Secure and Efficient Proof of Storage with Deduplication

Qingji Zheng
Department of Computer Science
University of Texas at San Antonio
qzheng@cs.utsa.edu

Shouhuai Xu
Department of Computer Science
University of Texas at San Antonio
shxu@cs.utsa.edu

ABSTRACT

Both security and efficiency are crucial to the success of cloud storage. So far, security and efficiency of cloud storage have been separately investigated as follows: On one hand, security notions such as Proof of Data Possession (PDP) and Proof of Retrievability (POR) have been introduced for detecting the tamperation of data stored in the cloud. One the other hand, the notion of Proof of Ownership (POW) has also been proposed to alleviate the cloud server from storing multiple copies of the same data, which could substantially reduce the consumption of both network bandwidth and server storage space. These two aspects are seemingly quite to the opposite of each other. In this paper, we show, somewhat surprisingly, that the two aspects can actually coexist within the same framework. This is possible fundamentally because of the following insight: *The public verifiability offered by* PDP/POR *schemes can be naturally exploited to achieve* POW. This "one stone, two birds" phenomenon not only inspired us to propose the novel notion of Proof of Storage with Deduplication (POSD), but also guided us to design a concrete scheme that is provably secure in the Random Oracle model based on the Computational Diffie-Hellman (CDH) assumption.

Categories and Subject Descriptors

C.2.4 [**Communication Networks**]: Distributed Systems; H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Security

Keywords

cloud storage, outsourced storage, proof of storage, deduplication, integrity checking, proof of ownership, proof of data possession, proof of retrievability

1. INTRODUCTION

Cloud computing is getting increasingly popular because it can provide low-cost and on-demand use of vast storage and processing resources. The present paper focuses on the security and efficiency of cloud storage, namely that clients outsource their data to cloud storage servers. While cloud storage offers compelling scalability and availability advantages over the current paradigm of "one storing and maintaining its own IT systems and data", it does not come without security concerns. This has led to studies on cloud storage security and efficiency, which are, however, addressed separately as we discuss below.

From the perspective of cloud storage security, there have been two notable notions:

- Proof of Data Possession (PDP): This notion was introduced by Ateniese et al. [2]. It allows a cloud client to verify the integrity of its data outsourced to the cloud in a very efficient way (i.e., far more efficient than the straightforward solution of downloading the data to the client-end for verification). This notion has been enhanced in various ways [8, 3, 15].
- Proof of Retrievability (POR): This notion was introduced by Juels and Kaliski [10]. Compared with PDP, POR offers an extra property that the client can actually "recover" the data outsourced to the cloud (in the flavor of "knowledge extraction" in zero-knowledge proof). This notion has been enhanced and extended in multiple aspects [12, 6, 5, 16].

From the perspective of cloud storage efficiency, deduplication technique has become a common practice of many cloud vendors. This is reasonable especially when there are many duplications in the data outsourced to the cloud (e.g., only 25% of data may be unique according to a survey [1]). As such, the cloud vendor can substantially save storage space by storing a single copy of each data — no matter how many clients outsourced it, which explains the term "deduplication". This issue was first introduced to the research community by [9]. Because straightforward deduplication is vulnerable to attacks (e.g., a dishonest client can claim that it has certain data while it does not), Halevi et al. [13] proposed the notion called Proof of Ownership (POW) as well as concrete constructions.

Our contributions. Both the security and efficiency perspectives mentioned above are important and would be needed by a single cloud storage solution, which is a new problem that has not been addressed. In this paper, we

tackle this problem by proposing a "2-in-1" notion we call Proof of Data Storage with Deduplication (POSD). Specifically, we introduce the novel concept of POSD, and formalize its functional and security definitions. Moreover, we propose the first efficient POSD scheme and prove its security in the Random Oracle model based on the Computational Diffie-Hellman (CDH) assumption. We also analyze and compare the performance of our scheme and the performance of some relevant PDP/POR/POW schemes, which suggests that our POSD scheme is as efficient as the PDP/POR/POW schemes.

Organization. The rest of the paper is organized as follows. Section 2 briefly reviews the related prior work. Section 3 discusses the notations and cryptographic settings. Section 4 presents the definitions of POSD. Section 5 describes our POSD scheme and its security as well as performance analysis. Section 6 concludes the paper.

2. RELATED WORK

Cloud storage security was not systematically studied until very recently, despite previous investigations for similar problems (cf. [2]). Ateniese et al. [2] introduced the concept of PDP, and Juels et. al [10] proposed the concept of POR, which was improved significantly by Shacham and Waters [12]. The main difference between the two notions is that POR uses Error Correction/Erasure Codes to tolerate the damage to portions of the outsourced data. These solutions are later enhanced in various ways [8, 4, 5, 6, 16].

Data deduplication of ciphertext data in the pre-cloud era was studied in [14, 7]. Data deduplication in the context of cloud computing was recently introduced [9]. Halevi et al. [13] dubbed the term of POW and presented the first systematic study of deduplication in cloud, including several alternative solutions that offer different trade-offs between security and performance.

3. PRELIMINARIES

3.1 Notations

Let ℓ be a security parameter. A function $\varepsilon(\ell)$ is negligible if it is smaller than ℓ^{-const} for any constant *const* and sufficiently large ℓ.

Let q a ℓ-bit prime and p a prime such that $q|(p-1)$. Let F be a data file consisting of n blocks, where the i^{th} block F_i is composed of m symbols in $\mathbb{Z}_q$, i.e. $\mathsf{F}_i = (\mathsf{F}_{i1} \cdots, \mathsf{F}_{im})$, where $\mathsf{F}_i \in \mathbb{Z}_q^m$.

Let fid be the identity that uniquely identifies data file F. Let each file be associated with some auxiliary information (i.e. cryptographic tags), denoted by $\mathtt{Tag}$. We consider two variants of $\mathtt{Tag}$: $\mathtt{Tag}_{int}$ is the cryptographic information for auditing data integrity, and $\mathtt{Tag}_{dup}$ is the cryptographic information for duplication checking.

Let [] denote the optional arguments of a function or algorithm; for example, $\mathsf{Alg}(a, b[, c])$ means that algorithm Alg has two arguments a and b, and optionally a third argument c.

3.2 Cryptographic Setting and Assumptions

Let ℓ be a security parameter. Let G and G_T be cyclic groups of prime order q and g be a generator of G. Let $e : G \times G \rightarrow G_T$ be a bilinear map, with the following properties: (i) e can be computed efficiently; (ii) for all $(u, v) \in G \times G$, and $a, b \in \mathbb{Z}_q$, $e(u^a, v^b) = e(u, v)^{ab}$; (iii) $e(g, g) \neq 1$.

The standard Computational Diffie-Hellman (CDH) Problem is the following: Given $(g, g^w, h) \in G^3$, where g, g^w, h are selected uniformly at random from G, compute h^w. The CDH Assumption says that no probabilistic polynomial-time (PPT) algorithm can solve the CDH Problem with a non-negligible probability (in ℓ).

The Discrete Log (DLOG) Problem is the following: Given any prime q-order cyclic group G and two random elements g and h, find w such that $\mathsf{g}^w = \mathsf{h}$. The DLOG Assumption says that no PPT algorithm can solve the DLOG Problem only with a non-negligible probability (in ℓ). The DLOG Assumption is weaker than the CDH Assumption.

Let $H_1 : \{0,1\}^* \rightarrow G$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q$ be randomly chosen from the respective families of hash functions. Both H_1 and H_2 are modeled as random oracles.

Let $\mathsf{PRF} : \{0,1\}^\ell \times \{0,1,\}^* \rightarrow \{0,1\}^\ell$ be a family of secure pseudorandom functions.

4. REQUIREMENTS, MODEL AND DEFINITIONS OF POSD

Requirements. Built on top of [2, 10, 13], we summarize the performance requirements of POSD as:

- A solution should use common functions (e.g., hash functions) so as to allow cross-client data deduplication and cross-client cloud data integrity auditing.
- A solution should consume bandwidth that is substantially less than the size of the data file in question. This prevents the aforementioned trivial solutions.
- A solution should not force the cloud server, when determining whether to conduct a deduplication operation, to retrieve any significant portion of the data file in question. This is plausible because it could be very resource-consuming to load a large data file from secondary storage to memory.
- A solution should only require the client to make a single pass over its data file, while using an amount of memory that is substantially smaller than the size of the data file in question.

