SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2018, Vol. 61 032107:1–032107:11
doi: 10.1007/s11432-017-9116-x

# Bi-directional and concurrent proof of ownership for stronger storage services with de-duplication

Taek-Young YOUN* & Ku-Young CHANG

*Electronics and Telecommunications Research Institute, Daejeon* 34129*, Korea*

Received 5 January 2017/Accepted 10 May 2017/Published online 13 November 2017

**Abstract** In storage service, data de-duplication is a specialized technique for eliminating duplicate copies of repeating data in storage. Especially, client-side de-duplication has more merits than server-side de-duplication since they can improve both the space efficiency and the communication bandwidth. For secure client-side de-duplication, we need a way to prove the ownership of a file to be stored. In the upload step, the server should verify the ownership of a client to give the right of the file without uploading it. On the contrary, the client also want to verify the retrievability for the file since he will delete it from his storage after protocol execution. Existing proof of ownership techniques have been designed for server's need. In this paper, we first point out that we need the second property in client's view point, and give a very simple and practical solution which can support the server and the client to prove that they have the same file. We first describe a generic strategy which can help us to construction a bi-directional and concurrent proof of ownership technique from an ordinary proof of ownership technique, and then give an efficient hash-based scheme with security proof in the random oracle model.

**Keywords** cloud storage, secure de-duplication, data out-source, proof of ownership, bi-directional and concurrent proof

**Citation** Youn T-Y, Chang K-Y. Bi-directional and concurrent proof of ownership for stronger storage services with de-duplication. Sci China Inf Sci, 2018, 61(3): 032107, doi: 10.1007/s11432-017-9116-x

## 1 Introduction

In these days, information services based on remote storages are rapidly growing due to the increase of the volume of information used for services. For such services, a client deposits his data to remote storage server without storing the data in his storage and a service provider possesses a huge storage to maintain large scaled data stored by its clients. Note that the server's storage can be a bottleneck for the above mentioned services, and thus there are many trials to improve the storage efficiency. Among existing approaches, data de-duplication schemes have been studied as specialized techniques for storage efficiency which eliminate duplicate copies of repeating data in storage [1–11].

There are two kinds of de-duplication techniques, server-side de-duplication techniques and client-side de-duplication techniques. In server-side de-duplication techniques, clients always upload their data to server's storage. Then the server searches and eliminates duplicate copies of repeating data in storage. In the first case, there is no merits for clients but the server can reduce the size of storage for maintaining clients' data. In client-side de-duplication techniques, a client and a server checks whether the file, which will be uploaded by the client, exists in the server's storage or not. If the file does not exist, the client

* Corresponding author (email: taekyoung@etri.re.kr)

© Science China Press and Springer-Verlag Berlin Heidelberg 2017 

simply uploads it. Otherwise, if the same file is already stored in server's storage, the client does not upload it and the server adds the clients as a owner of the file without receiving it from the client. The second technique is good for both the client and the server. The client can store his file without actually uploading the file if the file is already stored in the server's storage, and thus he can reduce the waiting time for uploading. At the same time, the server can eliminate duplicate copies, and thus he can improve storage efficiency. Since client-side de-duplication techniques have many merits than server-side de-duplication techniques, researches have designed many client-side de-duplication techniques with improved performance and functionality.

There are some security issues in client-side de-duplication techniques [4]. In the client's view point, the existence of the same file is the most important issue since he will delete the file after obtaining the ownership (without uploading it). Therefore, if the file does not exist or damaged file is stored in server's storage, the client loses his file. Though the client can complain in the case of data loss and make a reasonable claim to recover his file, it could be meaningless when the file is not recoverable. It is reasonable to assume that the server is a honest entity since he profits by supporting the storage service, and thus the server will not pretend to have a file without possessing it. Even now, we still cannot ignore the possibility of unintentional data loss. For example, honest server could be a victim of poison attacks, which implies that a damaged or maliciously forged file could be stored by an adversarial user. On the other side, to prove that the client actually possesses the file is very important since he will give the access capability of the client regarding the file without performing the upload step. If the client claims that he possess the file but the assertion is not true, it means that the server unintentionally helps the client to illegally obtain the file. Hence, the server have to verify that the client actually have the file at that time. To sum up, the server and the client have different worries when they perform client-side de-duplication for efficient data management. Though the worries of the server and the client come from different necessities, the following sentence is the main question:

"Does the counterpart actually possess the file that I have?"

