Provable Ownership of File in De-duplication Cloud Storage

Chao Yang[†‡], Jian Ren[‡] and Jianfeng Ma[†]
[†]School of CS, Xidian University
Xi'an, Shaanxi, 710071. Email: {chaoyang, jfma}@mail.xidian.edu.cn
[‡]Department of ECE, Michigan State University
East Lansing, MI 48824. Email: {chaoyang, renjian}@msu.edu

***Abstract*—The rapid adoption of cloud services has propelled network data sharing and storage. Client-side deduplication is proposed to minimize bandwidth and space needed to upload and store duplicated data. The existing solutions, however, were recently found to be vulnerable to attacks that enable the attackers to get full access to the entire file stored on the server after learning just a small piece of information about the file, namely its hash value. In this paper, to solve the problem mentioned above, we propose a cryptographically secure and efficient scheme for a client to prove to the server based on actual possession of the entire 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 (POF) and maintain a high detection probability of the 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.**

***Index Terms*—Cloud storage, deduplication, provable ownership, spot-checking**

I. Introduction

Data management through cloud is being viewed as a technique that can save the cost for data sharing and management. A key concept for remote data storage is client-side deduplication, in which the server stores only a single copy of each file, regardless of how many clients need to store that file. That is only the first client needs to upload the file to the server. This design will save both the communication bandwidth as well as the storage capacity. It is reported that business applications can achieve deduplication ratios from 1:10 to as much as 1:500, which will result in disk and bandwidth savings of more than 90% [1].

However, the client-side deduplication introduces some new security problems. Harnik et al. [2] 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 on the server and the file may even contain sensitive information [2]. 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 get the entire file from the server. Similar attacks have been implemented against Dropbox by Mulazzani et al. [4] 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 file. An attacker that learns it can get access to the entire file.

Recently, the importance of ensuring the remote data integrity has been highlighted by works of the proofs of retrievability (PoR) [5], [6] and proofs of data possession (PDP) [7], [8]. 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 has been a total role reversal between the server and the client. However, this role reversal is significant, especially the PoR and PDP protocols in the literature are inapplicable to this scenarios. 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 of solutions to this new problem have been proposed [3], [4]. Unfortunately, these solutions either cannot meet provably security requirements due to static spot checking [9] or have a relatively high computational complexity. As a result, their applicability in the scenario of client-side deduplication are greatly limited.

In this paper, to solve the problem in the scenario of the client-side deduplication mentioned above, we propose a cryptographically secure and efficient scheme, called a provable ownership of the file (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, 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 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 the unique proof in every challenge. This technique guarantees the provable-security goal.

The rest of the paper is organized as follows: Section II introduces the models and notations. We present the detailed description of our scheme in Section III. Section IV and section V provide the security analysis and performance evaluations respectively, followed by Section VI that overviews

978-1-4799-1353-4/13/$31.00 ©2013 IEEE

the related work. Finally, Section VII concludes the paper.

II. Preliminaries

A. System Model and Adversary Model

In the client deduplication scenario, the storage 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 on the storage server. Specifically, a client user first sends the hash value of the original file 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 on the server, the server will challenge the client to ask for the proof of possession of the original file. Upon a successful challenge, 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 the communication bandwidth as well as the storage space.

We consider three kinds of security threats. First, the deduplication system may use a standard hash function. e.g. SHA256. In that case, it is probable that an attacker could get hash values of files owned by other clients who may have published the hash value as the signature of the file or use the hash value elsewhere. Second, taking strong adversary into consideration, an attacker may be able to compromise a server temporarily to get access to its stored data, which includes the hash values for the files stored on it. Getting the proxy of a file, the attacker can download the original file, which may contain confidential data or information. Third, it is plausible that similar attacks would happen on the client side.