As in the cases of PDP/POR and POW, there are trivial solutions to fulfill the functions of POSD. Specifically, a client can download the whole data from the cloud to verify the integrity of its data outsourced to the cloud, and the server can ask the client to upload a data file to show that the client indeed has a copy of the data file before conducting the deduplication operation. However, this trivial solution is not practical because it incurs prohibitive communication overhead. On the other hand, it was also noted in [2, 10, 13] that simple heuristics will not solve the respective problems without shortcomings.

Model participants. We consider a cloud storage service model that involves the following three participants.

(i) Cloud storage server, denoted by S: It provides storage service with relevant assurance procedures, by which the cloud storage clients can check the integrity of their data stored in the cloud and the server can save storage space via data deduplication in a secure fashion.

(ii) Cloud storage clients, denoted by C: A client outsources its data to the cloud in a secure fashion, while allowing the cloud storage server to conduct data deduplication operations. (If a client does not want the server to conduct this operation, this can be achieved via an appropriate contract-level agreement that is out of the scope of the present paper.)

(iii) Third party, denoted by AUDITOR: A client may allow a third party to check the integrity of its data outsourced to the cloud. Moreover, any client, who possesses a data file that is duplicated (i.e., the same data file has been uploaded to the server by another client), can act as a AUDITOR of that specific data file.

Communication channels. If the data file outsourced to the cloud is not confidential, there is no need for private channels. (In this case, secure deduplication still can be relevant because it may be very expensive to eavesdrop the communication during the transfer of a large data file.) In the case the outsourced data files are confidential, we can assume the availability of private communication channels for the execution of certain protocols. This is common to PDP/POR/POW [2, 12, 13] and avoids unnecessary complications in describing the protocols (given that private channels can be implemented using standard techniques in a modular fashion). Note that in order to facilitate deduplication, the data will be stored in plaintext in the cloud, which is the same as in POW [13].

Functional definition. The following definition of POSD is built on the definitions of PDP/POR [2, 12] and POW [13].

Definition 1. (functional definition) A POSD scheme, denoted by Λ, consists of the following tuple of polynomial-time algorithms (KEYGEN,UPLOAD,AUDITINT,DEDUP).

KEYGEN**:** This is the *key generation* algorithm. It takes as input a security parameter ℓ, and outputs two pairs of public/private keys $(\mathrm{PK}_{int}, \mathrm{SK}_{int})$ and $(\mathrm{PK}_{dup}, \mathrm{SK}_{dup})$, where PK_{int} is made public and SK_{int} is the corresponding private key of a client (this pair of keys may be used for integrity protection/verification purpose), PK_{dup} is made public and SK_{dup} is the private key of the server (this pair of keys may be used for secure deduplication purpose).

UPLOAD**:** This is the *data uploading* protocol running by a client C and a server S over a private channel so that secrecy of the data is assured. Suppose C wants to upload a new data file F to the cloud, where S can easily determine that F has not be outsourced to the cloud by any client (e.g., by comparing the hash value provided by C against the list of hash values stored by the server). For preprocessing, Client C takes as input a new data file F with a unique identifier fid and the secret key SK_{int}, outputs some auxiliary information $\mathtt{Tag}_{int}$ that can be used to audit the integrity of F in the cloud. At the end of the execution, S stores $(\mathsf{fid}, \mathsf{F}, \mathtt{Tag}_{int})$ received from C as well as possibly some deduplication information $\mathtt{Tag}_{dup}$, which may be produced by the server using SK_{dup}. The server may also keep a hash value of the F's so as to facilitate the detection of data duplications and thus the need of deduplication.

AUDITINT**:** This is the *data integrity auditing* protocol. It is executed between server S and AUDITOR so that S convinces AUDITOR that integrity of some data file stored in the cloud is assured. The AUDITOR's input includes the data file identifier fid and the corresponding client's PK_{int}. The server's input includes the data file F corresponding to fid and the auxiliary information $\mathtt{Tag}_{int}$ associated to F. Essentially, the protocol is of challenge-response type, where AUDITOR sends a challenge chal to the server and the server computes and sends back a response resp. If resp is valid with respect to chal as well as the other relevant information, AUDITOR outputs 1, meaning that the integrity of F is assured, and 0 otherwise. Formally, we can write it as:

$$b \leftarrow (\textsc{Auditor}(\mathsf{fid}, \mathrm{PK}_{int}) \Longleftrightarrow \mathrm{S}(\mathsf{fid}, \mathsf{F}, \mathtt{Tag}_{int}))$$

where $b \in \{0, 1\}$.

DEDUP**:** This is the *deduplication checking* protocol. It is executed between server S and client C, who claims to possess a data file F (the detection of the need to deduplicate can be fulfilled by C sending the hash value of its data file to S, which can determine whether or not the data file has been in the cloud). This protocol is also essentially of challenge-response type. Basically, S sends a challenge chal to C, which returns a response resp that is produced using data file F and possibly other information. S verifies the validity of resp using possibly $\mathtt{Tag}_{dup}$ and PK_{dup}, and outputs 1 if the verification is successful (meaning that the client indeed has data file F) and 0 otherwise. Formally, we can write it as:

$$b \leftarrow (\mathrm{S}(\mathsf{fid}, \mathtt{Tag}_{dup}, [\mathrm{SK}_{dup},]\mathrm{PK}_{dup}) \Longleftrightarrow \mathrm{C}(\mathsf{fid}, \mathsf{F}))$$

where $b \in \{0, 1\}$.

Correctness definition. We require a POSD scheme $\Lambda =$ (KEYGEN, UPLOAD, AUDITINT, DEDUP) to be *correct* if, for honest client and server, the execution of the AUDITINT protocol will always output 1 and the execution of the DEDUP protocol will always output 1.

Security definition. We define security of POSD using games, which specify both the adversary's behavior (i.e., what the adversary is allowed to do) and the winning condition (i.e., under what circumstance we say the attack is successful). At a high-level, we require a POSD scheme to be *server_unforgeable*, which is similar to the security defined by the data possession game in [2], and (κ, θ)*_uncheatable*, which is similar to the security definition in [13].

Intuitively, we say a POSD scheme is *server_unforgeable* if no cheating server can successfully execute the AUDITINT protocol with an honest AUDITOR with a non-negligible probability. Formally, we have:

Definition 2. (server_unforgeability) For POSD scheme $\Lambda =$ (KEYGEN, UPLOAD, AUDITINT, DEDUP), consider the following game between an adversary $\mathcal{A}$ and a challenger, where $\mathcal{A}$ plays the role of the cloud server S while possibly controlling many compromised clients, and the challenger acts as an honest client.

Setup Stage:

- Run algorithm KEYGEN to generate $(\mathrm{PK}_{int}, \mathrm{SK}_{int})$ and $(\mathrm{PK}_{dup}, \mathrm{SK}_{dup})$. Make PK_{int} and PK_{dup} public, including giving SK_{int} to the respective client and SK_{dup} to $\mathcal{A}$. Note that the SK_{int} of the challenger is not given to $\mathcal{A}$. For any other client, the corresponding SK_{int} may be given to $\mathcal{A}$ as long as it requests (i.e., these clients are compromised by, or collude with, $\mathcal{A}$).

Challenge Stage: At this stage, $\mathcal{A}$ can do anything with respect to the clients other than the challenger. With respect to the challenger, $\mathcal{A}$ does the following.

- $\mathcal{A}$ adaptively chooses a data file $\mathsf{F} \in \{0,1\}^*$ for the challenger. The challenger picks a unique identifier fid and runs the UPLOAD protocol with $\mathcal{A}$. At the end of the execution, $\mathcal{A}$ obtains $(\mathsf{fid}, \mathsf{F}, \mathtt{Tag}_{int})$. The above process may be repeated for polynomial many times. Denote by $\mathcal{Q} = \{(\mathsf{fid}, \mathsf{F}, \mathtt{Tag}_{int})\}$, the set of tuples $\mathcal{A}$ received from the challenger when executing the UPLOAD protocol. Note that the challenger keeps a record of $\mathcal{Q}_{\mathsf{fid}} = \{\mathsf{fid}\}$, namely the projection of $\mathcal{Q}$ on attribute fid.

