan.zhu,yjhan,smchen}@pku.edu.cn, {hxhu,gahn}@asu.edu

***Abstract*—A hybrid cloud is a cloud computing environment in which an organization provides and manages some internal resources and the 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. In this paper, we address the construction of a collaborative integrity verification mechanism in hybrid clouds to support the scalable service and data migration, in which 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.**

***Index Terms*—Integrity Verification, Multi-Prover, Collaborative, Hybrid Clouds.**

I. Introduction

Cloud computing has become a faster profit growth point in recent years by providing a comparably low-cost, scalable, position-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 forces 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) [1], 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.

With the growing popularity of clouds, the tools and technologies for hybrid clouds are emerging recently, such as the Platform VM Orchestrator [1], VMware vSphere [2], and Ovirt [3]. 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, the confidential data in an enterprise may be illegally accessed by using remote interfaces, or the relevant data and archives are lost or tampered with when they are stored into an uncertain storage pool outside the enterprise. Therefore, it is indispensable for cloud service providers (CSPs) to provide secure management techniques to ensure their storage services.

Provable data possession (PDP) [2] 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 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 [3] and Dynamic PDP [4], 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 (see Section II 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 the 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 main techniques adopted in our approach: (1) fragment structure, (2) hash index hierarchy (HIH), and (3) homomorphic verifiable response (HVR). We argue that it is possible to construct a collaborative PDP (CPDP) scheme without compromising data privacy based on modern cryptographic techniques, such as multi-prover zero-knowledge proof system (MPZKP) [5].

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

[1] www.platform.com/Products/platform-vm-orchestrator
[2] www.vmware.com/products/vsphere
[3] http://ovirt.org

new hash index hierarchy is proposed to realize transparent property for the clients to store and manage their resources in hybrid clouds.

We also evaluate the performance of our CPDP scheme from four aspects: Firstly, we provide a brief security analysis of our scheme; Secondly, we 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. Next, 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 Section II, we address our motivation and research objectives. We describe the background techniques, which are adopted in our construction, in Section III. Section IV describes the security and performance analysis of our solution. We discuss the related work in Section V and Section VI concludes this paper.

II. 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 the following challenging questions that need to be addressed, which help us define our objectives in this paper.

1. Why Need Integrity Checking of Outsourced Data?

Storage outsourcing in clouds has become a new profit growth point by providing a comparably low-cost, scalable, location-independent platform for managing clients' data. However, security is critical for such convenient storage services due to the following reasons: the 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 towards the cloud users; and, furthermore, the dispute occasionally suffers from a lack of trust on CSPs. Consequently, the behaviors of CSPs may not be known by the cloud users, even if this dispute may result from the users' own improper operations. Therefore, it is crucial for a CSP to offer an efficient verification on the integrity and availability of the stored data to enable the credibility of cloud services.

We expect that the size of outsourced data cannot be too small to influence the verification efficiency. All outsourced data would require additional storages for the verification parameters which must be stored in a Trusted Third Party (TTP). Thus, from a practical standpoint, the outsourced data can be either a large file, a database, or a set of files in an application system including softwares, scripts, Web pages, snapshots, and so on. Especially, it is critical to check the integrity of application softwares in public clouds even if sensitive data are stored in private clouds. For instance, an attacker can modify application softwares or scripts, or load a trojan into a snapshot of virtual machine (VM) to compromise the applications in a cloud.

2. Why Need a New Mechanism for Ensuring the Security of Outsourced Data in Hybrid Clouds?

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. As cloud computing has been rapidly adopted, the hybrid model will be more prevalent for a number of reasons [1]: 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 the organization's specific goals (high availability, server consolidation to minimize power usage, and so on); and to meet an organization's changing resource needs.

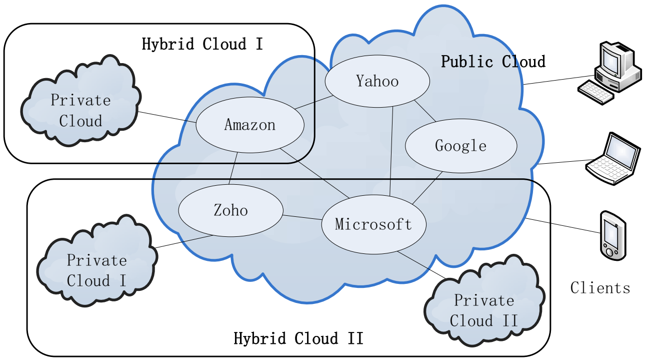

Fig. 1. Types of cloud computing: private cloud, public cloud and hybrid cloud.