B. Notation

- F - the original file;
- f - the number of blocks that the original file is divided and stored;
- $(b_1, \cdots, b_f)$ - all the blocks of the original file;
- α - a pseudo-random function (PRF) defined as $\alpha : \{0,1\}^* \times key \rightarrow \{0,1\}^\mu$, where μ is a security parameter;
- β - a pseudo-random permutation (PRP) defined as $\beta : \{0,1\}^q \times key \rightarrow \{0,1\}^q$;
- 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 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.

III. Provable Ownership of the File in Deduplication

A. Definitions

Definition 1 (Provable Ownership of the File (POF))**.** *A POF scheme is a collection of three polynomial-time algorithms (KeyDeriving, ProofGen, ProofCheck) such that:*

KeyDeriving $(sk, R_c) \rightarrow \{K_s, S_1, S_2\}$ *is a key generation algorithm that is run by the server to setup the scheme. It takes as input the symmetrical key shared between a client and a storage server* sk *and a random number* R_c*. It returns a new session key and two random numbers:* (K_s, S_1, S_2)*;*

ProofGen $(K_s, F, Chal) \rightarrow V$ *is run by the client in order to generate a POF. 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 POF* V *based on the blocks in the file* F *which are determined by the challenge set* $Chal$*.*

ProofCheck $(K_s, Chal, F, V) \rightarrow \{'True', 'False'\}$ *is run by the server in order to verify that the client does have 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* $'True'$ *or* $'False'$ *whether the* V *is a correct POF based on the blocks determined by* $Chal$*.*

B. Efficient and Secure POF Scheme

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

For each challenge, the proof V will be computed with different and fresh coefficients δ_τ, which are determined by a PRF 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 (MAC), 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.

IV. Security Analysis

One of the major design goal is to prevent the client from producing an authenticator for the remote server by simply accessing a fingerprint of the original file to pass the verification sent from the remote server. Our proposed scheme and protocol make it computationally infeasible to pass the verification without accessing the original 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 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: $(b_{i_1}, b_{i_2}, \cdots b_{i_\tau})$. In this

Algorithm 1 A Provable Ownership of Files Scheme (POF)

KeyDeriving(sk, R_c):

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

ProofGen$(K_s, F^{'}, Chal)$:

1: Let $F^{'} = (b_1^{'}, b_2^{'}..., b_f^{'})$ 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 $F = (b_1, b_2..., b_f)$, $V = (H^{'})$, 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'.

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: $(b_{i_1}, b_{i_2}, \cdots b_{i_\tau})$, 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 get these quantities: i) predicate these numbers without using $(b_{i_1}, b_{i_2}, \cdots b_{i_\tau})$. In this case $h_{K_s}(b_{i_\tau}, \delta_\tau)$ looks random to be predicated. The probability for this case is $1/\prod_{j=1,\cdots,\tau} |h_{K_s}(b_{i_j}, \delta_j)|$, 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 an $b_{i_\tau}^{'}$, 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, $b_{i_\tau}^{'}$ is a strong collision of the cryptographic hash function $h_k(*)$. ■

As mentioned above, there are also some important security requirements which new client-deduplication solutions should meet. Here we elaborate on these requirements, analyze and compare our proposed POF scheme with other two typical

Algorithm 2 A Protocol of Provable Ownership of Files Scheme (POF Protocol)

Setup:

1: The server S decomposes the file F into f blocks, $b_1, ...b_f$ 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 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 seeds output and send back $h_{K_s}(R_c, TS) \| TS$ in order 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 the ownership of a specific file stored at the server and requests to start a proof process;
2: The server S prepares to get the proof of ownership for c distinct blocks of the specific file F, where $1 \le c \le f$;
3: The server S forms the $(c, S_1, S_2) = Chal$, in order to determine the indices of c distinct blocks it wants to challenge and the two random seeds S_1, 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$(K_s, F' = (b_1^{'}, b_2^{'}..., b_f^{'}), Chal(c, S_1, S_2)) \rightarrow V$ and sends back to S the proof of ownership of the original file V;
5: The server S set $(c, S_1, S_2) = Chal$ and checks the validity of the proof V by running ProofCheck$(K_s, Chal, F, V)$.

scheme POW [3] and PDP [7], [8] in terms of the security requirements they have met.