- $\mathcal{A}$ can execute AUDITINT with the challenger with respect to any $\mathsf{fid} \in \mathcal{Q}_{\mathsf{fid}}$, and execute DEDUP with the challenger with respect to some data file (possibly chosen by $\mathcal{A}$). This process can be executed polynomial (in ℓ) number of times.

Forgery Stage:

- The adversary outputs an $\mathsf{fid} \in \mathcal{Q}_{\mathsf{fid}}$ corresponding to F that was outsourced to the cloud.

The adversary wins the game if for any $\mathsf{F}' \neq \mathsf{F}$,

$$1 \leftarrow (\textsc{Auditor}(\mathsf{fid}, \mathrm{PK}_{int}) \Longleftrightarrow \mathcal{A}(\mathsf{fid}, \mathsf{F}', \cdot)),$$

We say Λ is *server_unforgeable* if the winning probability for any PPT algorithm $\mathcal{A}$ is negligible in ℓ.

Intuitively, we say a POSD scheme is (κ, θ)*_uncheatable* if given a file F with min-entropy κ, no cheating client, who can find F' containing θ-bit Shannon entropy of F, convinces the server that it has F with a probability non-negligibly more than $2^{-(\kappa-\theta)}$. Formally, we have:

Definition 3. ((κ, θ)_uncheatability) For a POSD scheme $\Lambda = (\textsc{Keygen}, \textsc{Upload}, \textsc{AuditInt}, \textsc{Dedup})$, consider the following game between the adversary $\mathcal{A}$ (who plays the role of the compromised clients) and a challenger (who plays the role of the server and an honest client).

Setup Stage:

- Run algorithm KEYGEN to generate $(\mathrm{PK}_{int}, \mathrm{SK}_{int})$ as well as $(\mathrm{PK}_{dup}, \mathrm{SK}_{dup})$. Make PK_{int} and PK_{dup} public, including giving PK_{int} and PK_{dup} to the adversary $\mathcal{A}$. For a compromised client, the corresponding SK_{int} is given to $\mathcal{A}$. However, both SK_{dup} and the SK_{int} corresponding to the honest client are only given to the challenger. (Note that if the SK_{int} corresponding to the honest client is given to $\mathcal{A}$, $\mathcal{A}$ could use it to authenticate to the server to download the client's any data outsourced to the server.)

- The challenger chooses a data file F of κ-bit min-entropy, and a unique identifier fid. The challenger honestly executes the UPLOAD protocol by playing the roles of both the client and the server, and gives the publicly observable information to the adversary $\mathcal{A}$.

Challenge Stage: At this stage, $\mathcal{A}$ seeks to infer the content of F by running the UPLOAD, AUDITINT and DEDUP protocols with the challenger. In particular, $\mathcal{A}$ may penetrate into the cloud server to learn some portions of F. This is reasonable because stealing the whole F, or any form of its compressed version (because F has enough min-entropy and thus Shannon entropy), could alert the defender about the compromise because of the abnormal use of network bandwidth and/or CPU resources. Moreover, subliminal channels normally do not offer bandwidth compatible to the magnitude of κ. Note that the above adversarial model also accommodates that $\mathcal{A}$ can command the compromised clients to launch some type of guessing attacks with respect to F, which is possible for example when F has a public structure (e.g., Word document or movie file). In any case, suppose $\mathcal{A}$ learned up to θ-bit Shannon entropy of F.

Forgery Stage: $\mathcal{A}$ eventually outputs some F'. $\mathcal{A}$ wins the game if

$$1 \leftarrow (\mathrm{S}(\mathsf{fid}, \mathtt{Tag}_{dup}, [\mathrm{SK}_{dup},]\mathrm{PK}_{dup}) \Longleftrightarrow \mathrm{C}(\mathsf{fid}, \mathsf{F}')).$$

We say Λ is (κ, θ)*_uncheatable* if the winning probability for any PPT algorithm $\mathcal{A}$ is negligibly (in ℓ) more than $2^{-(\kappa-\theta)}$. Note that $\kappa - \theta \geq \ell$ would be the most often cases because we mainly deal with large data files, and thus $\mathcal{A}$'s winning probability is effectively required to be negligible in ℓ.

Discussion. In the above security definitions, we did not consider the notion of *fairness*, which was very recently introduced to prevent a dishonest client from legitimately accusing an honest cloud server of tampering its data in the setting of *dynamic* POR [16]. This is because POSD in this paper deals with *static* data, rather than dynamic data where fairness can be reasonably involved [16]. For static data, fairness can be easily achieved by letting a client sign sign F in the UPLOAD protocol or after a successful execution of the DEDUP protocol.

5. POSD CONSTRUCTION AND ANALYSIS

5.1 Basic Ideas

As discussed above, it is conceptually convenient to think "POSD=PDP/POR+POW" because POSD aims to fulfill the functionalities of both integrity audit and deduplication. In this paper, we focus on the scenario of "POSD=PDP+POW" because of the following. First, POR is more costly that PDP due to its use of Error Correcting/Erasure Codes for fulfilling retrievability. Second, in the DEDUP protocol of POSD (and POW), retrievability is actually not needed because the server already knows F. Nevertheless, the basic ideas are equally applicable to the scenario of "POSD=POR+POW." In what follows we first elaborate the insight that led us to our design.

Relationship between PDP/POR and POW, revisited. From the definition of POSD, we see some similarity between

the AuditInt protocol and the Dedup protocol. Specifically, both protocols are in a sense for verifying integrity except that one is for data in the cloud-end (i.e., cloud server attests to client) and the other is for data in the client-end (i.e., client attests to cloud server). Because the AuditInt protocol is the core of PDP/POR, there is some similarity between PDP/POR and POW (as noted in [13]) as well as POSD. However, it is stated in [13] that PDP/POR protocols are not applicable in the setting of POW (and thus POSD) — we call this the "Deduplication Gap" between PDP/POR and POW/POSD— because of the following:

> In PDP/POR, there is a preprocessing step that facilitates that the client can later verify the integrity of its data in the cloud. Whereas, in the setting of POW, a new client possesses a secret data file F, *but no other secrets.*

Halevi et al. [13] correctly excluded the possibility of using PDP/POR based on symmetric key cryptosystems [2, 12] for the purpose of POW (because the new client does not, and should not, know the secret keys). We observe, however, that Publicly Verifiable PDP/POR protocols are actually sufficient for the purpose of POW. As we will see, this is made possible because the client can compute the needed information from F on the fly and without using any secret information.

Why is the public verifiability sufficient to bridge the above "Deduplication Gap"? Conceptually, public verifiability of PDP/POR meaning that a third party, who may be given some *non-secret* information by a client, can verify the integrity of the client's data in the cloud. Putting this into the setting of POW/POSD, we can let the Dedup protocol be essentially the same as the core PDP/POR protocol by reversing the roles of the client and the server. More specifically, if the AuditInt protocol is publicly verifiable, it is possible that only the public key PK_{int} is needed for the cloud server and only the data file F (as well as the public parameters, of course) is needed for the client.

5.2 Construction

Recall that q is an ℓ-bit prime and p is another prime such that $q|(p-1)$. G and G_T are cyclic groups of prime order q, and $e : G \times G \to G_T$ is a bilinear map. We use two hash functions (random oracles) $H_1 : \{0,1\}^* \to G$ and $H_2 : \{0,1\}^* \to \mathbb{Z}_q$.

F is a data file consisting of n blocks of m symbols in $\mathbb{Z}_q$, namely $\mathsf{F}_i = (\mathsf{F}_{i1} \cdots, \mathsf{F}_{im})$, where $\mathsf{F}_i \in \mathbb{Z}_q^m$ for $1 \le i \le n$. F is uniquely identified by fid.