In other words, to mutually prove the ownership of a file is one of fundamental security requirements for secure client-side de-duplication.

In this paper, we give the first bi-directional and concurrent proof of ownership technique which can support the server and the client to prove that they have the same file. At first, we design a generic construction strategy to design a bi-directional and concurrent proof of ownership technique from an ordinary proof of ownership technique, and then give an efficient instantiation which can be implemented by using (only) hash functions. The proposed hash-based construction is very efficient in terms of the computational complexity and the communication overhead in the sense that the scheme requires almost the same cost compared with existing client-side de-duplication techniques which do not support the bi-directional and concurrent ownership proof. To demonstrate the security of the proposed scheme, we give a proof which shows that any adversarial party cannot pretend to have a file by passing the protocol execution of the proposed scheme without actually possessing the file.

### 1.1 Related work

Recall that the main subject of this work is the proof of ownership. Especially, we are interested in giving bi-directional and concurrent proof of ownership differently from existing techniques which support uni-directional proof. Existing proof of ownership techniques are uni-directional construction in the sense that they are aimed to verify the ownership of client not to verify the ownership of both the client and the server.

In the literature, a number of techniques have been invented to verify the ownership of a client or a server [12–25] . Among them, the schemes in [15, 18, 21–24] are exactly identical with our approach in the server's view point since they are designed to verify the ownership of a client. However, they are not bi-directional ownership proof in the sense that a client does not verify the ownership of a server. There are some techniques which permit a client to prove the ownership of a server [12–14, 16, 17, 20, 25–28], but

they are not designed to permit a client to prove the existence of a file before uploading it. The techniques are designed to permit a client to monitor the current status of his data which has already stored by the client. Note that existing techniques are not designed to support clients to verify whether the file to be stored is already exists in server's storage or not. Here, we want to emphasis the following two points: (1) we need a bi-directional and concurrent proof of ownership technique when a client uploads a file to a server, and (2) we cannot easily support bi-directional and concurrent proof of ownership in the upload procedure using existing techniques.

In [19], a new ownership proof technique has been proposed, which can be used for examining the ownership of both a client and a server. The scheme can be used as a ownership proof technique for a client and a server, but it does not mean that the scheme can be used to prove two parties' ownership at once as our scheme does. To support bi-directional and concurrent proof of ownership using the protocol in [19], we have to execute the protocol twice by changing the prover and the verifier, and this approach requires double cost compared with the cost of single protocol execution. As we know, in the literature, our scheme is the first ownership proof technique which can support bi-directional and concurrent ownership proof without increasing the cost twice as the trivial solution does.

## 2 Brief review of client-side de-duplication

Before describing our protocol, we review a basic procedure for client-side de-duplication which does not adopt the proof of ownership technique. To upload the file $F$, the client Clt and the server Srv perform the following steps:

- Step 1. To check the existence of the file $F$, Clt generates a tag $\mathsf{tag}_F$ for the file. In general, we use a cryptographic hash function $h(\cdot)$ to generate the tag, and the tag is simply computed as $\mathsf{tag}_F = h(F)$. Then Clt sends the tag $\mathsf{tag}_F$ to Srv.
- Step 2. Srv checks the existence of $F$ using given $\mathsf{tag}_F$. If the same file is not stored in the server's storage, Srv sends $\mathsf{N/A}$ to Clt. Otherwise, Srv assigns the the ownership of Clt regarding the file $F$ and informs this fact to Clt.
- Step 3. If Clt receives $\mathsf{N/A}$ from Srv, he uploads the file $F$ and deletes the file $F$ from his storage. Otherwise, Clt deletes the file $F$ without uploading it.

