SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2015; **8:**2457–2468
Published online 19 July 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.784

SPECIAL ISSUE PAPER

Provable ownership of files in deduplication cloud storage

Chao Yang[1,2], Jian Ren[2]* and Jianfeng Ma[1]

[1] School of CS, Xidian University, Xi'an, Shaanxi, 710071, China
[2] Department of ECE, Michigan State University, East Lansing, MI 48824, U.S.A

ABSTRACT

With the rapid adoption of cloud storage services, a great deal of data is being stored at remote servers, so a new technology, client-side deduplication, which stores only a single copy of repeating data, is proposed to identify the client's deduplication and save the bandwidth of uploading copies of existing files to the server. It was recently found, however, that this promising technology is vulnerable to a new kind of attack in which by learning just a small piece of information about the file, namely its hash value, an attacker is able to obtain the entire file from the server. In this paper, to solve this problem, we propose a cryptographically secure and efficient scheme for a client to prove to the server his ownership on the basis of actual possession of the entire original file instead of only partial information about it. Our scheme utilizes the technique of spot checking in which the client only needs to access small portions of the original file, dynamic coefficients and randomly chosen indices of the original files. Our extensive security analysis shows that the proposed scheme can generate provable ownership of the file and maintain high detection probability of client misbehavior. Both performance analysis and simulation results demonstrate that our proposed scheme is much more efficient than the existing schemes, especially in reducing the burden of the client. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS
cloud storage; deduplication; provable ownership; spot checking

***Correspondence**
Jian Ren, Department of ECE, Michigan State University, East Lansing, MI 48824, U.S.A
E-mail: renjian@msu.edu

1. INTRODUCTION

With the rapid adoption of cloud services, more and more volume of data is stored at remote servers, so techniques to save disk space and network bandwidth are needed. A key concept in this context is deduplication, in which the server stores only a single copy of each file, regardless of how many clients want to store that file. All clients possessing that file only use the link to the single copy of the file stored at the server. Furthermore, if the server already has a copy of the file, then clients do not have to upload it again to the server, which will save bandwidth as well as storage. This technique is called client-side deduplication. It is reported that business applications can achieve deduplication ratios from 1:10 to as much as 1:500, resulting in disk and bandwidth savings of more than 90% [1].

However, client-side deduplication introduces some new security problems. Harnik *et al*. found that when a server tells a client that it does not have to transmit the file, it means that some other clients have the same file already stored at the server, which could be a sensitive information [2]. More seriously, Halevi *et al*. recently found some new attacks to the client-side deduplication system [3]. In these attacks, by learning just a small piece of information about the file, namely its hash value, an attacker is able to obtain the entire file from the server. These attacks are not just theoretical. Some similar attacks that were implemented against Dropbox [4] were also discovered by Mulazzani *et al*. [5] recently. The root cause of all these attacks is that there is a very short and fixed piece of information that is used to represent the whole file. An attacker that learns it can have access to the entire file.

To solve the aforementioned problem, a solution where a client proves to the server that it indeed has the entire file without uploading it must be carefully designed. The solution and corresponding protocol should meet some important requirements as follows:

Bandwidth requirements: The protocol of proof runs between the server and the client should be bandwidth

efficient. Specifically, it must consume much less bandwidth than the size of the file; otherwise, the client could just upload the file itself.

Computation requirements: The server typically has to handle a large number of files concurrently. So the solution should not impose too much burden on the server, even though it has more powerful computation capability. On the other hand, the client, with limited storage as well as computation resources, is the leading actor in the deduplication scenario who has to prove to the server that it possesses the exact same file already stored at the server. So the design of the solution should pay more attention to reducing the burden on the client in terms of the computation and storage and, at the same time, keeping the burden on the server at a relatively low level.

Security requirements: The verification and proof should be based on the availability of the original data in its original form, instead of any stored message authentication code or previously used verification results. Furthermore, the requested parts of the original file should be randomly chosen every time, and the generated proofs should be totally different from any other proof in each challenge. So it is infeasible for anybody to forge or prepare the proof in advance and to pass the verification challenge.

Recently, the importance of ensuring remote data integrity has been highlighted by works of the proofs of retrievability (PoR) [6–8] and proofs of data possession (PDP) [9,10]. In these works, it is the server that will prove to the client that it possesses the original file correctly and integrally. But in the scenario of client-side deduplication, there is a total role reversal between the server and the client. However, this role reversal is significant, and especially the PoR and PDP protocols in the literatures are inapplicable to this scenario. In addition, these protocols are vulnerable to the recent security attacks using a small hash value as a proxy for the entire file, and only a few solutions to the new problem have been proposed [3,5]. Unfortunately, these solutions either cannot provably meet the security requirements mentioned earlier because of static spot checking [11] or have a relatively high computational complexity. As a result, their applicability in the scenario of client-side deduplication is greatly limited.

In this paper, to solve the problem in the scenario of client-side deduplication, we propose a cryptographically secure and efficient scheme, called a provable ownership of files (POF), for a client to prove to the server that it indeed has the file. We achieve the efficient goal by relying on dynamic spot checking [11], in which the client only needs to access small but dynamic portions of the original file to generate the proof of possession of the original file, thus greatly reducing the burden of computation on the client and minimizing the input/output (I/O) between the client and the server. At the same time, by utilizing dynamic coefficients and randomly chosen indices of the original files, our scheme mixes the randomly sampled portions of the original file with the dynamic coefficients to generate a unique proof in every challenge. This technique helps us meet the key security requirements and achieve the provable security goal.

The rest of the paper is organized as follows. Section 2 overviews the related work. Then, we introduce the system model, adversary model and notations in Section 3 and present the detailed description of our scheme in Section 4. Sections 5 and 6 provide the security analysis and performance evaluations, respectively. Finally, Section 7 gives the concluding remark of the whole paper.

2. RELATED WORK

Juels and Kaliski [6] presented a formal PoR model to check the remote data integrity. Their scheme uses disguised blocks called sentinels hidden in the original file blocks and an error-correcting code to detect data corruption and ensure both possession and retrievability of files at remote cloud servers. From this model, Shacham and Waters [7] constructed a random linear function-based homomorphic authenticator that enables an unlimited number of checks and greatly decreases communication overhead. Wang *et al*. [8] improved the PoR model [7] by introducing the Merkle hash tree construction for forming the authentication block tag and addressed the problem of how to provide public verifiability and data dynamic operation support for remote data integrity check in cloud computing. Ateniese *et al*. [9,10] defined the PDP model for ensuring possession of file on untrusted storage. This scheme utilized public key-based homomorphic tags for authenticating the data file and supported public verifiability. It is noted that our scheme is somewhat similar to these solutions. However, there is a big difference between them: in their scenario, the server proves to the client that it stores the file integrally, but in our scenario of client-side deduplication, with a big role reversal, it is the client that proves to the server that it indeed possesses the original file completely. Furthermore, this role reversal is significant, and the PoR and PDP schemes are not applicable in our scenario. The reason is that, in the PoR and PDP schemes, the input file must be pre-processed by the client to embed some secrets into it. This method makes it possible for the server to show the proof of possessing the file by replying to the client queries with answers that are consistent with these secrets. In the scenario of client-side deduplication, a new client has the only original file itself, so it is impossible to insert secrets into it in advance for the purpose of proof. This excludes many similar solutions [6,7,9,12–14].