The POSD scheme is described as follows (in the end of this subsection we will explain some design decisions to help understand our scheme):

Keygen: This algorithm generates cryptographic keys as follows:

- Select v_1 and v_2 uniformly at random from $\mathbb{Z}_p^*$ such that the orders of v_1 and v_2 are q (if v_2 is generated from v_1, then the DLOG of v_2 to base v_1 should be erased afterwards). Select s_{j1}, s_{j2} uniformly at random from $\mathbb{Z}_q^*$ for $1 \le j \le m$. Set $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}} \mod p$ for $1 \le j \le m$.
- Let g be a generator of G. Select u uniformly at random from G. Select w uniformly at random from $\mathbb{Z}_q^*$, and set $z_g = g^w$.
- Set $\mathrm{PK}_{int} = \{q, p, g, u, v_1, v_2, z_1, \cdots, z_m, z_g\}$ and the client's private key $\mathrm{SK}_{int} = \{(s_{11}, s_{12}), \cdots, (s_{m1}, s_{m2}), w\}$. Note that using an appropriate Pseudorandom Function (PRF), we can further reduce the storage at the client-end to constant (i.e., using a single key to the PRF for generating the $s_{11}, s_{12}, \cdots, s_{m1}, s_{m2}, w$).
- Set $\mathrm{PK}_{dup} = \mathrm{PK}_{int}$ and $\mathrm{SK}_{dup} = \mathsf{null}$, where PK_{dup} is also made public.

Upload: This protocol is performed between a client, who is to outsource a data file F to the cloud, and the cloud server as follows:

- For each data block F_i, where $1 \le i \le n$, the client selects r_{i1}, r_{i2} uniformly at random from $\mathbb{Z}_q^*$ and computes:

$$\begin{aligned} x_i &= v_1^{r_{i1}} v_2^{r_{i2}} \mod p, \\ y_{i1} &= r_{i1} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j1} \mod q, \\ y_{i2} &= r_{i2} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j2} \mod q, \\ t_i &= \left(H_1(\mathsf{fid}||i) \cdot u^{H_2(x_i)}\right)^w \quad (\text{in } G). \end{aligned}$$

  The client sends $(\mathsf{fid}, \mathsf{F}, \mathsf{Tag}_{int})$ to the server, where $\mathsf{Tag}_{int} = \{(x_i, y_{i1}, y_{i2}, t_i)_{1 \le i \le n}\}$.
- Upon receiving $(\mathsf{fid}, \mathsf{F}, \mathsf{Tag}_{int})$, the server sets $\mathsf{Tag}_{dup} = \mathsf{Tag}_{int}$.

AuditInt: This protocol is executed between an auditor, which can be the client itself, and the cloud server to verify the integrity of the client's data file F stored in the cloud. Note that the client does not need to give any information to the auditor except the public keys PK_{int} and the data file identifier fid.

- The auditor chooses a set of c elements $\mathrm{I} = \{\alpha_1, \ldots, \alpha_c\}$ where α_i is selected uniformly at random from $\{1, \ldots, n\}$, and chooses a set of coefficients $\beta = \{\beta_1, \ldots, \beta_c\}$ where β_i is selected uniformly at random from $\mathbb{Z}_q^*$. The auditor sends $\mathsf{chal} = (\mathrm{I}, \beta)$ to the server.
- The server computes:

$$\mu_j = \sum_{i \in \mathrm{I}} \beta_i \mathsf{F}_{ij} \mod q$$

  for $1 \le j \le m$, and

$$\begin{aligned} \mathrm{Y}_1 &= \sum_{i \in \mathrm{I}} \beta_i y_{i1} \mod q, \\ \mathrm{Y}_2 &= \sum_{i \in \mathrm{I}} \beta_i y_{i2} \mod q, \\ \mathrm{T} &= \prod_{i \in \mathrm{I}} t_i^{\beta_i} \quad (\text{in } G). \end{aligned}$$

  The server sends $\mathsf{resp} = (\{\mu_j\}_{1 \le j \le m}, \{x_i\}_{i \in \mathrm{I}}, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{T})$ to the server, where $x_i = v_1^{r_{i1}} v_2^{r_{i2}} \mod p$ for $1 \le i \le n$ were generated by the client in the execution of the Upload protocol.

- Upon receiving resp, the auditor parses resp as $\{\{\mu_j\}_{1\le j\le m}, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{T}, \{x_i\}_{i\in \mathrm{I}}\}$, computes

$$\mathrm{X} = \prod_{i\in \mathrm{I}} x_i^{\beta_i} \mod p,$$
$$\mathrm{W} = \prod_{i\in I} H_1(\mathsf{fid}||i)^{\beta_i},$$

and verifies

$$\mathrm{X} \stackrel{?}{=} v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \mod p$$
$$e(\mathrm{T}, g) \stackrel{?}{=} e(\mathrm{W}u^{\sum_{i\in \mathrm{I}} \beta_i H_2(x_i)}, z_g) \quad (\text{in } G_T).$$

If both hold, return 1; otherwise, return 0.

DEDUP: This protocol is executed between the client, who claims to have a data file F with identifier fid that was already outsourced to the cloud by another client, and the server. This is a simple variant of the above AUDITINT protocol, where the auditor only needs to know the public keys PK_{int} and the data file identifier fid. Here, we let the cloud server play the role of the auditor (with some minor adaptations because there are some information that is not known to the client in producing the response), who naturally knows PK_{int} and fid.

- The server chooses a set of c elements $\mathrm{I} = \{\alpha_1, \ldots, \alpha_c\}$ where α_i is selected uniformly at random from $\{1, \ldots, n\}$, and chooses a set of coefficients $\beta = \{\beta_1, \ldots, \beta_c\}$ where β_i is selected uniformly at random from $\mathbb{Z}_q^*$. The server sends $\mathsf{chal} = (\mathrm{I}, \beta)$ to the client.
- The client computes

$$\mu_j = \sum_{i\in \mathrm{I}} \beta_i \mathsf{F}_{ij} \mod q$$

for $1 \le j \le m$, and sends $\mathsf{resp} = (\{\mu_i\}_{1\le i\le m})$ to the server.

- The server computes from $\mathsf{Tag}_{dup} = \{(x_i, y_{i1}, y_{i2}, t_i)_{1\le i\le n}\}$:

$$\mathrm{Y}_1 = \sum_{i\in \mathrm{I}} \beta_i y_{i1} \mod q,$$
$$\mathrm{Y}_2 = \sum_{i\in \mathrm{I}} \beta_i y_{i2} \mod q,$$
$$\mathrm{W} = \prod_{i\in I} H_1(\mathsf{fid}||i)^{\beta_i},$$
$$\mathrm{X} = \prod_{i\in \mathrm{I}} x_i^{\beta_i} \mod p,$$
$$\mathrm{T} = \prod_{i\in \mathrm{I}} t_i^{\beta_i} \quad (\text{in } G).$$

The server verifies

$$\mathrm{X} \stackrel{?}{=} v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \mod p,$$
$$e(\mathrm{T}, g) \stackrel{?}{=} e\left(\mathrm{W}u^{\sum_{i\in \mathrm{I}} \beta_i H_2(x_i)}, z_g\right) \quad (\text{in } G_T).$$

If both hold, return 1; otherwise, return 0.

Discussion on some design decisions. To help understand our scheme, now we discuss some design decisions we made to satisfy both the above performance design requirements and the security definitions. First, our UPLOAD and AUDITINT protocols are new. When compared with existing protocols for the similar purpose [2, 12, 8, 5, 6, 16], it has the following significant advantage during the execution of the UPLOAD protocol (a thorough comparison will be present in Section 5.5). Our scheme only requires the client to perform $O(n)$ exponentiation operations plus $O(mn)$ multiplication operations, where n is the number of data blocks and m is number of symbols in each block (i.e., mn is the number of symbols in a data file). In contrast, the referred schemes require the client to perform $O(mn)$ exponentiation operations plus $O(mn)$ multiplication operations. As such, our AUDITINT protocol would be of independent value as it could also be used as the core of PDP/POR protocols.

Second, in our POSD scheme we used $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}}$ for verification, which is reminiscent of the signature scheme in [11]. However, our scheme is not a digital signature scheme because we actually allow a sort of manipulation. On the other hand, we use $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}}$ rather than, for example, $z_j = v_1^{-s_{j1}}$. This is because security of our construction partially relies on the DLOG problem, or more precisely the DLOG of v_2 with respect to base v_1.