As indicated in [4], there are some security flaws in the above described naive client-side de-duplication when the server Srv does not verify the ownership of the client Clt. To solve the problem, a proof of ownership technique has been proposed in [18], and most of client-side de-duplication schemes supporting the proof of ownership are executed as following:

- Step 1. To check the existence of the file $F$, Clt generates a tag $\mathsf{tag}_F = h(F)$ and sends the tag to Srv.
- Step 2. Srv checks the existence of $F$ using given $\mathsf{tag}_F$. If the file is not stored in his storage, Srv sends $\mathsf{N/A}$ to Clt. Otherwise, Srv generates a challenge $\mathsf{c}$ and sends it to Clt.
- Step 3. If Clt receives $\mathsf{N/A}$ from Srv, he uploads the file $F$ and stops the protocol. Otherwise, Clt computers $\pi = \mathsf{Proof}(\mathsf{c}, F)$ as a proof of the ownership and sends $\pi$ to Srv.
- Step 4. Srv verifies given proof $\pi$. If it is a correct proof, Srv admits the ownership of Clt regarding the file $F$ and informs the fact to Clt. Otherwise, Srv sends $\mathsf{N/A}$ to Clt.
- Step 5. When Clt receives $\mathsf{N/A}$, Clt uploads the file $F$ and deletes the file $F$ from his storage. Otherwise, Clt deletes the file $F$ from his storage without uploading the file.

In the above procedure, the server can prove the ownership of a client, but the client cannot prove the server's ownership regarding the file to be stored. Recall that the main goal of this paper is to give a generic ownership proof technique for secure client-side de-duplication, which supports bi-directional and concurrent proof of ownership in the sense that the server and the client can prove each other's ownership at the same time. Trivially, we can give bi-directional ownership proof by performing uni-directional proof of ownership protocol twice by exchange the role of the prover and the verifier. However, this approach requires (roughly) the cost of two times. The main contribution of this work is to design an efficient

generic construction which requires almost the same cost compared with the client-side de-duplication with uni-directional proof of ownership.

## 3 Bi-directional and concurrent proof of ownership for client-side de-duplication

In this section, we give a generic construction for bi-directional and concurrent proof of ownership technique which supports bi-directional ownership proof in client-side de-duplication. To design the generic scheme, we will use two well-known schemes, a commitment scheme and a (uni-directional) proof of ownership technique. Since our generic scheme requires an uni-directional ownership proof technique as a component, our technique can be seen as a translator which transforms an uni-directional ownership proof to a bi-directional ownership proof. Before describing our construction, we will give brief reviews for the schemes and some other component algorithms, and then we give our generic scheme based on them.

### 3.1 Component algorithms

Here, we briefly describe components algorithms for our construction including the above mentioned commitment scheme and (uni-directional) proof of ownership scheme.

**Tag Generation Function.** Let $\mathsf{GenT}(F)$ be a function which returns a tag for given file $F$. In this paper, we are interested in deterministic algorithm which returns a fixed value for a file. Since a tag is used for identifying a file, we expect that a tag is evaluated from only one file. The property resembles to the collision resistance of hash function, and thus the hash functions are widely used for generating tag.

**Pseudo-Random Number Generator.** Let $\mathsf{prng}(\cdot)$ be a deterministic algorithm which takes a seed as an input and generates a pseudo random number (or sequence) as the corresponding output based on the input seed. For the security of our construction, it should be hard to guess the computed random number (or sequence) without the seed, which is naturally guaranteed by existing pseudo random number generators.

**Commitment Scheme.** We need a commitment scheme $\mathsf{Commit}(\cdot)$ which returns a commitment for a message. The commitment is a sealed message which can be revealed later with the binding property. The binding property means that nobody can change the committed value after a commitment is generated. In general, the commitment generation function $\mathsf{Commit}(\cdot)$ requires random information to generate a commitment, and the random information is used for de-commitment.

**Uni-directional Ownership Proof Scheme.** Let $\mathsf{Proof}(\cdot)$ be a (uni-directional) proof generation function which generates a proof for given a file. Note that, in general, the proof generation is a challenge/reponse protocol between the server and the client. In an uni-directional ownership proof, the server chooses a random challenge and gives it to the client, then the client generates a proof based on the target file and the challenge. For the security of the scheme, we expect that the client can generate valid proof only if he actually has the file.

### 3.2 Description

From now on, we describe the protocol execution between a client Clt and a server Srv.