First of all, when the server asks the client for the proof of ownership 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 proof of the ownership, 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 above between these schemes are depicted in Table I. Although the PDP scheme has similar security characteristics as our proposed POF scheme, the input file in 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 is not only provably secure but also satisfies these important security requirements with the advantages over the other two typical schemes POW and PDP.

V. Performance Evaluation

A. Theoretical analysis

We first provide a theoretical analysis on the performance of the proposed scheme. We then compare it with other two typical schemes: proofs of ownership in remote storage systems (POW) [3] and PDP [8]. 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. Since the client usually has less computational capability and storage capacity, so 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 functions. This is a big advantage over the other two schemes. During the challenge phase, the client in our scheme computes hash functions of $c+1$ times, while the client in POW has to compute hash function of $(\frac{l^2+10l}{8})$ times, though the computational complexity of this scheme is less than the PDP. Furthermore, the requested blocks c will only change slightly with the fixed ratio of x to f, and the size l of compacted blocks in POW is also constant. In this way, our scheme is more efficient than the POW and PDP in the challenge phase. In summary, our scheme has a great advantage over other two typical schemes in terms of performance.

B. Simulation Results

We implement our proposed POF and a typical scheme of POW [3] as a basis for comparison to measure their performance. We ran the protocols on random files of sizes from 16KByte through 1GByte. The experiments were conducted on an Intel 3.0GHz Intel Core 2 Duo system with 64KB cache, 1333MHz EPCI bus, and 2048MB of RAM. The system runs Ubuntu10.04, kernel version 2.6.34. We used C++ for the implementation. We also used the SHA256 from Crypto++ version 0.9.8b [10]. The files are stored on an ext4 file system on a Seagate Barracuda 7200.7 (ST23250310AS) 250GB Ultra ATA/100 drive. All experimental results represent the mean of 10 trials. These results are depicted in Figure 1 and the full set of numbers are given in Table III.

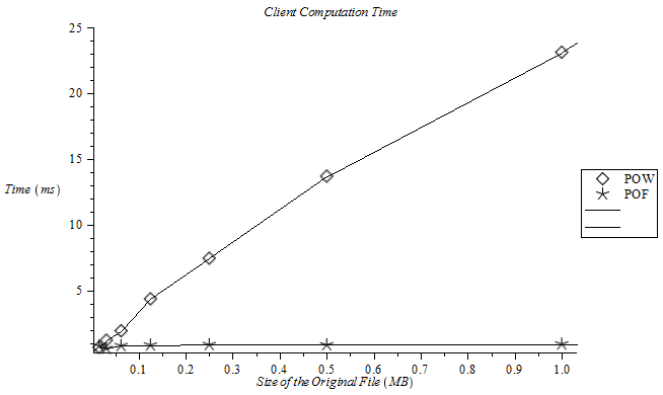

Fig. 1. Client Computation Time (PoW & POF)

Client Computation Time: For our proposed POF scheme, the client computation time includes (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 presented above for POF. For the POW scheme, the client computation time includes (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 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 the number of missed blocks for a client is at most 5% , while 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 to 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 of 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., $\geq 64MB$). Moreover, building the Merkle Tree takes 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 in order to answer the challenges from the server.

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 verification is very low, so

TABLE I
SECURITY REQUIREMENTS AND COMPARISON

Scheme	Random Index	Calculated With Original File	Dynamic Proof
POF	√	√	√
POW	√	NONE	NONE
PDP	√	√	√

TABLE II
THEORETICAL ANALYSIS AND COMPARISON OF PERFORMANCE, WHERE r AND u ARE THE LENGTH OF INPUT AND OUTPUT OF A HASH FUNCTION, RESPECTIVELY; w IS THE LENGTH OF INPUT OF THE XOR; l IS THE NUMBER OF COMPACTED BLOCKS USED TO BUILD A MERKLE TREE; t IS THE PARAMETER OF A RS-CODE; e AND d IS THE PARAMETER OF THE MODULAR EXPONENTIATION COMPUTATION; n IS THE PARAMETER OF EXPONENTIATION COMPUTATION.