Recently, some security problems to the client-side deduplication system have been identified. Harnik *et al*. found that the deduplication systems could be used as a side channel to reveal sensitive information about the contents of files and also could be used as a covert channel by which malicious software can communicate with the outside world, regardless of the firewall settings of the compromised machine [2]. They suggested using encryption

DOI: 10.1002/sec

to address the problem, but the encryption will essentially prevent deduplication from taking place and might therefore be unacceptable. Also, they wanted to decrease the security risks of deduplication by setting a random threshold for every file and performing deduplication only if the number of copies of the file exceeds this threshold. But their solution only reduces the security risks instead of getting rid of it. Mulazzani *et al*. [5] discovered and implemented a new attack against Dropbox's deduplication system in which an attacker can obtain the original file from the server after acquiring only a small piece of information about the file. They also proposed a preliminary and simple revisal to the communication protocol of Dropbox to address the problem. But their scheme has limited security features and was not well evaluated. At the same time, Halevi *et al*. also found some similar attacks to the client-side deduplication system in their work [3]. They put forward the proof-of-ownership (POW) model, in which a client can prove to a server on the basis of the Merkle tree and error-control encoding that it indeed has a copy of a file without actually uploading it. However, their scheme cannot guarantee the freshness of the proof in every challenge. Furthermore, their scheme has to build the Merkle tree on the encoded data, which is inherently inefficient.

Another well-studied related problem is how to verify the integrity of memory contents in computing platforms and ensure that no memory corruption affects the performed computations [15]. Many of these schemes utilize the Merkle tree [16] for memory and data authentication.

3. SYSTEM AND ADVERSARY MODEL

3.1. System model

A typical network architecture for cloud data storage is illustrated in Figure 1. There are two entities as follows:

Storage server: It will provide cloud storage service to all kinds of users. Its computation and storage capability (CPU, I/O, network, etc.) are stronger than each single user. The storage server will maintain the integrity of users' data and the availability of cloud service.

Client users. There are many client users who create user accounts and passwords from a storage server. Each of them could log into the cloud storage server with their accounts and then upload files to or retrieve files from the server.

In the deduplication scenario, the server keeps a single copy of the original file, regardless of the number of clients that request to store the file. All client users that possess the original file only use the link to the single copy of the original file stored at the server. Specifically, a client user first computes the hash value of the original file by cryptographic hash functions and sends it to the server, and then the server checks whether the hash value already exists in its database. If the hash value is the same as an existing hash value stored at the server, the server will challenge the client to ask for the proof of possession of the original file. Upon a successful challenge and proof, the client does not have to upload the file again to the server. At the same time, the server marks the client as an owner of the original file. From then on, there is no difference between the client and the users who uploaded the original file. Thus, the deduplication process saves bandwidth as well as storage capacity.

3.2. Adversary model

We consider three kinds of security threats to storage systems that carry out client-side deduplication across multiple users. First, the deduplication system may use a standard hash function (e.g., SHA256). In this case, it is probable that an attacker could obtain hash values of files owned by other clients who may have published the hash value as the signature of the file or used the hash value elsewhere. The weak adversary could achieve this kind of attack by public ways, or they might receive a file that is

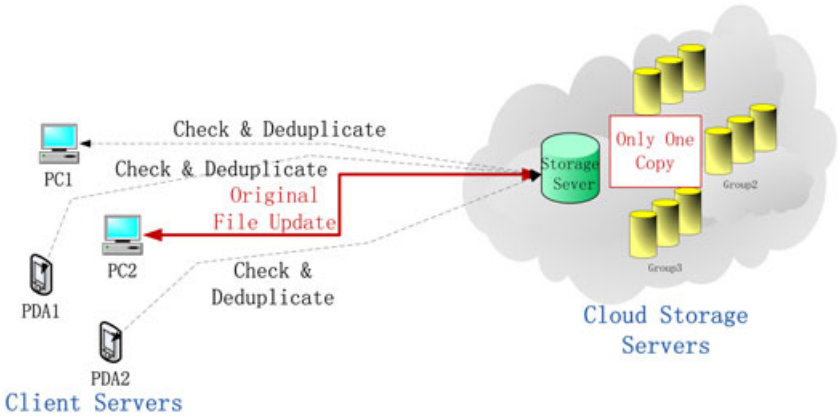

Figure 1. Cloud data storage with deduplication.

similar to the file of interest via other channels. For example, different employees of the same company may obtain long profile documents from the company, where such documents may only differ in small places such as personal particulars and salaries. Second, taking a strong adversary into consideration, an attacker may be able to compromise a server temporarily, obtaining access to its stored data, which include the hash values for the files stored on it. Obtaining the proxy of a file, the attacker can download the original file, which may include confidential data or information. The attacker also may be able to publish these stolen hash values and make anyone else obtain these private files. Last but not least, it is plausible that similar attacks would happen on the client side. For example, an attacker could embed some malicious software into a client's machine. The malicious software could use a low-bandwidth covert channel to send out the hash values of interesting files and enable others to obtain the private files from the cloud server. This attack can be aggravated even further in the following way. Some examinations revealed that the client software of storage services, including Dropbox, stores local unencrypted manifest files. These files include an entry for each file stored by the service, which includes the hash value of the file. A malicious software can leak to the adversary the hash value that can be used to recover the manifest file and the hash values that can be used to recover files of interest.

4. PROVABLE OWNERSHIP OF FILES IN DEDUPLICATION

4.1. Notation and preliminaries

F - The original file
f - The number of blocks that the original file is divided into and stored.
$(b_1, ..., b_f)$ - A collection of all blocks of the original file
α - A pseudo-random function defined as $\alpha : \{0,1\}^* \times \text{key} \rightarrow \{0,1\}^{\mu}$, where
μ is a security parameter
β - A pseudo-random permutation defined as $\beta : \{0,1\}^q \times \text{key} \rightarrow \{0,1\}^q$;
h_k - A keyed-cryptographic hash function with key k as an input
c - The number of blocks requested in a single challenge
sk - The symmetrical key shared between the client user and the storage server
S_1, S_2 - Random seeds used to form the challenge set in every challenge: $S_1 \leftarrow_R \{0,1\}^*, S_2 \leftarrow_R \{0,1\}^*$
R_c - A random number used to generate the session key in every challenge:
$R_c \leftarrow_R \{0,1\}^*$
TS - The current timestamp

4.2. Definitions

We start with the definition of a POF scheme and protocol.