• Step 1. To check the existence of the file $F$, Clt generates a tag $\tau = \mathsf{GenT}(F)$. Then he chooses a random $\alpha$ and sends it with the tag to the server.

• Step 2. The server checks the existence of $F$ using given tag $\tau$. If the file is not stored in the server's storage, he sends $\mathsf{N/A}$ to the client. Otherwise, the server chooses a random number $\beta$ and computes $r = \mathsf{prng}(\alpha||\beta)$. He also chooses a random $t$[1)] and generates a commitment $\sigma = \mathsf{Commit}(t, \pi)$ where $\pi = \mathsf{Proof}(r, F)$. Then the server sends $\beta$ and $\sigma$ to Clt.

1) Note that the random value $t$ will be used for de-commitment. For some case, a value evaluated from $t$ is opened for de-commitment instead of $t$, but we will describe the case where the value is used without modification.

• Step 3. If Clt receives $\mathsf{N/A}$ from the server, he uploads the file $F$ and stops the protocol execution. Otherwise, he computes $r = \mathsf{prng}(\alpha||\beta)$ and $\pi = \mathsf{Proof}(r, F)$. Then he sends the proof $\pi$ to the server.

• Step 4. The server tests if the received proof is identical with $\pi$. If they are not the same, he sends $\mathsf{N/A}$ to the client. Otherwise, he admits the ownership of Clt regarding the file $F$ and sends $t$ to the client.

• Step 5. If the client receives $\mathsf{N/A}$ from the server, he uploads the file $F$ to the server and deletes the file $F$ from his storage. Otherwise, if he receives $t$ from the server, he verifies given commitment $\sigma$ using $t$. If $\sigma$ is verified as a valid commitment for $\pi$, the client deletes the file $F$ from his storage. If it is not a valid commitment for $\pi$, the client uploads the file $F$ to the server and deletes the file $F$ from his storage.

### 3.3 Security analysis

The proposed generic construction is composed of four primitives, a tag generation function, a (pseudo) random number generator, a commitment scheme, and an (uni-directional) ownership proof scheme. Our scheme is secure if component algorithms satisfy the requirements discussed in Subsection 3.1. Note that we will not give detailed security proof for the generic construction since the security of the proposed scheme can be easily understood without complex formal security proof. Recall that the proposed scheme is designed to prove the ownership of the client and the server at once, which means that the security requirements for the scheme is two folds:

• The client cannot obtain the ownership of a file without possessing it; and

• The server can ensure the existence of a file only if he actually possessing it.

From now on, we discuss the security of the proposed generic construction in terms of two security requirements.

#### 3.3.1 *Client's ownership*

In an adversarial client's viewpoint, the proposed generic construction is almost identical with the underlying uni-directional ownership proof scheme except two differences. The first difference is the way of generating random challenge. In the proposed scheme, the challenge is generated by two parties instead of chosen by the server as in existing uni-directional proof of ownership schemes. Though the client participates in the generation of the challenge, it is not easy to guess the random challenge before obtaining full input for the random number generator. Hence, the client cannot guess the challenge $r$ until he receives $\beta$ from the server. The second difference is that the client can obtain a committed proof $\sigma$ from the server before he generates a correct proof. If the committed proof can help the client to generate valid proof without the file, the proposed scheme is less secure than the underlying uni-directional ownership proof scheme in the client's viewpoint. However, a secure commitment scheme guarantees that a commitment does not reveal any information of the committed value, which implies that the client has no merit in generating valid proof $\pi$ if the underlying commitment scheme is secure. Hence, the client cannot obtain the ownership of the file by cheating the server if the underlying uni-directional ownership proof scheme and the commitment scheme are secure.

#### 3.3.2 *Server's ownership*

To pass the protocol described in Subsection 3.2 without possessing the target file $F$, the server should generate a commitment $\sigma$ in Step 2, and it should be a valid commitment for the proof $\pi$. The server cannot generate valid proof $\pi$ without $F$ since the underlying uni-directional ownership proof scheme is assumed to be secure, which implies that the commitment $\sigma$ is not evaluated from a valid proof. Recall that the server can deceive the client only if the commitment can be opened in Step 5. The client sends the proof $\pi$ to the server in Step 3, and thus the server can use the valid proof to find suitable $t$ which satisfies the condition $\mathsf{Open}(\sigma, t, \pi) = \mathsf{true}$. However, to find valid opening information $t$, the server should break the security of the underlying commitment scheme. Hence, the server cannot obtain the ownership of the file by cheating the client if the underlying uni-directional ownership proof scheme and the commitment scheme are secure.