In hybrid clouds, one of core design principles is dynamic scalability, which guarantees cloud storage services to handle growing amounts of application data in a flexible manner. By employing virtualization, such as VIM, 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 other services in private clouds, but the application scripts, intermediate data and results are executed and stored in public clouds [6], [7]. Since this new collaborative paradigm still faces a variety of security concerns, it is necessary to develop a new method for ensuring the security of outsourced data in hybrid clouds.

3. Are Existing PDP Schemes Efficient for Hybrid Clouds?

The traditional cryptographic technologies for data integrity and availability, based on Hash functions and signature schemes [8], [9], cannot support the outsourced data without a local copy of data. It is evidently impractical for a cloud storage service to download the whole data for data validation due to the expensiveness of communication, especially, for large-size files. Recently, several PDP schemes are proposed to address this issue. In fact, PDP is essentially an interactive proof 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 public clouds, but these schemes are not suitable for a hybrid cloud environment since they were originally constructed based on a two-party interactive proof system. For a hybrid cloud, these schemes can only be used in a trivial way: clients must invoke them repeatedly to check the integrity of data stored in each single cloud. This means that clients must know the exact position of each data block in outsourced data. Moreover, this process will consume higher communication bandwidth and computation costs at client sides. Thus, it is of utmost necessary to construct an efficient verification scheme with collaborative features for hybrid clouds.

In response to practical requirements for outsoucred storages in hybrid clouds, the concerns to improve the performance of PDP services are mainly from three aspects:

- How to design a more efficient PDP model for hybrid clouds to reduce the storage and network overheads and enhance the transparency of verification activities;
- How to provide an efficient sampling policy to help provide a more cost-effective verification service; and
- How to optimize the parameters of PDP scheme to minimize the computation overheads of verification services in hybrid clouds.

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.

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. For example, existing PDP schemes could provide an efficient integrity checking for outsourced data, however, most of these schemes ignore the problem of information leakage among the interactive processes. Thus, as a public verification service without a strong security mechanism for data protection, a malicious attacker could easily exploit such a service to obtain private data. This attack is extremely dangerous to the confidential data of an enterprise.

Even though existing PDP schemes have addressed various aspects such as public verifiability [2], dynamics [4], scalability [3], and privacy preservation [10], we still need a careful consideration to the following attacks, which are more easily compromise the security of storage services in hybrid environments than those in public clouds:

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 the interactive proof process. For instance, Attack 1 and Attack 3 described in Appendix A and B demonstrates that a verifier can get the stored data after running or wiretapping sufficient verification communications. It is obvious that such an attack 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, Attack 2 and Attack 4 given in Appendix A and B shows that a successful forgery attack can occur only if one of the following cases is happened:

- Clients modify data blocks in a file;
- Clients insert and delete blocks repeatedly in a file;
- Clients reuse the same file name to store multiple different files.

Some security mechanisms, such as client-side encryption and access control, can be implemented in clouds to enhance the security of existing PDP schemes, but they will undoubtedly increase the computation and communication overheads of PDP services.

In summary, it is essential to develop an efficient verification method for the data security in hybrid cloud environments. Furthermore, from the above-mentioned challenges, our objectives for checking integrity of outsourced data in hybrid clouds are as follows:

Security aspect: Our scheme should provide adequate security features to resist some existing attacks, such as data leakage attack and tag forgery attack;

Usability aspect: In the way of collaboration, a client should make use of the integrity check via a cloud service provider. Our scheme should conceal the details of the storage to reduce the burden on clients; and

Performance aspect: Our scheme should have a higher performance for anomaly detection and only introduce lower communication and computation overheads.

III. Framework and Main Techniques

In this section, we present our verification framework for hybrid clouds and a formal definition of collaborative PDP. In order to construct such a PDP, we propose three main techniques: fragment structure, hash index hierarchy, and homomorphic verifiable response. These techniques lay the foundation of our CPDP scheme.

A. 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 terms of bandwidth and time. To solve this problem, we consider a hybrid cloud storage service as illustrated in Figure 2. In this architecture, we consider a data storage service involving three different entities: Granted clients, who have a large amount of data to be stored in hybrid clouds and have the permissions to access and manipulate these stored data; Cloud service providers (CSPs), who work together to provide data storage services and have enough storage space and computation resources; and Trusted third parties (TTPs), who are trusted to store the verification parameters and offer the query services for these parameters.