Third, in the UPLOAD protocol, the purpose of t_i is to prevent the server from forging any new legitimate tuple of (x_i', y_{i1}', y_{i2}') from a legitimate (x_i, y_{i1}, y_{i2}) and F_i. To see this, let us consider the case without using t_i. Note that (x_i, y_{i1}, y_{i2}) with respect to block F_i satisfies

$$x_i = v_1^{y_{i1}} v_2^{y_{i2}} \prod_{j=1}^{m} z_j^{\mathsf{F}_{ij}} \mod p.$$

Without using t_i, the server can choose r_{i1}' and r_{i2}' from $\mathbb{Z}_q^*$, and set

$$x_i' = x_i v_1^{r_{i1}'} v_2^{r_{i2}'} \mod p,$$
$$y_{i1}' = y_{i1} + r_{i1}' \mod q,$$
$$y_{i2}' = y_{i2} + r_{i2}' \mod q$$

so that (x_i', y_{i1}', y_{i2}') also satisfies

$$x_i' = v_1^{y_{i1}'} v_2^{y_{i2}'} \prod_{j=1}^{m} z_j^{\mathsf{F}_{ij}} \mod p.$$

As another example of attacks, the server can generate (x_i', y_{i1}', y_{i2}') as follows: let $x_i' = x_i z_1 \mod p$, $y_{i1}' = y_{i1}$ and $y_{i2}' = y_{i2}$. During the execution of the UPLOAD protocol, the server may return $\mathsf{F}_i' = \{\mathsf{F}_{i1} + 1, \mathsf{F}_{i2}, \ldots, \mathsf{F}_{im}\}$ by adding one to the first symbol of block F_i. As a consequence, (x_i', y_{i1}', y_{i2}') and F_i' also satisfy

$$x_i' = v_1^{y_{i1}'} v_2^{y_{i2}'} z_1^{\mathsf{F}_{i1}+1} \prod_{j=2}^{m} z_j^{\mathsf{F}_{ij}} \mod p.$$

5.3 Correctness Analysis

With respect to correctness definition, the correctness of

the POSD scheme can be verified as follows:

$$
\begin{aligned}
& v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \\
= \; & v_1^{\sum_{i\in \mathrm{I}} \beta_i y_{i1}} v_2^{\sum_{i\in \mathrm{I}} \beta_i y_{i2}} \prod_{j=1}^{m} (v_1^{-s_{j1}} v_2^{-s_{j2}})^{\mu_j} \\
= \; & v_1^{\sum_{i\in \mathrm{I}} \beta_i y_{i1}} v_2^{\sum_{i\in \mathrm{I}} \beta_i y_{i2}} \prod_{j=1}^{m} (v_1^{-s_{j1}} v_2^{-s_{j2}})^{\sum_{i\in \mathrm{I}} \beta_i \mathsf{F}_{ij}} \\
= \; & v_1^{\sum_{i\in \mathrm{I}} \beta_i (r_{i1} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j1})} v_2^{\sum_{i\in \mathrm{I}} \beta_i (r_{i2} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j2})} \\
& \prod_{j=1}^{m} (v_1^{-s_{j1}} v_2^{-s_{j2}})^{\sum_{i\in \mathrm{I}} \beta_i \mathsf{F}_{ij}} \\
= \; & v_1^{\sum_{i\in \mathrm{I}} \beta_i r_{i1}} v_2^{\sum_{i\in \mathrm{I}} \beta_i r_{i2}} \\
= \; & (v_1^{r_{i1}} v_2^{r_{i2}})^{\sum_{i\in \mathrm{I}} \beta_i} \\
= \; & \prod_{i\in \mathrm{I}} x_i^{\beta_i} \\
= \; & \mathrm{X}.
\end{aligned}
$$

and

$$
\begin{aligned}
& e(\mathrm{T}, g) \\
= \; & e\left(\prod_{i\in \mathrm{I}} t_i^{\beta_i}, g\right) = e\left(\prod_{i\in \mathrm{I}} \left(H_1(\mathsf{fid}||i) u^{H_2(x_i)}\right)^{w\beta_i}, g\right) \\
= \; & e\left(\prod_{i\in \mathrm{I}} \left(H_1(\mathsf{fid}||i) u^{H_2(x_i)}\right)^{\beta_i}, g^w\right) \\
= \; & e\left(\left(\prod_{i\in \mathrm{I}} H_1(\mathsf{fid}||i) \prod_{i\in \mathrm{I}} u^{H_2(x_i)}\right)^{\beta_i}, z_g\right) \\
= \; & e\left(\mathrm{W} u^{\sum_{i\in \mathrm{I}} \beta_i H_2(x_i)}, z_g\right).
\end{aligned}
$$

5.4 Security Analysis

Now we prove that the POSD scheme satisfies Definitions 2 and 3.

THEOREM 1. *Assume H_1 and H_2 are hash functions modeled as random oracles, and the* CDH *problem is hard. The* POSD *scheme is* server_unforgeable.

PROOF. We show our proof through a sequence of games between a challenger, who plays the role of an honest client, and adversary $\mathcal{A}$, who acts as the malicious server. The overall proof strategy is: given fid' corresponding to $(\mathsf{F}, \mathtt{Tag}_{int})$ stored in the server and a challenge randomly selected by the challenger, if the adversary can pass the verification using $(\mathsf{F}', \mathtt{Tag}'_{int}) \neq (\mathsf{F}, \mathtt{Tag}_{int})$, then there is an algorithm that can solve the CDH problem.

Game$_0$: **Game**$_0$ is defined as in Definition 2, where the challenger only keeps the relevant public and private keys, and $\mathcal{Q}_{\mathsf{fid}}$, which is the list of the data file identifiers fid's it has used (as mentioned before, a PRF can reduce the storage of the fid's to constant).

Game$_1$: **Game**$_1$ is the same as **Game**$_0$ except that the challenger keeps $\mathcal{Q} = \{(\mathsf{fid}, \mathsf{F}, \mathtt{Tag}_{int})\}$,, the list of $(\mathsf{fid}, \mathsf{F}, \mathtt{Tag}_{int})$ involved in the execution of the UPLOAD protocol. In this case, we prove that if the adversary $\mathcal{A}$ can produce a forgery $(\mathsf{fid}', \mathsf{F}', \mathtt{Tag}'_{int}) \notin \mathcal{Q}$ that can pass the test in the AUDITINT protocol with respect to the challenger's challenge (I, β), then there is an efficient algorithm that can solve the CDH problem.

The simulator is constructed as follows:

- For generating the keys, the simulator works as follows:
  - Select v_1 and v_2 uniformly at random from $\mathbb{Z}_p^*$ such that the order of v_1 and v_2 is q. Select uniformly at random s_{j1} and s_{j2} from $\mathbb{Z}_q^*$ for $1 \leq j \leq m$. Set $z_j = v_1^{-s_{j1}} v_2^{-s_{j2}} \mod p$ for $1 \leq j \leq m$.
  - Let g be a generator of group G, and select h from G at random. Set $u = g^\gamma h^\eta$, where γ and η are chosen uniformly at random from $\mathbb{Z}_q^*$.
  - Select z_g uniformly at random from group G, which means that the simulator does not know the corresponding w with $z_g = g^w$.
  - Set $\mathrm{PK}_{int} = \{p, q, g, u, h, v_1, v_2, z_1, \ldots, z_m, z_g\}$ and $\mathrm{PK}_{dup} = \mathrm{PK}_{int}$. However, the simulator only knows secrets $\mathrm{SK} = \{(s_{11}, s_{12}), \ldots, (s_{m1}, s_{m2})\}$ but not the w.
- The simulator model $H_2(\cdot)$ as a random oracle. Given x_i, if x_i has been queried, return $H_2(x_i)$. Otherwise, select η uniformly at random from $\mathbb{Z}_q^*$ and return η. The simulator keeps the list of $(x_i, H_2(x_i))$.
- When the simulator is asked to compute $\mathtt{Tag}_{int}$ for data file F, the simulator executes the following: for each data block F_i where $1 \leq i \leq n$, select r_{i1} and r_{i2} uniformly at random from $\mathbb{Z}_q^*$ and computes

$$
\begin{aligned}
x_i &= v_1^{r_{i1}} v_2^{r_{i2}} \mod p, \\
y_{i1} &= r_{i1} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j1} \mod q, \\
y_{i2} &= r_{i2} + \sum_{j=1}^{m} \mathsf{F}_{ij} s_{j2} \mod q.
\end{aligned}
$$