# 4 Light instantiation based on hash function

In Section 3, we have described a generic bi-directional and concurrent proof of ownership technique. In this section, we give an efficient instantiation which can be implemented only using cryptographic hash functions. We prove the security of the light instantiation by measuring the advantages of adversarial parties.

## 4.1 Component algorithms

### 4.1.1 *Tag generation function*

Any function which generates uniquely identifiable tag for each file can be used as a tag generation function, and thus we generally use a cryptographic hash function due to its collision resistance. Even though a hash function is an efficient algorithm, the cost of hash function could be a burden when the size of an input file is large. Fortunately, we can efficiently implement the tag generation function when the ownership proof is considered in client-side de-duplication since the collision resistance of a tag can be relaxed[2)]. Since we test again if the file to be stored is identical with a file (already stored in server's DB) by performing the ownership proof procedure, it is not meaningful to make a collision in adversaries' view point. Therefore, any tag generation function can be used in our construction if a tag generated by the function can be used to identify a file.

A candidate construction is to evaluate the hash value for attributes of a file instead of hashing the whole file. We may need various attributes to evaluate the tag to give sufficient collision resistance, and the variety of attributes could be determined by suitable simulation. Surely, we can use the above mentioned example since the values are typically used to find a file in a computer.

### 4.1.2 *Pseudo-random number generator*

Recall that we use a pseudo-random number generator (PRNG) to guarantee that an adversarial party not knowing the input has only negligible advantage in guessing the generator's output sequence which will be used as a challenge. In the light instantiation, we will use a hash function as a PRNG. If the input value has sufficient entropy, it is not easy to guess the output of a hash function without obtaining the input. In our scheme, two random values, which are separately chosen by the server and the client, are used as inputs for generating a challenge, and thus each party cannot guess the challenge until he obtains the other party's random value.

### 4.1.3 *Commitment*

We can implement a commitment scheme based on a hash function as follows. Let $\tau$ be the hashed value of $r||m$ where $r$ is a random value and $m$ is a message. Then, $\tau$ can be used as a commitment for the message. Recall that we need a commitment scheme to open a sealed message with the binding property. In the above setting, $\tau$ is the sealed message for $m$, and we can prove that the value is computed from $m$ by opening the random value $r$. Anyone can verify the commitment by testing the following equality: $\tau = h(r||m)$.

### 4.1.4 *(Uni-directional) proof of ownership*

A hash function could be a simple and secure uni-directional proof of ownership scheme. We can use $\pi = h(chal||File)$ as a ownership proof for the file $File$ and the challenge $chal$. It is clear that we can compute a valid proof for given a challenge only if the whole file is given.

2) When the ownership proof is not considered in client-side de-duplication, the only way to find duplicated copy is to find the same tag. In this case, we expect unforgeable correctness of the tag-based sameness test, and thus a hash value for the whole file was inevitably used as a tag to support strong collision resistance.

### 4.2 Description

In this section, we give a practical instantiation of the generic construction described in Section 3. We will design a concrete scheme which can be implemented only using hash functions. We need four hash functions $H_{\mathsf{t}}(\cdot)$, $H_{\mathsf{r}}(\cdot)$, $H_{\mathsf{c}}(\cdot)$, and $H_{\mathsf{p}}(\cdot)$ for the tag generation function, the pseudo-random number generator, the commitment scheme, and the uni-directional proof of ownership scheme, respectively. From now on, we describe the protocol execution between a client Clt and a server Srv.

**Step 1.** To check the existence of the file $F$, the client generates a tag $\mathsf{tag}_F = H_{\mathsf{t}}(F_{\text{metadata}})$ where $F_{\text{metadata}}$ is a set of metadata of the file $F$. To define the metadata, we can consider various attributes of files such as the file name, the file type, file size, encoding format, and so on. Then the client chooses a random seed $\alpha$ and sends it with the tag to the server.