Scheme	**Server**	**Client**
POF	Setup: $2\mathcal{O}(log(r)log(u))$ Challenge: $(c+3)\mathcal{O}(log(r)log(u))$	Setup: $2\mathcal{O}(log(r)log(u))$ Challenge: $(c+3)\mathcal{O}(log(r)log(u))$
POW	Setup: $f[\mathcal{O}(w)+\mathcal{O}(log(r)log(u))]$ $+5l\mathcal{O}(w)+\frac{l^2+10l}{8}\mathcal{O}(log(r)log(u))$ Challenge: $Zero$	Setup: $f[\mathcal{O}(w)+\mathcal{O}(log(r)log(u))]+5l\mathcal{O}(w)$ Challenge: $(\frac{l^2+10l}{8})[\mathcal{O}(log(r)log(u))]$
PDP	Setup: $Zero$ Challenge: $(c+1)\mathcal{O}(log(r)log(u))$ $+(c+1)2^{\mathcal{O}(n)}+2\mathcal{O}(2^e d^2)$	Setup: $2ft+f[\mathcal{O}(2^e d^2)+\mathcal{O}(log(r)log(u))]$ Challenge: $(c+1)\mathcal{O}(log(r)log(u))$ $+c2^{\mathcal{O}(n)}+\mathcal{O}(2^e d^2)$

TABLE III
PERFORMANCE MEASUREMENTS AND COMPARISON (CLIENT COMPUTATION TIME)

Size (MB)	**PoW**			**POF**		
	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	10932.49	12201.95	5.53	0.64	6.17
1024	2581.56	23344.83	25926.39	5.52	0.63	6.15

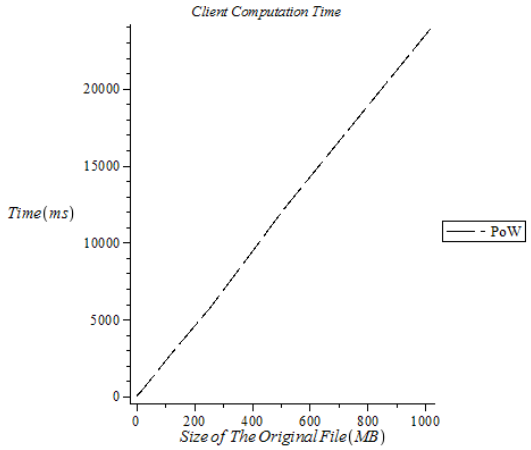

Fig. 2. Client Computation Time (PoW)

the time duration is also short for the server.

Network Transmission Time: This is an estimated time required to transfer the protocol data. The overall size of the message transmitted by the protocol is less than 1KByte in each challenge for both schemes. Thus, the network transmission time is also negligible. From the measurements above, 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.

VI. RELATED WORK

The importance of ensuring the remote data integrity has been highlighted by works of the proofs of retrievability (PoR) [5], [6] and proofs of data possession (PDP) [7], [8].

However, the PoR and PDP schemes are not applicable to our scenario since the input file for the PoR and PDP must be

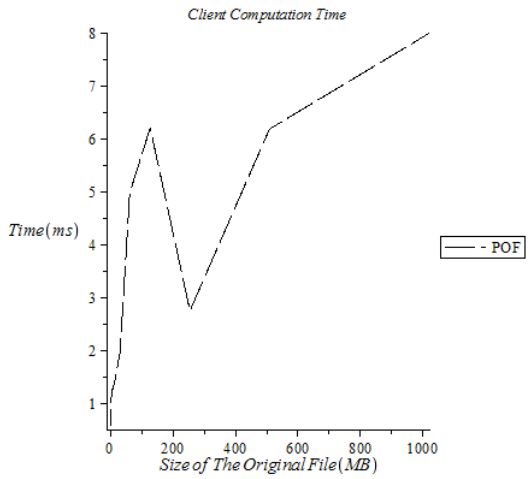

Fig. 3. Client Computation Time (POF)

pre-processed by the client to embed some secrets into it. In client-side deduplication, a new client has only the original file itself, it is impossible to insert secrets into it in advance for the purpose of proof. This excludes many similar solutions [11]–[13].