  Select λ_i uniformly at random from $\mathbb{Z}_q^*$ and set

$$H_1(\mathsf{fid}||i) = g^{\lambda_i} / \left(u^{H_2(x_i)}\right).$$

  Thus, we have

$$t_i = \left(H_1(\mathsf{fid}||i) u^{H_2(x_i)}\right)^w = (g^w)^{\lambda_i} = (z_g)^{\lambda_i}.$$

  Set the cryptographic tag for block F_i as $(x_i, y_{i1}, y_{i2}, t_i)$ and thus $\mathtt{Tag}_{int} = \{(x_i, y_{i1}, y_{i2}, t_i)_{1\leq i\leq n}\}$. The simulator keeps the list of $(\mathsf{fid}||i, H_1(\mathsf{fid}||i))$. Note that λ_i is unknown to $\mathcal{A}$.
- When $\mathcal{A}$ queries $H_1(\mathsf{fid}||i)$ separately, the simulator operates as follows. If $\mathsf{fid}||i$ has been queried, return $H_1(\mathsf{fid}||i)$. Otherwise, select λ'_i uniformly at random from $\mathbb{Z}_q^*$ and return $h^{\lambda'_i}$. Note that λ'_i is unknown to $\mathcal{A}$.
- The simulator interacts with $\mathcal{A}$ until $\mathcal{A}$ outputs a forgery $(\mathsf{fid}', (\mathrm{I}, \beta), \{x_i\}_{i\in \mathrm{I}}, \mathrm{Y}'_1, \mathrm{Y}'_2, \mathrm{T}', \{\mu'_j\}_{1\leq j\leq m})$ at the Forgery stage and wins the game, where (I, β) is chosen at random by the simulator.

Suppose $\mathcal{A}$ produces $(\mathsf{fid}', (\mathrm{I}, \beta), \{x_i\}_{i\in\mathrm{I}}, \mathrm{Y}_1', \mathrm{Y}_2', T', \{\mu_j'\}_{1\le j\le m})$ As the probability that to win $\mathbf{Game}_1$. This means that $\mathsf{fid}' \in \mathcal{Q}_{\mathsf{fid}}$, but

$$(\{\mu_j'\}_{1\le j\le m}, \{x_i'\}_{i\in\mathrm{I}}, \mathrm{Y}_1', \mathrm{Y}_2', \mathrm{T}') \neq (\{\mu_j\}_{1\le j\le m}, \{x_i\}_{i\in\mathrm{I}}, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{T}), \quad (1)$$

where $\mu = \sum_{i\in\mathrm{I}} \beta_i \mathsf{F}_i \mod q$, $\mu' = \sum_{i\in\mathrm{I}} \beta_i \mathsf{F}_i' \mod q$, and $(\mathsf{fid}', \mathsf{F}, \mathtt{Tag}_{int}) \in \mathcal{Q}$ from which $\{x_i\}_{i\in\mathrm{I}}, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{T}$ are computed. The correctness of the scheme implies

$$e(\mathrm{T}, g) = e\left(\prod_{i\in\mathrm{I}} H_1(\mathsf{fid}'||i)^{\beta_i} u^{\sum_{i\in\mathrm{I}} \beta_i H_2(x_i)}, z_g\right) \quad (2)$$

Since $\mathcal{A}$ wins in $\mathbf{Game}_1$, we have

$$e(\mathrm{T}', g) = e\left(\prod_{i\in\mathrm{I}} H_1(\mathsf{fid}'||i)^{\beta_i} u^{\sum_{i\in\mathrm{I}} \beta_i H_2(x_i')}, z_g\right). \quad (3)$$

In what follows, we will consider three cases of Eq. (1):

- Case 1: $\mathrm{T} \neq \mathrm{T}'$.
- Case 2: $\mathrm{T} = \mathrm{T}'$, but $x_i \neq x_i'$ for some $i \in \mathrm{I}$.
- Case 3: $\mathrm{T} = \mathrm{T}'$, $x_i = x_i'$ for all $i \in \mathrm{I}$, but $(\mathrm{Y}_1, \mathrm{Y}_2, \{\mu_j\}_{1\le j\le m}) \neq (\mathrm{Y}_1', \mathrm{Y}_2', \{\mu_j'\}_{1\le j\le m})$.

In each case, we will utilize, among other things, Eqs. (2) and (3), to show that the simulator can solve the CDH problem, which means that $\mathcal{A}$ cannot win $\mathbf{Game}_1$ with a non-negligible probability. This will complete the proof.

Case 1: $\mathrm{T} \neq \mathrm{T}'$.
In this case, we have

$$e(\mathrm{T}/\mathrm{T}', g) = e\left(u^{\sum_{i\in\mathrm{I}} \beta_i (H_2(x_i) - H_2(x_i'))}, z_g\right).$$

By substituting u with $g^\gamma h^\eta$, we have

$$e(\mathrm{T}/\mathrm{T}', g) = e\left((g^\gamma h^\eta)^{\sum_{i\in\mathrm{I}} \beta_i (H_2(x_i) - H_2(x_i'))}, z_g\right).$$

Rearrange the terms, we get

$$\mathrm{T}/\mathrm{T}' = (g^{w\gamma} h^{w\eta})^{\sum_{i\in\mathrm{I}} \beta_i (H_2(x_i) - H_2(x_i'))}.$$

We claim that

$$\sum_{i\in\mathrm{I}} \beta_i (H_2(x_i) - H_2(x_i')) \neq 0 \mod q.$$

Otherwise, we get $\mathrm{T}/\mathrm{T}' = 1$, which contradicts the assumption $\mathrm{T} \neq \mathrm{T}'$. Together with the fact that $z_g = g^w$, we can get

$$h^w = \left((\mathrm{T}/\mathrm{T}') \cdot z_g^{-\gamma(\sum_{i\in\mathrm{I}} (H_2(x_i) - H_2(x_i')))}\right)^{\frac{1}{\sum_{i\in\mathrm{I}} (H_2(x_i) - H_2(x_i'))}},$$

which means that if $\mathrm{T} \neq \mathrm{T}'$, the simulator can solve the CDH problem by computing h^w with respect to given g and $z_g = g^w$ for unknown w.

Case 2: $\mathrm{T} = \mathrm{T}'$, but $x_i \neq x_i'$ for some $i \in \mathrm{I}$.
Because $\mathrm{T} = \mathrm{T}'$, we have

$$\prod_{i\in\mathrm{I}} H_1(\mathsf{fid}'||i)^{\beta_i} u^{\sum_{i\in\mathrm{I}} \beta_i H_2(x_i)} = \prod_{i\in\mathrm{I}} H_1(\mathsf{fid}'||i)^{\beta_i} u^{\sum_{i\in\mathrm{I}} \beta_i H_2(x_i')}.$$

By arranging the term, we have

$$u^{\sum_{i\in\mathrm{I}} \beta_i (H_2(x_i) - H_2(x_i'))} = 1.$$

$$\sum_{i\in\mathrm{I}} \beta_i (H_2(x_i) - H_2(x_i')) = 0 \mod q$$

is negligible and $u = g^\gamma h^\eta$, we have

$$\begin{aligned} h &= g^{\frac{\sum_{i\in\mathrm{I}} \gamma (H_2(x_i) - H_2(x_i'))}{\sum_{i\in\mathrm{I}} \eta (H_2(x_i) - H_2(x_i'))}} \\ &= g^{\gamma\eta^{-1} \sum_{i\in\mathrm{I}} (H_2(x_i) - H_2(x_i'))}. \end{aligned}$$

This means that the simulator can solve the DLOG of random h with respect to base g, which immediately breaks the CDH assumption.