Definition 1 (POF). *A POF scheme is a collection of three polynomial-time algorithms (KeyDeriving, ProofGen and ProofCheck):*

KeyDeriving: $(sk, R_c) \rightarrow \{K_s, S_1, S_2\}$ *is a key generation algorithm that is run by the server to set up the scheme. It takes as input the symmetrical key shared between the client and the storage server sk and a random number* R_c*. It returns a new session key and two random seeds:* (K_s, S_1, S_2)*;*
ProofGen $(K_s, F, Chal) \rightarrow V$ *is run by the client to generate a POW of the original file. It takes as inputs a secret session key* K_s*, a collection of* f *blocks of the file* F *and a challenge set Chal. It returns a POW of the original file* V *based on the blocks in the file* F *determined by the challenge set Chal.*
ProofCheck $(K_s, Chal, F, V) \rightarrow \{$*"True", "False"*$\}$ *is run by the server to verify that the client indeed has the POF. It takes as inputs a secret session key* K_s*, a challenge set Chal, a collection of* f *blocks of the file* F *and a proof* V*. It returns either "True" or "False" depending on whether the* V *is a correct POW of the original file based on the blocks determined by Chal.*

We construct the POF protocol from the POF scheme in two phases, *setup* and *challenge*.

Setup: The server S possesses the original file F and divides it into blocks to store them. The server S and the client C run $KeyDeriving(sk, R_c) \rightarrow \{K_s, S_1, S_2\}$ to generate and store the same new session key K_s and the output random seeds.
Challenge: The server S generates a challenge set *Chal* indicating the specific blocks for which S wants to ask for a POF. S then sends the *Chal* to C. C runs $ProofGen(K_s, F', Chal) \rightarrow V$ and sends the generated proof V back to S. Finally, S can check the validity of the proof V by running $ProofCheck(K_s, Chal, F, V)$.

These preceding steps can be executed an unlimited number of times to confirm whether C has ownership of the original file.

4.3. Efficient and secure provable-ownership-of-files scheme

Next, we elaborate on the POF scheme and the corresponding POF protocol presented in Algorithms 1 and 2, respectively.

For each challenge, the proof V will be computed with different and fresh coefficients δ_τ, which are determined by a pseudo-random function with a fresh randomly generated key and randomly requested file blocks. This POF algorithm ensures that the generated proof V is based on the availability of the original data in its original form, instead of any stored message authentication code or previously used verification results. In other words, it ensures that the client C exactly possesses each one of the requested blocks.

Also, the efficient communication between the server and the client will be maintained because of the final combined proofs of each blocks.

Algorithm 1 A POF scheme

KeyDeriving(sk, R_c):

1: Choose two random seeds $S_1 \leftarrow_R \{0,1\}^*$ and $S_2 \leftarrow_R \{0,1\}^*$, and generate a new session key $K_s = h_{sk}(R_c)$, where sk is the symmetrical key shared between the client and the storage server and $R_c \leftarrow_R \{0,1\}^*$, and then output (K_s, S_1, S_2);

ProofGen$(K_s, F', Chal)$:

1: Let $F' = (b'_1, b'_2, \ldots, b'_f)$ and $(c, S_1, S_2) = Chal$, where $1 \le c \le f$;
2: Compute temporary keys: $k_1 = h_{K_s}(S_1)$, $k_2 = h_{K_s}(S_2)$;
3: For $1 \le \tau \le c$,

- Derive the indices of the blocks for which the proof is generated: $i_\tau = \beta_{k_1}(\tau)$;
- Compute dynamic coefficients: $\delta_\tau = \alpha_{k_2}(\tau)$;

4: Compute $H' = h_{K_s}[h_{K_s}(b'_{i_1}, \delta_1) \parallel h_{K_s}(b'_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b'_{i_\tau}, \delta_\tau)]$, where $\parallel$ denotes concatenation;
5: Output $V = (H')$.

ProofCheck$(K_s, Chal, F, V)$:

1: Let $F = (b_1, b_2, \ldots, b_f)$, $V = (H')$ and $(c, S_1, S_2) = Chal$, where $1 \le c \le f$;
2: Compute temporary keys: $k_1 = h_{K_s}(S_1)$, $k_2 = h_{K_s}(S_2)$;
3: For $1 \le \tau \le c$,

- Derive the indices of the blocks for which the proof is generated: $i_\tau = \beta_{k_1}(\tau)$;
- Compute dynamic coefficients: $\delta_\tau = \alpha_{k_2}(\tau)$;

4: Compute $H = h_{K_s}[h_{K_s}(b_{i_1}, \delta_1) \parallel h_{K_s}(b_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b_{i_\tau}, \delta_\tau)]$, where $\parallel$ denotes concatenation;
5: If $H = V$, then output "True"; otherwise, output "False".

Algorithm 2 A protocol of POF scheme (POF protocol)

Setup

1: The server S decomposes the file F into f blocks, $b_1, \ldots, b_f$, to avoid data encryption. The f blocks of the file may be stored in f logically different locations.
2: Then, the server S in possession of the file F chooses a random number R_c, identifies the corresponding symmetrical key sk shared with the client and runs *KeyDeriving*(sk, R_c) to generate a new session key K_s and two random seeds S_1 and S_2; S then sends the random number R_c to the client C who will run the same function to generate the same new session key K_s, omit the random seed output and send back $h_{K_s}(R_c, TS) \| TS$ to confirm the K_s; the session key K_s will be kept secret by the client and server, and the random number R_c may be deleted.

Challenge

1: The client claims ownership of a specific file stored at the server and requests to start a proof process;
2: The server S prepares to obtain the POW for c distinct blocks of the specific file F, where $1 \le c \le f$;
3: The server S forms the $(c, S_1, S_2) = Chal$ to determine the indices of c distinct blocks it wants to challenge and the two random seeds S_1 and S_2, which will be used to generate the dynamic coefficients δ_τ, and sends $Chal$ to the client C;
4: The client C runs *ProofGen*$(K_s, F' = (b'_1, b'_2, \ldots, b'_f), Chal(c, S_1, S_2)) \rightarrow V$ and sends back to S the POW of the original file V;
5: The server S sets $(c, S_1, S_2) = Chal$ and checks the validity of the proof V by running *ProofCheck*$(K_s, Chal, F, V)$.

5. SECURITY ANALYSIS

5.1. Security proof and characteristic analysis

One of the major design goals is to prevent the client from producing an authenticator for the remote server by simply accessing a fingerprint of the original file to meet the verification sent from the remote server. Our proposed scheme and protocol make it computationally infeasible for the cheating remote client, without possessing a specific file, to convince the remote server that it has ownership of the file. In fact, we have the following theorem.

Theorem 1. *For the proposed POF scheme, the complexity for cheating of the ownership verification is at least as difficult as performing a strong collision attack of the hash function.*

Proof. Suppose that the remote client can generate the response without accessing the original data. This requires the remote client to be able to produce a quantity x so that $h(x) = h_{K_s}[h_{K_s}(b_{i_1}, \delta_1) \parallel h_{K_s}(b_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b_{i_\tau}, \delta_\tau)]$ without using the original data components: $(b_{i_1}, b_{i_2}, \ldots, b_{i_\tau})$. In this case, it means that either $x = h_{K_s}(b_{i_1}, \delta_1) \parallel h_{K_s}(b_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b_{i_\tau}, \delta_\tau)$ or $x \neq h_{K_s}(b_{i_1}, \delta_1) \parallel h_{K_s}(b_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b_{i_\tau}, \delta_\tau)$.