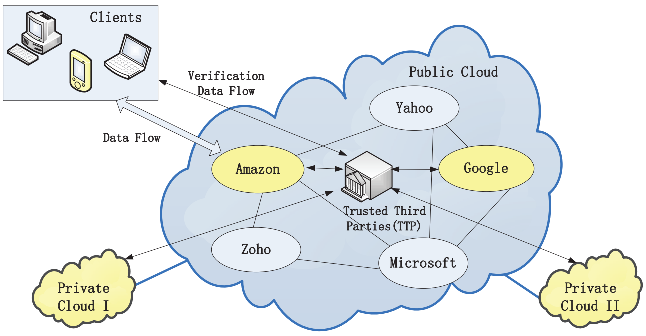

Fig. 2. Verification architecture for data integrity in hybrid clouds.

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 integrated with multiple virtual machine managers, transfer managers, and external cloud providers. In Figure 3, we describe such a cloud computing platform based on OpenNebula architecture [1], in which a service module of collaborative PDP is added into cloud computing management platform (CCMP). This module is able to response the PDP requests of TTP through cloud interfaces. In addition, a hash index hierarchy (HIH), which is described in details in Section III-C, 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 original OpenNebula architecture.

In this architecture, we consider the existence of multiple CSPs to collaboratively store and maintain the 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: Firstly, the client (data owner) uses the secret key to pre-processes the 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; At a later time, by using a verification protocol for collaborative PDP, the clients can issue a challenge for one CSP to check the integrity and availability of outsourcing data in terms of public verification information stored in TTP.

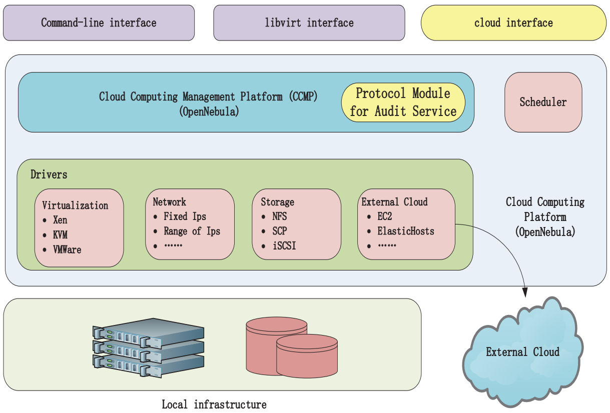

Fig. 3. Cloud computing platform for CPDP service based on OpenNebula architecture [1].

B. 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 interactive proof system (IPS) [11]:

Definition 1 (Collaborative-PDP). *A collaborative 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 κ 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 (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;*

$\mathcal{P}roof(\mathcal{P}, V)$: *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 run, V returns a bit $\{0|1\}$ 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 approach that diminishes the advantages of cloud storage: scaling arbitrarily up and down on-demand [12]. For the sake of clarity, we list some used signals in Table I.

C. Hash Index Hierarchy for Collaborative PDP

As a virtualization approach, we introduce a simple index-hash table to record the changes of file blocks, as well as generate the Hash value of block in the verification process.

TABLE I
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., $F = \{m_{i,j}\}^{i\in[1,n]}_{j\in[1,s]}$;
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.

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 [1].

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;
- 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 this architecture, the resources in Express Layer are split and stored into three CSPs , that have different colors, in Service Layer. In turn, each CSP fragments and stores the assigned data into the storage servers in Storage Layer. We also make use of colors to distinguish different CSPs. 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.

In storage layer, we define a common fragment structure that provides probabilistic verification of data integrity for outsourcing 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 storage layer: an outsourcing file F is split into n blocks $\{m_1, m_2, \cdots, m_n\}$, and each block m_i is split into s sectors $\{m_{i,1}, m_{i,2}, \cdots, m_{i,s}\}$. 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 secrets $\tau = (\tau_1, \tau_2, \cdots, \tau_s)$. 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 $(\mu_1, \mu_2, \cdots, \mu_s, \sigma')$, where μ_i comes from all $\{m_{k,i}, v_k\}_{k\in I}$ and σ' is from all $\{\sigma_k, v_k\}_{k\in I}$.

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, sets $\xi^{(2)}_k = H_{\xi^{(1)}}(\text{“}Cn\text{”})$;
- 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 $\xi^{(3)}_{i,k} = H_{\xi^{(2)}_k}(\chi_i)$ [4], 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.

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 i-th 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 [2], [13]. 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: 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; and 4) the hierarchy structure provides a virtualization manner to conceal the storage details of multiple CSPs.