Case 3: $\mathrm{T} = \mathrm{T}'$, $x_i = x_i'$ for all $i \in \mathrm{I}$, but $(\mathrm{Y}_1, \mathrm{Y}_2, \{\mu_j\}_{1\le j\le m}) \neq (\mathrm{Y}_1', \mathrm{Y}_2', \{\mu_j'\}_{1\le j\le m})$.
Note that

$$\prod_{i\in\mathrm{I}} {x'}_i^{\beta_i} = v_1^{\mathrm{Y}_1'} v_2^{\mathrm{Y}_2'} \prod_{j=1}^{m} z_j^{\mu_j'} \mod p$$

and

$$\prod_{i\in\mathrm{I}} x_i^{\beta_i} = v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \mod p.$$

Because $x_i = x_i'$ for all $i \in \mathrm{I}$, we have

$$v_1^{\mathrm{Y}_1'} v_2^{\mathrm{Y}_2'} \prod_{j=1}^{m} z_j^{\mu_j'} = v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \mod p.$$

By replacing z_j with $v_1^{-s_{j1}} v_2^{-s_{j2}}$ in the above equation, we get

$$v_1^{\mathrm{Y}_1' - \mathrm{Y}_1 - \sum_{j=1}^m s_{j1}(\mu_j' - \mu_j)} = v_2^{\mathrm{Y}_2 - \mathrm{Y}_2' - \sum_{j=1}^m s_{j1}(\mu_j - \mu_j')} \mod p.$$

Thus, if $(\mathrm{Y}_1, \mathrm{Y}_2, \{\mu_j\}_{1\le j\le m}) \neq (\mathrm{Y}_1', \mathrm{Y}_2', \{\mu_j'\}_{1\le j\le m})$, then the simulator can compute the DLOG of random v_2 with respect to base v_1, which immediately breaks the CDH assumption. □

THEOREM 2. *Assume H_1 and H_2 are hash functions modeled as random oracles, and the* CDH *problem is hard. The* POSD *scheme is (κ, θ)_uncheatable with respect to challenge of $c = \frac{n}{\theta - \kappa} \log(2^{\theta-\kappa} + \epsilon)$ blocks in the* DEDUP *protocol, where κ is the min-entropy of the file F in question, θ is the amount of entropy leaked to or stolen by the adversary, and ϵ is negligible in security parameter ℓ.*

PROOF. According to Theorem 1, given that H_1 and H_2 are hash functions modeled as random oracles, and the CDH problem is hard, our scheme is server_unforgeable. That is, given the challenge $\{\mathrm{I}, \beta\}$ with file identifier fid, the response $\{\mu_1, \cdots, \mu_m, \{x_i\}_{i\in\mathrm{I}}, \mathrm{Y}_1, \mathrm{Y}_2, T\}$ must be computed honestly from $(\mathsf{fid}, \mathsf{F}, \mathtt{Tag}_{int})$, so that

$$\begin{aligned} \prod_{i\in\mathrm{I}} x_i^{\beta_i} &= v_1^{\mathrm{Y}_1} v_2^{\mathrm{Y}_2} \prod_{j=1}^{m} z_j^{\mu_j} \mod p, \\ e(\mathrm{T}, g) &= e\left(\prod_{i\in I} H_1(\mathsf{fid}||i)^{\beta_i} u^{\sum_{i\in\mathrm{I}} \beta_i H_2(x_i)}, z_g\right) \end{aligned}$$

Without loss of generality, let (I, β) be the challenge with corresponding file identifier fid in the execution of DEDUP. Let $(\mu_1', \cdots, \mu_m')$ be the response from the malicious client and $\{x_i'\}_{i\in\mathrm{I}}, \mathrm{Y}_1', \mathrm{Y}_2', \mathrm{T}'$ are computed from $\mathtt{Tag}_{dup}$ by the

cloud server. Recall that $\mathtt{Tag}_{dup} = \mathtt{Tag}_{int}$, then we have $\{x'_i = x_i\}_{i\in \mathrm{I}}, \mathrm{Y}'_1 = \mathrm{Y}_1, \mathrm{Y}'_2 = \mathrm{Y}_2, \mathrm{T}' = \mathrm{T}$. Therefore, in order to satisfy

$$\prod_{i\in \mathrm{I}} {x'}_i^{\beta_i} = v_1^{\mathrm{Y}'_1} v_2^{\mathrm{Y}'_2} \prod_{j=1}^{m} z_j^{\mu'_j} \mod p,$$

$$e(\mathrm{T}', g) = e\left(\prod_{i\in I} H_1(\mathsf{fid}||i)^{\beta_i} u^{\sum_{i\in \mathrm{I}} \beta_i H_2(x'_i)}, z_g\right)$$

we have $\mu'_j = \mu_j, 1 \le j \le m$, otherwise we can solve the DLOG problem of random v_2 with respect to base v_1 (see case 3 in the proof of Theorem 1). That is, the malicious client must compute $(\mu'_1, \cdots, \mu'_m)$ honestly from F. In other words, the malicious client can win the game only if it can figure out the unknown bits entropy in the data blocks specified by the set I.

Let Evnt denote the event that there are c data blocks with unknown bits entropy and the adversary tries to guess the unknown bits entropy in order to cheating successfully. In order to simplify the model, assume that the unknown bits entropy distributes over the data blocks of F uniformly. Meanwhile, because the challenged data blocks are chosen uniformly at random so that we can assume that the unknown bits entropy distributes over F uniformly. Therefore the probability

$$\begin{aligned}
\Pr[\mathsf{Evnt}] &= \frac{1}{2^{c(\kappa-\theta)/n}} \\
&= 2^{c(\theta-\kappa)/n} \\
&= 2^{\log(2^{\theta-\kappa}+\epsilon)} \quad \left(c = \frac{n}{\theta-\kappa}\log(2^{\theta-\kappa}+\epsilon)\right) \\
&= 2^{\theta-\kappa} + \epsilon \\
&= \frac{1}{2^{\kappa-\theta}} + \epsilon.
\end{aligned}$$

This completes the proof. □

5.5 Performance Analysis

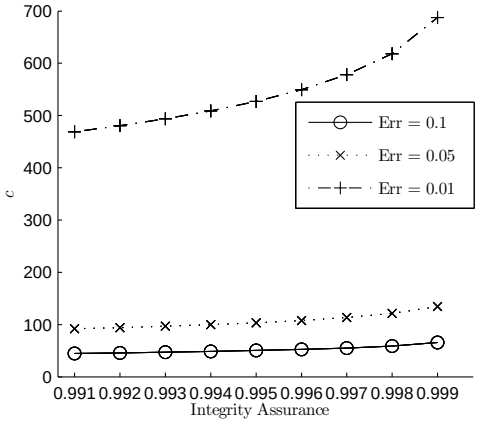

Figure 1: The impact of ERR and integrity assurance on challenge size c

On the size of the challenges. The size of the challenges in the AUDITINT protocol is an important performance parameter. Let ERR be the probability of block being corrupted (i.e., portions of the data modified by the server). Figure 1 shows the required size of challenges in order to achieve integrity assurance in the interval $[0.991, 0.999]$ under three circumstances: ERR= 0.1, ERR= 0.05 and ERR= 0.01. Consider, for example, the case of ERR= 0.01. It only requires to send less than 500 challenges in order to achieve 99.1% integrity assurance, regardless of the size of the data blocks. This also explain the advantage of POR and PDP.

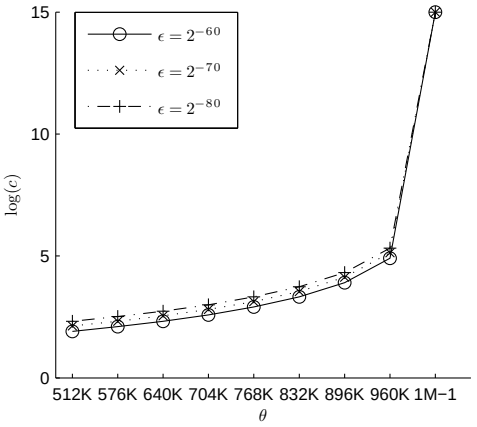

Figure 2: The impact of θ on the challenge size c: $n = 2^{15}$ and $\kappa = 1M$ bits

Theorem 2 shows a lower bound on the number of challenged data blocks in order to fulfill (κ, θ)-*uncheatability*. In order to illustrate the impact of the lower bound of c with parameters κ, θ, n and ϵ, we consider two cases: one file with small min-entropy (1M bits) and the other with large min-entropy (128M bits). With respect to negligible probability 2^{-80}, Figure 2 shows that given a file of 2^{15} data blocks with 1M bits min-entropy, our scheme can fulfill $(1M, 960K)$-*uncheatability* by challenging about 2^6 data blocks (or 0.1% portions of the data file).