When $x \neq h_{K_s}(b_{i_1}, \delta_1) \parallel h_{K_s}(b_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b_{i_\tau}, \delta_\tau)$, it means that x is a strong collision of $h_{K_s}(b_{i_1}, \delta_1) \parallel h_{K_s}(b_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b_{i_\tau}, \delta_\tau)$. When $x = h_{K_s}(b_{i_1}, \delta_1) \parallel h_{K_s}(b_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b_{i_\tau}, \delta_\tau)$ and x is derived without accessing the original data components, $(b_{i_1}, b_{i_2}, \ldots, b_{i_\tau})$, it means that the remote client can derive $h_{K_s}(b_{i_1}, \delta_1) \parallel h_{K_s}(b_{i_2}, \delta_2) \parallel \cdots \parallel h_{K_s}(b_{i_\tau}, \delta_\tau)$ without accessing the original data components. There are two possible ways to obtain these quantities. (i) Predicate these numbers without using $(b_{i_1}, b_{i_2}, \ldots, b_{i_\tau})$. In

this case, $h_{K_s}(b_{i_\tau}, \delta_\tau)$ looks random to be predicated. The probability for this case is $1/\prod_{j=1,\ldots,\tau} |h_{K_s}(b_{i_j}, \delta_j)|$, which is negligible. (ii) Compute $h_{K_s}(b_{i_\tau}, \delta_\tau)$ from previous stored $h_{K_s}(b_{i_\tau})$ and δ_τ. In this case, essentially, the remote client needs to find a b'_{i_τ}, such that $h_{K_s}(b'_{i_\tau}) = h_{K_s}(b_{i_\tau}, \delta_\tau)$, where $b'_{i_\tau} \neq b_{i_\tau} \parallel \delta_\tau$. In other words, b'_{i_τ} is a strong collision of the cryptographic hash function $h_k(*)$. □

As mentioned earlier, there are also some important security requirements that new client deduplication solutions should meet. Here, we elaborate on these requirements and analyze and compare our proposed POF scheme with two other typical schemes POW [3] and PDP [9,10] in terms of the security requirements they have met.

First of all, when the server asks the client for the POW of the original file, the indices of the original file as the challenge content should be randomly generated so that the client cannot predict the requested blocks and forge or prepare the proof in advance. We call this security requirement as *random index*.

Secondly, when the client generates the POW, the original file blocks should get involved in calculating the ownership proof in each challenge sent from the server. In this way, the client cannot cheat the server by providing just a small piece of information about the original file, for example a hash value, to pass the challenge test. We call this security requirement as *calculated with original file*.

Thirdly, when the server and client carry out the proof protocol, the generated proof in each challenge should be totally different from any other proof. In other words, in each challenge, a unique and 'fresh' proof should be generated and checked to determine whether the client can pass the challenge test. So, this mechanism can be used to protect the proof schemes against the replay attack. We call this security requirement as *dynamic proof*.

The comparison results about the security characteristics mentioned earlier between these schemes are depicted in Table I. Although the PDP scheme has similar security characteristics to our proposed POF scheme, the input file in the PDP scheme must be pre-processed by the client to embed some secrets into it, which is radically not applicable in the scenario of client-side deduplication. Therefore, our proposed POF scheme not only is provably secure but also satisfies these important security requirements with advantages over the other two typical schemes POW and PDP.

5.2. Detection probability analysis

Next, we will analyze the guarantee that our proposed POF scheme offers.

On one hand, we pay attention to the probability that a client succeeds in generating the POW of the original file to the server.

Suppose that the client claims that it stores an f-block file, out of which it may miss x blocks, and the server S will ask for c blocks in one challenge intending to detect the client's misbehavior. Let X be a discrete random variable used to represent the number of missed blocks that have been detected; let P_x be the probability that at least one missed block has been detected, and let $(1 - P_x)$ be the probability that no missed block has been detected. So, we have

$$\begin{aligned} P_x &= P\{x \geq 1\} \\ &= 1 - \frac{\binom{f-x}{c}}{\binom{f}{c}} \\ &= 1 - \prod_{i=0}^{c-1} \frac{f-x-i}{f-i} \end{aligned} \tag{1}$$

It follows that

$$\left(1 - \frac{x}{f-(c-1)}\right)^c \leq 1 - P_x \leq \left(1 - \frac{x}{f}\right)^c \tag{2}$$

Because $c - 1 \ll f$, the difference between the left-hand side and the right-hand side is very small. Therefore, we have

$$1 - P_x \approx \left(1 - \frac{x}{f}\right)^c, \text{ that is, } P_x \approx 1 - \left(1 - \frac{x}{f}\right)^c \tag{3}$$

In this way, we can obtain the approximate minimum c required for each challenge to detect at least one missed block, which can be expressed as

$$c = \left\lceil \log_{\left(1-\frac{x}{f}\right)} (1 - P_x) \right\rceil \tag{4}$$

First, we fix the number of the file blocks f and show P_x as a function of c for four values of x (x = 1%, 5%, 10% and 15% of f), which is shown in Figures 2 and 3.

To achieve high detection probability, for example, P_x = 99%, the server S has to request 315, 83, 42 and 28 blocks in one challenge respectively for x = 1%, 5%, 10% and

Table I. Security requirements and comparison.

Scheme	Random index	Calculated with original file	Dynamic proof
Provable ownership of files	√	√	√
Proof of ownership	√	NONE	NONE
Proof of data possession	√	√	√

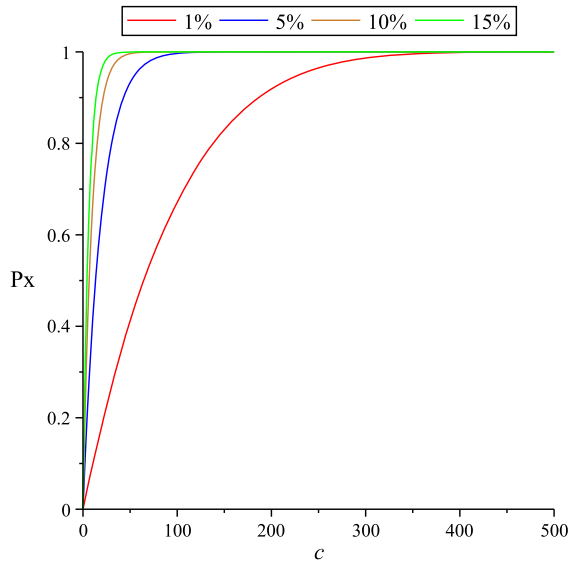

Figure 2. The detection probability $P_x(f = 1000)$.