D. Homomorphic Verifiable Response for Collaborative PDP

A homomorphism is a map $f : \mathbb{P} \rightarrow \mathbb{Q}$ 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) in [2]: 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 $m_i + m_j$.

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

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 2 (homomorphic verifiable response)**.** *A response is called homomorphic verifiable response 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 outsourcing data in hybrid clouds.

E. Collaborative Provable Data Possession

According to the above-mentioned 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 (CSPs) 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 4, as follows: 1) the organizer initiates the protocol and sends a 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 $\mathcal{P}$ 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 which CSPs or in what geographical locations their files reside.

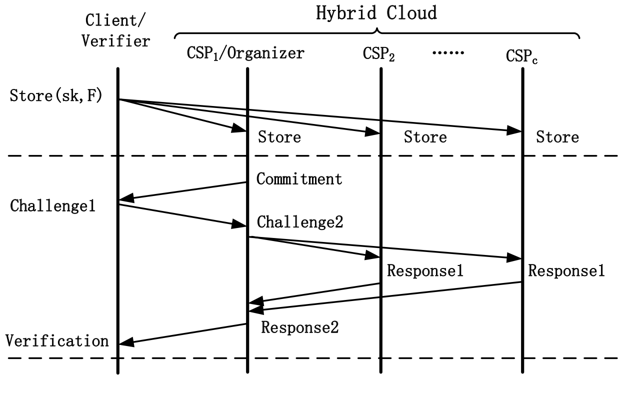

Fig. 4. The flowchart of verification process in our CPDP scheme.

IV. Security and Performance Analysis

A. Security Analysis for CPDP Scheme

The collaborate integrity verification for distrusted outsourcing data, in essence, is a multi-prover interactive proof system (IPS), so that the correspondence construction should satisfy the security requirement of IPS. Moreover, in order to ensure the security of verified data, this kind of construction is also a Multi-Prover Zero-knowledge Proof (MPZKP) system [5], [11], which can be considered as an extension of the notion of an interactive proof system (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 three following properties: (1) **Completeness**: whenever $x \in L$, there exists a strategy for provers that convinces the verifier that this is the case; (2) **Soundness**: whenever $x \notin L$, whatever strategy the provers employ, they will not convince the verifier that $x \in L$; (3) **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 doesn't exists an interactive machine P^* 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 Section II, 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. Firstly, randomness is adopted into the CSP's response in order to resist Attack 2 and 4 in Appendix A and B, that is, 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.

B. 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

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

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.[5] 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 k-th CSP, ρ_k denotes the probability of file corruption in the k-th CSP, and $r_k \cdot \omega$ denotes the possible number of blocks queried by the verifier in the k-th 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 sever misbehavior with a certain probability P by asking proof for a constant amount of blocks $t = \frac{\log(1-P)}{\log(1-\rho_b)} = \frac{\log(1-P)}{s\log(1-\rho)}$ 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 [2].

C. CPDP for Integrity Audit Services

In actual practice, we introduce the collaborative PDP scheme to construct an audit system architecture for outsourcing data in hybrid clouds by replacing TTP with a third party auditor (TPA) in Figure 2. In this architecture, data owner 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 the stored data, nor assume that data owner has 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 their outsourcing 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 outsourcing 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 on-line 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;

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

- Privacy-preserving: to ensure that there exists no way for TPA to derive users' data from the information collected during the auditing process;
- 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/sec 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 5,200 lines of codes and has been 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 [1]. In Figure 5, we show an example of Hadoop distributed file system (HDFS) [6], which a distributed, scalable, and portable file system [14]. 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.

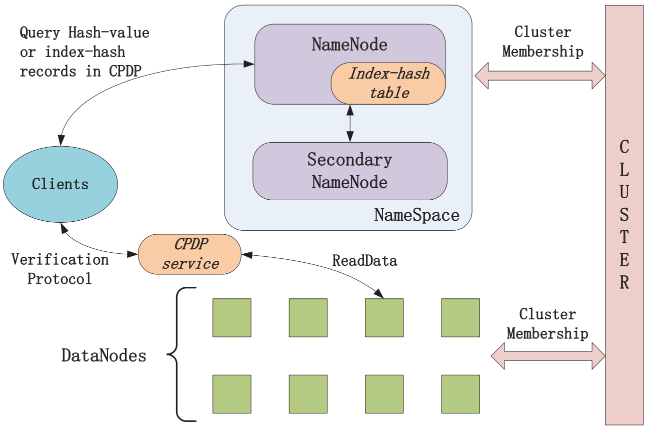

Fig. 5. An example of hash index hierarchy in Hadoop distributed file system (HDFS).