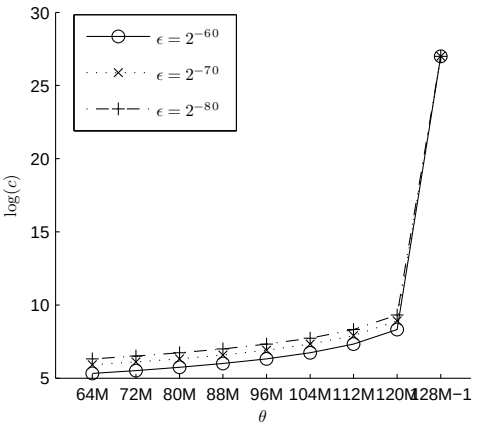

Figure 3: The impact of θ on the challenge size c: $n = 2^{27}$ and $\kappa = 128M$ bits

Figure 3 shows that given a file of 2^{27} data blocks with 128M bits min-entropy, fulfilling $(128M, 120M)$-*uncheatability* requires to challenge about 2^9 data blocks (or 2^{-18} portions

of the data file). This shows that even if the adversary has obtained 93.75% of the data file (e.g., by penetrating into the cloud server in a stealthy fashion and without being detected), the attacker cannot cheat against reasonably small challenges.

Comparison with some relevant schemes. Because POSD is the first scheme that simultaneously allows proof of storage integrity and deduplication, in Table 1 we compare its efficiency to the most efficient PDP scheme in [2], the most efficient POR scheme in [12], and the only existing POW scheme in [13], respectively. The two particular PDP and POR schemes are chosen also because they offer the afore-mentioned *public verifiability*, namely that a third-party can examine the storage integrity on behalf of a client, which is exploited to construct POSD. The POW scheme is the one based on Merkle-tree in [13]; it is chosen because its security is compatible with our POSD scheme (there are more efficient but less secure solutions in[13]). Note that in the client storage, we consider a single file F. In principle, each client can outsources polynomially-many data files to the cloud. In this case, storage of the identifies, fid's, still can be made constant by letting each client use a Pseudo-random Function PRF to generate fid's from its secret key while maintaining a counter.

From the perspective of assuring cloud data storage integrity, we draw the following observations from Table 1. First, our POSD scheme requires $O(n)$ exponentiations for a client to preprocess a data file before uploading it to the cloud. This complexity is substantially smaller than the preprocessing complexity $O(mn)$ exponentiations of the schemes in [2, 12]. Second, our POSD scheme incurs $O((m+c)\ell)$ communication overhead in the audit process, which is higher than the $O(m\ell)$ communication overhead of the PDP and POR schemes. To demonstrate that the extra communication is not significant especially when we deal with large files, let us consider the following realistic example. Suppose a data file consists of 2^{27} blocks of 2^8 symbols (2.5-GB file if $\ell = 160$). Assume that the probability of block corruption is $\text{ERR} = 0.01$. That is, roughly 2^{21} data blocks are corrupted. Suppose we want to achieve 99.5% integrity assurance (i.e., with probability 99.5% the tamperation of a data file is detected), the extra communication overhead in our POSD scheme is only 2^{16} bits (8KB). Moreover, it should be noted that the PDP and POR schemes cannot fulfill deduplication.

From the perspective of secure data deduplication, , we draw the following observations from Table 1. First, our POSD scheme is slightly less efficient than the POW scheme. However, the POW scheme cannot fulfill the auditability of cloud storage security (note that it was well-known that Merkle-tree is not sufficient to fulfill PDP/POR [2, 12, 16]). Second, our POSD scheme incurs smaller communication overhead because $O(m\ell)$ is often much smaller than $O(c\log(n)m\ell)$. Third, the POW scheme is indeed secure in the standard model based on the existence of Collision-Resistant Hash (CRH) functions. However, it cannot fulfill auditability of cloud storage integrity.

6. CONCLUSION

We motivated the need of the cloud storage notion we call proof of storage with deduplication or POSD, to fulfill data integrity and duplication simultaneously. We also presented an efficient POSD scheme, which is proven secure in the Random Oracle model based on the Computational Diffie-Hellman assumption. Compared with the PDP/POR/POW schemes, which cannot achieve one of the two goals, our scheme is as efficient as theirs.

One interesting future work is to remove the random oracle in the protocol without jeopardizing performance. Another is to seek a different design methodology for such protocols so as to achieve even substantially better performance.

7. REFERENCES

[1] The digital universe decade - are you ready? International Data Corporation, 2010. http://idcdocserv.com/925.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 598–609, New York, NY, USA, 2007. ACM.

[3] G. Ateniese, R. Di 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 '08, pages 9:1–9:10, New York, NY, USA, 2008. ACM.

[4] G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 319–333, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 43–54, New York, NY, USA, 2009. ACM.

[6] Y. Dodis, S. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 109–127, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, ICDCS '02, pages 617–, Washington, DC, USA, 2002. IEEE Computer Society.

[8] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 213–222, New York, NY, USA, 2009. ACM.

[9] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security and Privacy*, 8:40–47, November 2010.

[10] A. Juels and B. S. Kaliski, Jr. Pors: proofs of retrievability for large files. In *Proceedings of the 14th*

	PDP [2]	POR [12]	POSD (this paper)	POW [13]
total key size	$O(m)$	$O(m)$	$O(m)$	0 (no keys)
use Random Oracle?	yes	yes	yes	no
security assumption	RSA	CDH	CDH	CRH
For integrity audit purpose				
client storage	$O(1)$	$O(1)$	$O(1)$	N/A
server storage	$O(n)$	$O(n)$	$O(n)$	N/A
audit preprocessing comp.	$O(mn)$Ex+ $O(mn)$Mu	$O(mn)$Ex+$O(mn)$Mu	$O(n)$Ex+$O(mn)$Mu	N/A
audit computation (client)	$O(c)$Ex+ $O(cm)$Mu	$O(c)$Ex+ $O(cm)$Mu	$O(c)$Ex+ $O(cm)$Mu	N/A
audit computation (server)	add	add	add	N/A
audit communication	$O(m\ell)$	$O(m\ell)$	$O((m+c)\ell)$	N/A
integrity assurance	$1-(1-\text{Err})^c$	$1-(1-\text{Err})^c$	$1-(1-\text{Err})^c$	N/A
For deduplication purpose				
dedup preprocessing comp.	N/A	N/A	$O(n)$Ex+$O(mn)$Mu	ECC +$O(n^2)$H
dedup. computation (client)	N/A	N/A	$O(cm)$Mu	$O(n^2)$H
dedup. computation (server)	N/A	N/A	$O(c)$Ex+$O(cm)$Mu	$O(c\log(n))$H
dedup. communication	N/A	N/A	$O(\ell m)$	$O(cm\ell\log(n))$

Table 1: Efficiency comparison between some PDP, POR, POW and our POSD schemes, where n is the number of blocks of a data file, m is the number of symbols of a data block, c is the number of blocks that will be challenged, Err is the probability of block corruption, Ex represents modular exponentiation operation, Mu represents modular multiplication operation, and N/A indicates that a property is not applicable to a certain scheme.

ACM conference on Computer and communications security, CCS '07, pages 584–597, New York, NY, USA, 2007. ACM.

[11] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 31–53, London, UK, 1993. Springer-Verlag.

[12] H. Shacham and B. Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] B. P. A. S.-P. Shai Halevi, Danny Harnik. Proofs of ownership in remote storage systems. Cryptology ePrint Archive, Report 2011/207, 2011. http://eprint.iacr.org/.

[14] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller. Secure data deduplication. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, StorageSS '08, pages 1–10, New York, NY, USA, 2008. ACM.

[15] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Trans. Parallel Distrib. Syst.*, 22:847–859, May 2011.

[16] Q. Zheng and S. Xu. Fair and dynamic proofs of retrievability. In *Proceedings of the first ACM conference on Data and application security and privacy*, CODASPY '11, pages 237–248, New York, NY, USA, 2011. ACM.