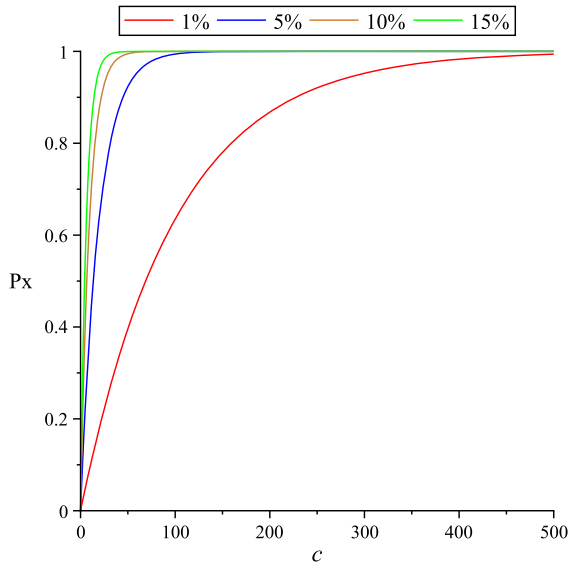

Figure 3. The detection probability $P_x(f = 30\,000)$.

15% of f, where $f = 1000$ blocks. When $f = 30\,000$ blocks (a typical DVD file contains 30-GB data and could be divided into 30 000 blocks with 1 MB in one block), the number of blocks that the server S should request to achieve the same detection probability is 452, 90, 44 and 29 blocks, respectively, in one challenge.

By fixing the detection probability P_x, it can be seen that the increase of the number of missed blocks x will make the number of requested blocks c in one challenge decrease rapidly. At the same time, it also can be seen that when the number of missed blocks x is relatively small (e.g., $x \leq 1\%$), the increase of the number of the entire file blocks has a considerable impact on the number of blocks requested in one challenge. But when the number of missed blocks x is relatively large (e.g. $x \geq 15\%$), the increase of f has a marginal impact on c.

Next, we fix the number of missed blocks x ($x = 5\%$ of f) and show P_x as a function of c in four different values of f ($f = 1000$, 3000, 5000 and 30 000). Then, we fix $x = 15\%$ of f and draw the curve of P_x as a function of c in the same figure, which is shown in Figure 4.

From Figure 4, we can see that with the fixed number of missed blocks x, the increase of the number of the file blocks has a minimal impact on the function relationship between the detection probability P_x and the number of blocks c verified in one challenge. But by fixing the number of the entire file blocks f, it can be seen that the increase of x will decrease the c rapidly, which is the same as the results from Figures 2 and 3.

On the other hand, in a typical client-side deduplication scenario, if a client possesses majority of the original file, for example, 95% of the original file, we consider that it is enough to tell that the client has full ownership of the original file although some bits of the original file are missed by the client. This is because, in many situations, it is not necessary to demand that the client must have every bit of the original file exactly. For example, maybe the client wants to claim ownership of a movie DVD with different subtitles or a slightly different version of a free software. So, next we will study the relationship between the detection probability and the number of times of challenge under two conditions.

(1) Only once will the server S challenge the client C to ask for the POW of the original file.

Suppose the result of the challenge is that none of the missed blocks has been detected. In this case, we want to calculate how many blocks the server S has to request in a one-time challenge to ensure that the client C possesses at least 95% data of the original file (i.e., the number of missed blocks by the client $x \leq 5\%$ of f) with a high detection probability $P_x = 99\%$. The reason for choosing a relatively high detection probability ($P_x \geq 99\%$) is that it would guarantee more accurately that the client possesses at least 95% data of the original file. Because the amount of the blocks of the entire file has a minimal impact, we might as well fix $f = 30\,000$ blocks.

Let $x = 5\%$ of f, $P_x = 99\%$; we substitute these concrete values into formula (1). It follows that $c = \lceil 89.781 \rceil$, so $c = 90$ blocks. So, it can be concluded that the server S challenges the client C only one time for the POW of the original file. By fixing $f = 30\,000$, if the requested blocks in the challenge are equal to or greater than 90 blocks, then it can be ensured that the client C possesses at least 95% data of the original file with a high detection probability of $P_x = 99\%$. Actually, a more visualized

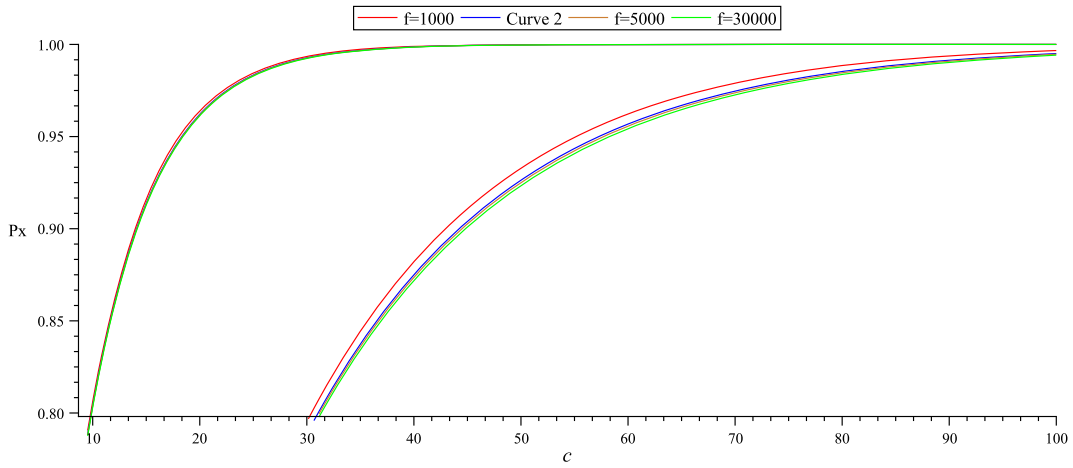

Figure 4. The detection probability P_x ($x = 5\%$ and 15% of f).

conclusion can be drawn from Figure 3, in which the blue curve represents the case of $x \leq 5\%$ of f, when $f = 30\,000$. So, all the corresponding c values represented by points (P_x, c) in the area below the blue curve can ensure that the number of missed blocks is $x \leq 5\%$ of f with different detection probabilities.

(2) The server S will challenge the client C totally T times to ask for the POW of the original file.

In the T challenges, suppose that there are γ times ($\gamma \leq T$) in which none of the missed blocks has been detected and $T - \gamma$ times in which at least one missed block has been detected. These two kinds of possible results are called "failure" and "success", respectively. Then, we want to know what probability of γ failures in T challenges can ensure that the number of missed blocks is $x \leq 5\%$ of f. In other words, what probability of γ out of T can give us enough confidence to tell that the client C possesses majority of the original file (95% data of the original file), that is, the client approximately possesses all of the original file.

For simplicity, considering T challenges as independent Bernoulli experiments of T times, the experiment result is either a failure, whose probability that no missed block has been detected is represented by $(1 - P_x)$, or a success, whose probability that at least one missed block has been detected is represented by P_x. Let Y be the discrete random variable defined as the number of times of failure and P_y be the probability that there are γ times in which none of the missed blocks has been detected out of the total T challenges. So, we have

$$\begin{aligned} P_y &= P\{Y = \gamma\} \\ &= \binom{T}{\gamma}(1 - P_x)^{\gamma}(P_x)^{T-\gamma} \\ &= \frac{T!}{\gamma!(T-\gamma)!}\left[(1-\lambda)^c\right]^{\gamma}\left(1-(1-\lambda)^c\right)^{T-\gamma} \end{aligned} \tag{5}$$

where let $\lambda = x/f$.