V. Related Work

Traditional cryptographic technologies for data integrity and availability, based on hash functions and signature schemes [8], [9], 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) [2] and Proofs of Retrievability (POR) [15]. Ateniese et al. [2] 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 [3]. 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. [4] 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 [15] 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 [13] 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. [10] presented a dynamic scheme with $O(\log n)$ cost by integrating the above CPOR scheme and Merkle Hash Tree (MHT) in DPDP. Furthermore, serveral POR schemes and models have been proposed recently including [16], [17]. Since the response of challenges has homomorphic property, the above schemes (especially CPOR schemes) can leverage

[6] 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.

the PDP construction in hybrid clouds.

VI. Conclusions

In this paper, we addressed the construction of collaborative integrity verification mechanism for distributed data outsourcing in hybrid clouds. Based on homomorphic verifiable responses and hash index hierarchy, we proposed a collaborative provable data possession scheme to support dynamic scalability on multiple storage servers. Our performance analysis indicated that our proposed solution only incurs a small constant amount of communications overhead.

VII. Acknowledgments

The work of Y. Zhu and S. Chen was supported by the National Natural Science Foundation of China (Project No. 61170264) and the NDRC under Project “A cloud-based service for monitoring security threats in mobile Internet”. 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) and Department of Energy (DE-SC0004308).

References

[1] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. T. Foster, “Virtual infrastructure management in private and hybrid clouds,” *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[2] 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*, 2007, pp. 598–609.

[3] 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*, 2008, pp. 1–10.

[4] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, “Dynamic provable data possession,” in *ACM Conference on Computer and Communications Security*, 2009, pp. 213–222.

[5] L. Fortnow, J. Rompel, and M. Sipser, “On the power of multi-prover interactive protocols,” in *Theoretical Computer Science*, 1988, pp. 156–161.

[6] 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, pp. 1–10.

[7] 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.

[8] 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.

[9] A. R. Yumerefendi and J. S. Chase, “Strong accountability for network storage,” *TOS*, vol. 3, no. 3, 2007.

[10] 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.

[11] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, and H. Hu, “Zero-knowledge proofs of retrievability,” *Science China: Information Sciences (Series F)*, vol. 54, no. 8, pp. 1608–1617, 2011.

[12] 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, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[13] H. Shacham and B. Waters, “Compact proofs of retrievability,” in *ASIACRYPT*, 2008, pp. 90–107.

[14] 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. [Online]. Available: http://lucene.apache.org/hadoop/

[15] A. Juels and B. S. K. Jr., “Pors: proofs of retrievability for large files,” in *ACM Conference on Computer and Communications Security*, 2007, pp. 584–597.

[16] 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*, 2009, pp. 187–198.

[17] Y. Dodis, S. P. Vadhan, and D. Wichs, “Proofs of retrievability via hardness amplification,” in *TCC*, 2009, pp. 109–127.

Appendix

A. Attacks for 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 $e : G \times G \rightarrow G_T$ be a computable bilinear map with group G’s support being $\mathbb{Z}_p$ and $H : \{0,1\}^* \rightarrow G$ 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

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

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 extend the challenge coefficients, that is, $v_i = 0$ for any $(i, v_i) \notin Q$. Such that the adversary gets the responses $\{(\sigma^{(1)}, \mu^{(1)}), \cdots, (\sigma^{(1)}, \mu^{(n)})\}$ after he finishes n times queries. These responses can generate the equations

$$\left\{ \begin{array}{lcl} \mu^{(1)} & = & v_1^{(1)} m_1 + \cdots + v_n^{(1)} m_n \\ \vdots & & \vdots \\ \mu^{(n)} & = & v_1^{(n)} m_1 + \cdots + v_n^{(n)} m_n \end{array} \right.$$

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 $\sigma^{(i)} = {\sigma_1}^{v_1^{(i)}} \cdot {\sigma_2}^{v_2^{(i)}} \cdots {\sigma_n}^{v_n^{(i)}}$ 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, 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. ■

B. Attacks for 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

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

This scheme is not secure due to the leakage of outsourcing data 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 $\sigma^*_k = H(Fn,k)^x \cdot \prod_{j=1}^{s} (u_j^x)^{m^*_{k,j}} = \sigma_k \cdot \prod_{j=1}^{s} (u_j^x)^{m^*_{k,j} - m_{k,j}}$. ■