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 [2]. They suggest using encryption to address the problem, but the encryption will essentially prevent deduplication from taking place, and might therefore be unacceptable. Also, they want 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. [4] discovered a method to get the original file from the Dropbox server after acquiring only a small piece of information about the file. They also proposed a communication protocol for of the Dropbox. But their scheme has limited security features and was not well evaluated. 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 based on Merkle trees and the error-control coding 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 Merkle Tree on the encoded data, which is inherently inefficient.

Another well studied, related problem is how to verify the integrity of memory contents [14]. Many of these schemes utilize Merkle trees for memory and data authentication.

VII. Conclusion

To solve the new security problems in client-side deduplication, in this paper, we propose a cryptographically secure and efficient scheme, called a provable ownership of the file (POF), in which a client proves to the server that it indeed possesses the entire file without uploading the file. We provide rigorous security proof and extensive performance analysis. Our simulation results demonstrate that the proposed scheme is very efficient especially in reducing the burden of the client.

References

[1] M. Dutch and L. Freeman, "Understanding data de-duplication ratios," http://www.snia.org/, 2009.

[2] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services, the case of deduplication in cloud storage," *IEEE Security and Privacy Magazine*, vol. special issue of Cloud Security, pp. 40–47, 2010.

[3] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *ACM conference on Computer and communications security*. Chicago, IL, USA: ACM, OCT 2011, pp. 491–500.

[4] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, "Dark clouds on the horizon: Using cloud storage as attack vector and online slack space," in *20th USENIX conference on Security*. San Francisco, CA, USA: USENIX Association Berkeley, AUG 2011, pp. 5–5.

[5] A. Juels and J. P. B. S. Kaliski, "proofs of retrievability for large files," in *ACM conference on Computer and communications security*. Alexandria, VA, USA: ACM, OCT 2007, p. 584597.

[6] H. Shacham and B. Waters, "Compact proofs of retrievability," in *The 14th Annual International Conference on the Theory and Application of Cryptology & Information Security*. Melbourne, Australia: Springer-Verlag, DEC 2008, pp. 90–107.

[7] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *ACM conference on Computer and communications security*. Alexandria, VA, USA: ACM, OCT 2007, p. 598609.

[8] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote data checking using provable data possession," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, pp. 12:1–12:34, 2011.

[9] F. Ergün, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan, "Spot-checkers," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. Dallas, Texas, United States: ACM, MAY 1998, pp. 259–268.

[10] W. Dai, "Crypto++ library, 5.6.1," http://www.cryptopp.com/, Jan, 2011.

[11] K. Bowers, A. Juels, and A. Opera, "Hail: a high-availability and integrity layer for cloud storage," in *ACM conference on Computer and communications security*. Chicago, IL, USA: ACM, Nov. 2009, p. 187198.

[12] 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*. New York, NY, USA: ACM, NOV 2009, pp. 43–54.

[13] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "Mr-pdp: Multiple-replica provable data possession," in *International Conference on Distributed Computing Systems (ICDCS)*. Beijing, China: IEEE, JUN 2008, pp. 411–420.

[14] R. Elbaz, D. Champagne, C. Gebotys, R. B. Lee, N. Potlapally, and L. Torres, "Hardware mechanisms for memory authentication: A survey of existing techniques and engines," *Transactions on Computational Science*, vol. IV, pp. 1–22, 2009.