Assume that T, γ and c are all constant, it can be seen that P_y is a decreasing function of independent variable λ.

Example 1. Let $f = 30\,000$, $x = 5\%f$, $c = 50$, $T = 10$ and $\gamma = 2$, we have

$$\begin{aligned} P_y &\approx \frac{10!}{2!\times 8!}\left[\left(1-\frac{1500}{30\,000}\right)^{50}\right]^2\left[1-\left(1-\frac{1500}{30\,000}\right)^{50}\right]^8 \\ &= 0.1404 = 14.4\% \end{aligned}$$

Example 2. Let $f = 30\,000$, $x = 5\%f$, $c = 30$, $T = 10$ and $\gamma = 2$, we have

$$\begin{aligned} P_y &\approx \frac{10!}{2!\times 8!}\left[\left(1-\frac{1500}{30\,000}\right)^{30}\right]^2\left[1-\left(1-\frac{1500}{30\,000}\right)^{30}\right]^8 \\ &= 0.3000 = 30.00\% \end{aligned}$$

So, from the preceding analysis, we can conclude that the server S challenges the client C 10 times for the POW of the original file. If the probability of two failures in 10 challenges is equal to or greater than 14.04%, then this result can ensure that the number of missed blocks is $x \leq 5\%$ of f; in other words, the client approximately has the full ownership of the original file. From the preceding formula, different results could be drawn from different initial conditions.

6. PERFORMANCE EVALUATION

First of all, we carry out a theoretical analysis of our scheme's performance and compare it with two other typical schemes: POW in remote storage systems [3] and PDP

DOI: 10.1002/sec

[10]. Then, a practical simulation will be implemented to evaluate the performance of these schemes.

6.1. Theoretical analysis

6.1.1. Phase 1 setup.

In our scheme of POF, to derive a new session key from the pre-shared key between the server and the client, both of them will complete the corresponding computation process. During this phase, the server will carry out the key generation algorithm once and then the cryptographic hash function once. The client will do the same thing. The key generation algorithm in our scheme has the same computational complexity as a hash function. The computational complexity of cryptographic hash functions is $\mathcal{O}(r^*u) = \mathcal{O}(\log(r)\log(u))$ if the hash function is like $\{0,1\}^{\log(r)} \rightarrow \{0,1\}^{\log(u)}$. So the whole cost will be $2\mathcal{O}(\log(r)\log(u))$ for the server and the client.

In the scheme of POW, the server first carries out a reduction phase in which the f-block file will be reduced into a buffer of l blocks. In this reduction stage, each block from the file is XORed to a constant number of random locations in the buffer and will compute initial variables by using a cryptographic hash function. The XOR's computational complexity is $\mathcal{O}(w)$ in case of two w-digit numbers. So the computational complexity of this stage is $f[\mathcal{O}(w) + \mathcal{O}(\log(r)\log(u))]$. Next, the server will run a mixing process on the buffer of l blocks, the computational complexity of which is $5l\mathcal{O}(w)$. Last, to build a Merkle tree on the compacted l blocks, the server has to compute $l + (1 + 2 + \cdots + (l/2)) = l^2 + 10l/8$ hash functions to prepare the values of all the nodes. At the same time, the client will do the same thing, except for building the Merkle tree. So the whole cost is $f[\mathcal{O}(w) + \mathcal{O}(\log(r)\log(u))] + 5l\mathcal{O}(w) + (l^2 + 10l/8)\mathcal{O}(\log(r)\log(u))$ for the server and $f[\mathcal{O}(w) + \mathcal{O}(\log(r)\log(u))] + 5l\mathcal{O}(w)$ for the client.

In the scheme of PDP, with the server doing nothing in this phase, the client encodes the entire file by using the Reed–Solomon with a computational cost of f^*2t if the parameters of the Reed–Solomon code is $(f, f-t, t)$ [17]. Furthermore, the client needs to compute the authenticator (TagBlock) for every encoded blocks of the file. The computation of generating the "TagBlock" on one block involves the modular exponentiation once and then the cryptographic hash function once. The computational cost of the modular exponentiation is $\mathcal{O}(2^e d^2)$ in case of two d-digit numbers and an e-bit exponent. So the whole cost of the client is $2ft + f\left[\mathcal{O}\left(2^e d^2\right) + \mathcal{O}\left(\log(r)\log(u)\right)\right]$.

6.1.2. Phase 2 challenge.

In our scheme of POF, the server needs to compute two temporary keys and check the POW of the file sent from the client, which includes $2+(c+1)$ hash functions in total. So the cost will be $(c+3)\mathcal{O}(\log(r)\log(u))$. At the same time, the client will compute the POW of the file at the same cost of $(c+3)\mathcal{O}(\log(r)\log(u))$.

In the scheme of POW, the server also has to choose c leaves of the Merkle tree as requested blocks to challenge the client and check the correctness of the proof after the client sends the proof back, the cost of which is rather low and can be ignored. On the other hand, to answer the challenge sent from the server, the client needs to build a Merkle tree on the compacted l blocks, the cost of which is $[(l^2 + 10l)/8]\mathcal{O}(\log(r)\log(u))$.

In the scheme of PDP, the server, to generate an effective proof, has to compute $c+1$ hash functions, $c+1$ exponentiations and two modular exponentiations. Suppose the size of each block is n. So the total computational complexity of the server is $(c+1)\mathcal{O}(\log(r)\log(u)) + (c+1)2^{\mathcal{O}(n)} + 2\mathcal{O}(2^e d^2)$. On the other hand, the client's computation includes $c+1$ hash functions, c exponentiations and one modular exponentiation. So the total computational complexity of the client is $(c+1)\mathcal{O}(\log(r)\log(u)) + c2^{\mathcal{O}(n)} + \mathcal{O}(2^e d^2)$.

Table II summarizes the computational complexity of the three schemes in different phases.

In our scheme, it is the client that proves to the server that it indeed possesses the original file. Because the client usually has less computation capability and storage capacity, we mainly focus on the computational complexity of the client. From the results in Table II, we can see that, during the setup phase, only our scheme's computational complexity has no relation with the number of the file blocks, which is fixed to the computation of two hash func-

Table II. Theoretical analysis and comparison of performance.

Scheme		Server	Client
Provable ownership of file	Setup	$2\mathcal{O}(\log(r)\log(u))$	$2\mathcal{O}(\log(r)\log(u))$
	Challenge	$(c+3)\mathcal{O}(\log(r)\log(u))$	$(c+3)\mathcal{O}(\log(r)\log(u))$
Proof of ownership	Setup	$f[\mathcal{O}(w) + \mathcal{O}(\log(r)\log(u))] + 5l\mathcal{O}(w) + [(l^2 + 10l)/8]\mathcal{O}(\log(r)\log(u))$	$f[\mathcal{O}(w) + \mathcal{O}(\log(r)\log(u))] + 5l\mathcal{O}(w)$
	Challenge	0	$[(l^2 + 10l)/8][\mathcal{O}(\log(r)\log(u))]$
Proof of data possession	Setup	0	$2ft + f[\mathcal{O}(2^e d^2) + \mathcal{O}(\log(r)\log(u))]$
	Challenge	$(c+1)\mathcal{O}(\log(r)\log(u)) + (c+1)2^{\mathcal{O}(n)} + 2\mathcal{O}(2^e d^2)$	$(c+1)\mathcal{O}(\log(r)\log(u)) + c2^{\mathcal{O}(n)} + \mathcal{O}(2^e d^2)$

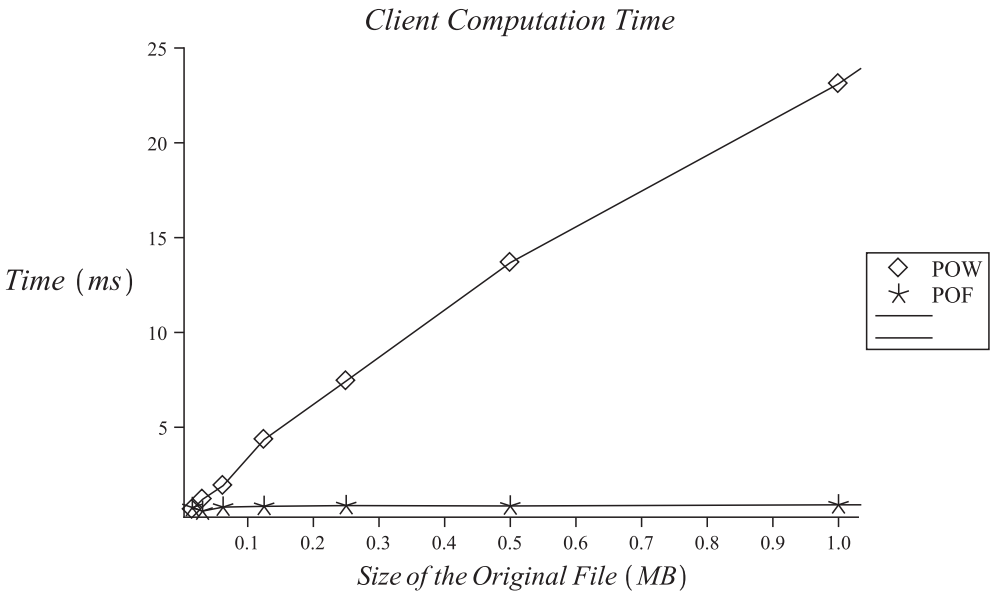

Figure 5. Client computation time (provable ownership of file (POF) and proof of ownership (POW)).

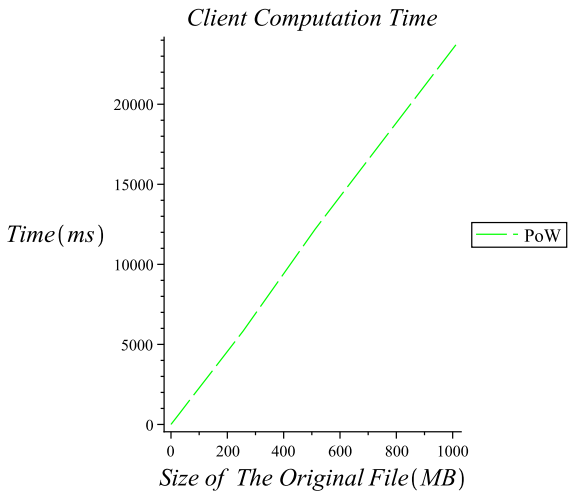

Figure 6. Client computation time (proof of ownership, POW).

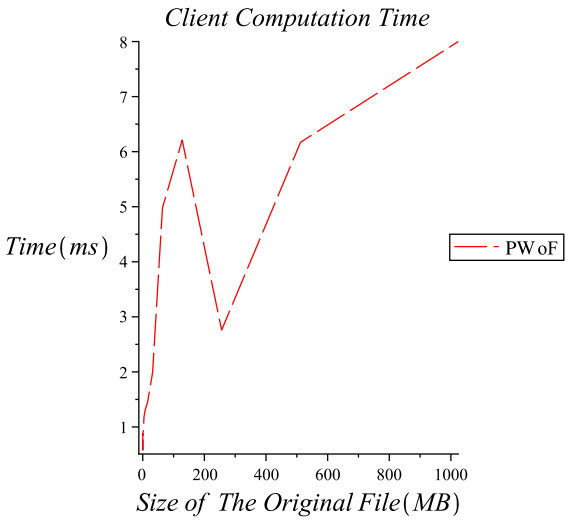

Figure 7. Client computation time (provable ownership of file, PWOF).

tions. This a big advantage over the other two schemes. During the challenge phase, the client in our scheme computes a hash function $c+1$ times, but the client in POW has to compute the hash function $(l^2 + 10l)/8$ times, which is even less than that of the PDP. At the same time, although the size l of compacted blocks in POW is constant, the requested blocks c are much less than the size of l and only change slightly with a fixed ratio of x to f. In summary, our scheme has a great advantage over the two other typical schemes in terms of performance.

6.2. Simulation analysis

We implement our proposed POF scheme and implement the typical scheme of POW [3] as a basis for comparison to measure their performance. We ran the protocols on random files of sizes 16 KB–1 GB. The experiments were conducted on an Intel 3.0-GHz Intel Core 2 Duo system with 64-KB cache, 1333-MHz EPCI bus and 2048-MB random access memory. The system runs Ubuntu10.04, kernel version 2.6.34. We used C++ for the implementation. We also used SHA256 from Crypto++ version 0.9.8b [18]. The files are stored on an ext4 file system

DOI: 10.1002/sec

Table III. Performance measurements and comparison (client computation time).

	Proof of ownership			Provable ownership of file		
Size (MB)	Disk read (ms)	Merkle tree (ms)	Total (ms)	Disk read (ms)	Algorithm 1 (ms)	Total (ms)
0.015625	0.09	0.57	0.66	0.15	0.62	0.77
0.03125	0.13	1.07	1.2	0.16	0.42	0.58
0.0625	0.19	1.73	1.92	0.17	0.62	0.79
0.125	0.34	4.01	4.35	0.20	0.63	0.83
0.25	0.62	6.82	7.44	0.24	0.63	0.87
0.5	1.17	12.51	13.68	0.27	0.58	0.85
1	2.03	21.08	23.11	0.29	0.62	0.91
2	4.44	42.46	46.90	0.31	0.62	0.93
4	8.19	84.46	92.65	0.55	0.63	1.18
8	14.76	168.43	183.19	0.66	0.65	1.31
16	28.62	334.88	363.50	0.82	0.64	1.46
32	56.75	669.38	726.13	1.32	0.67	1.99
64	112.58	1352.01	1464.59	4.34	0.64	4.98
128	223.08	2692.07	2915.15	5.56	0.65	6.21
256	437.84	5393.64	5831.48	2.11	0.65	2.76
512	1269.46	10 932.49	12 201.95	5.53	0.64	6.17
1024	2581.56	23 344.83	25 926.39	5.52	0.63	6.15

on a Seagate Barracuda 7200.7 (ST23250310AS) 250-GB Ultra ATA/100 drive. All experimental results represent the mean of 10 trials. These results are depicted in Figures 5–7, and the full set of numbers is given in Table III.

6.2.1. Client computation time.

For our proposed POF scheme, the client computation time includes the following: (i) the time to read the requested portions of the original file from the disk according to the randomly chosen indices given by the server; (ii) the time used to compute their SHA256 values; and (iii) the time to execute Algorithm 2 for POF. For the POW scheme, the client computation time includes the following: (i) the time to read the whole file from the disk; (ii) the time used to compute the SHA256 value; and (iii) the time to perform the reducing and mixing and finally to compute the Merkle tree over the resulting buffer [3]. For simplicity, the implementation of POW in this paper only includes the time used to compute the Merkle tree over the original file. Nevertheless, our scheme costs less time than the POW.

For the POF, the measurements show that as the size of the whole file increases, the time of reading the requested portions of the original file from the disk increases accordingly and fluctuates considerably sometimes. This is because the requested portions of the original file is randomly distributed on the disk. At the same time, we assume that the number of missed blocks for a client is at most 5%, whereas the detection rate is at least 99% in this simulation. It can be found that the number of blocks requested by the server is small as compared with the whole number of blocks of the file. And the required number of blocks can remain about the same regardless of the size of the file. In this way, we can keep the number of the blocks fixed in Algorithm 2, which will make the running time of POF comparatively short and with little variation.

For the POW, the measurements show that both the disk reading time and Merkle Tree building time increase linearly with the size of the original file. The time increase is significant especially for relatively large files (e.g., $\geq$64 MB). Moreover, building the Merkle tree takes a much longer time than running Algorithm 1. This is because the client in scheme POW has to build a Merkle tree on all blocks of the whole file to answer the challenges from the server.

6.2.2. Server computation time.

For our POF scheme, the server computation time is approximately the same as or shorter than the client computation time, which is comparatively short as a whole. For the POW scheme, this is the time spent by the server to check the Merkle tree authentication signatures. The overhead of these verifications is very low, so the time duration is also short for the server.

6.2.3. Network transmission time.

This is the estimated time required to transfer the protocol data. The overall size of the message transmitted by the protocol is less than 1 KB in each challenge for both schemes. Thus, the network transmission time is also negligible.

From the measurements earlier, it can be seen that the simulation results are consistent with the theoretical analysis on the performance of the POF and POW. Therefore, it can be concluded that our proposed POF exceeds the other two schemes in performance.

7. CONCLUSIONS

The new technology of client-side deduplication can save the bandwidth of uploading copies of existing files to the server in cloud storage services. However, this technology introduces some new security problems, which have not been well understood. In this paper, we propose a cryptographically secure and efficient scheme, called a POF, in which a client proves to the server that it indeed possesses the entire file on the basis of partial information about it. The aim of the scheme is to solve the problem that by learning just a small piece of information about the file, namely its hash value, an attacker is able to obtain the entire file from the server in the client deduplication scenario. We achieve the efficient goal by relying on dynamic spot checking, in which the client could only access small portions of the original file to generate the proof of possessing the original file correctly and integrally, thus greatly reducing the burden of computation on the client and minimizing the I/O between the client and the server, while maintaining a high detection probability of the misbehavior of clients. At the same time, by utilizing dynamic coefficients and randomly chosen indices of the original files, our scheme mixes the randomly sampled portions of the original file with the dynamic coefficients together to generate the unique proof in every challenge. This technique helps us meet the key security requirements and makes it infeasible for the cheating remote client, without possessing a specific file, to convince the remote server that it has ownership of the file. Finally, rigorous security proof and extensive performance simulation and analysis are conducted, and the results show that the proposed POF scheme is not only provably secure but also highly efficient, especially in reducing the burden of the client.

REFERENCES

1. Dutch M, Freeman L. Understanding data deduplication ratios. SNIA, February 2009. Available from: http://www.snia.org/.
2. Harnik D, Pinkas B, Shulman-Peleg A. Side channels in cloud services, the case of deduplication in cloud storage. *IEEE Security and Privacy Magazine, Special Issue of Cloud Security* 2010; **8**: 40–47.
3. Halevi S, Harnik D, Pinkas B, Shulman-Peleg A. Proofs of ownership in remote storage systems. In *ACM CCS '11: ACM Conference on Computer and Communications Security*. ACM(Association for Computing Machinery): New York, NY, USA, 2011; 491–500.
4. Dropbox Cloud Service. Available from: https://www.dropbox.com.
5. Mulazzani M, Schrittwieser S, Leithner M, Huber M, Weippl E. Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In *USENIX Security*. USENIX Association: Berkeley, CA, USA, 2011; 5–15.
6. Juels A, Kaliski BS, Jr. PoRs: proofs of retrievability for large files. In *ACM CCS '07*. ACM, 2007; 584–597.
7. Shacham H, Waters B. Compact proofs of retrievability. In *ASIACRYPT '08*. Springer-Verlag, 2008; 90–107.
8. Wang Q, Wang C, Li J, Ren K, Lou W. Enabling public verifiability and data dynamics for storage security in cloud computing. In *ESORICS'09*. Springer-Verlag, 2009; 355–370.
9. Ateniese G, Burns R, Curtmola R, *et al*. Provable data possession at untrusted stores. In *ACM CCS'07*. ACM, 2007; 598–609.
10. Ateniese G, Burns R, Curtmola, R, *et al*. Remote data checking using provable data possession. *ACM Transactions on Information and System Security (TISSEC)* 2011; **14**(1): 12:01 Ű 12:34.
11. Ergün F, Kannan S, Ravi Kumar S, Rubinfeld R, Viswanathan M. Spot-checkers, *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, Dallas, TX, May 24–26, 1998; 259–268.
12. Bowers K, Juels A, Opera A. Hail: a high-availability and integrity layer for cloud storage. In *ACM CCS '09*. ACM, 2009; 187–198.
13. Bowers KD, Juels A, Oprea A. Proofs of retrievability: theory and implementation. *Cryptology ePrint Archive, Report 2008/175*, 2008. Available from: http://eprint.iacr.org/.
14. Curtmola R, Khan O, Burns R, Ateniese G. MR-PDP: Multiple-replica provable data possession, *Proceedings of the ICDCS '08*, Beijing, China, 2008; 411–420.
15. Elbaz R, Champagne D, Gebotys C, Lee RB, Potlapally N, Torres L. Hardware mechanisms for memory authentication: a survey of existing techniques and engines. In *ToCS IV, LNCS*. IEEE Computer Society: Washington, DC, USA, 2009; 1–22.
16. Merkle RC. A certified digital signature. In *Proceedings on Advances in Cryptology, CRYPTO'89*. Springer-Verlag, Inc.: New York, NY, 1989; 218–238.
17. Biard L, Noguet D. Reed–Solomon codes for low power communications. *Journal of Communications* 2008; **3**(2): 13–21.
18. Dai W. Crypto++ Library, 5.6.1, January 2011. Available from: http://www.cryptopp.com/.

DOI: 10.1002/sec