**Step 2.** Then the server checks the existence of $F$ using given $\mathsf{tag}_F$. If the file is not stored in the server's storage, he sends $\mathsf{N/A}$ to the client. Otherwise, the server chooses a random number $\beta$ and computes $s = H_{\mathsf{r}}(\alpha||\beta)$ and $h_F = H_{\mathsf{p}}(s||F)$. He also chooses a random $t$ and evaluates $\tau = H_{\mathsf{c}}(t||h_F)$ (as a commitment). Then the server sends $\beta$ and $\tau$.

**Step 3.** If Clt receives $\mathsf{N/A}$ from the server, he uploads the file $F$ and stops the protocol execution. Otherwise, when the client receives $\beta$, he computes $s = H_{\mathsf{r}}(\alpha||\beta)$ and $h_F = H_{\mathsf{p}}(s||F)$. Then he sends $h_F$ to the server.

**Step 4.** The server tests if the received hash value is identical with $h_F$. If they are not identical, he sends $\mathsf{N/A}$ to the client. Otherwise, he admits the ownership of Clt regarding the file $F$ and sends $t$ to the client.

**Step 5.** If the client receives $\mathsf{N/A}$ from the server, he uploads the file $F$ to the server and deletes the file $F$ from his storage. Otherwise, if he receives $t$ from the server, he tests if $H_{\mathsf{c}}(t||h_F)$ is identical with $\tau$. If the equality test holds, the client deletes the file $F$ from his storage. If the test fails, the client uploads the file $F$ to the server and deletes the file $F$ from his storage.

We use four hash functions for underlying schemes. We can implement them using one hash function, but we separate them for the security proof which will be given in the following section.

### 4.3 Security proof

Here, we will prove the security of the hash-based scheme in the random oracle model where a hash function is simulated as a random oracle.

**Theorem 1.** The hash-based client-side de-duplication technique given in Subsection 4.2 permits the server and the client to verify the ownership of their counterpart regarding a file to be stored.

*Proof.* Let Srv be a server and Clt be a client served by the server Srv. We assume that Srv provides storage services using the proposed hash-based client-side de-duplication technique. Let $\{H_{\mathsf{t}}, H_{\mathsf{r}}, H_{\mathsf{p}}, H_{\mathsf{c}}\}$ be the set of hash functions which are used in the proposed scheme.

Recall that the proposed hash-based technique is secure if

1. A server cannot guarantee the existence of the file $F$ without possessing it; and

2. A client cannot obtain the ownership of the file $F$ without possessing it.

In other words, the proposed technique is secure if the server and the client cannot pass the protocol execution for a file without actually possessing the file. To prove this, we will show that the advantage of an adversarial party is bounded by a negligible value due to the cryptographic property of hash functions.

Recall that the goal of an adversarial party is to pass the protocol execution for a file without actually possessing the file. The server and the client have to generate different values to pass the protocol execution, and thus we will separately discuss the security of the proposed scheme with respect to the server and the client. Let $F$ be a target file. Then, a malicious server or client may try to pass the protocol execution to prove its own ownership regarding the file $F$ even though he doesn't actually has it.

*Security against Srv.* Among the execution of protocol, the server publishes only three values $\{\beta, \tau, t\}$,

and the server succeeds in deceiving the client only if the following equation holds:

$$H_{\mathsf{c}}(t||h_F) = \tau,$$

where $h_F = H_{\mathsf{p}}(s||F)$ and $s = H_{\mathsf{r}}(\alpha||\beta)$. The server can compute $s$ using two inputs $\alpha$ and $\beta$ since they are exchanged before evaluating the random challenge $s$. Since the server doesn't have the file $F$, there are only two possible scenarios:

1. Srv correctly guesses $h_F$ and computes $\tau$ by choosing a random $t$; and
2. Srv sends a random $\tau$ to the client and finds a valid $t$ after receiving $h_F$ from the client.

Let $\mathsf{E}$ be the event that the server correctly guesses $h_F$ without the file $F$ and $\mathsf{R}$ be the event that the relation $H_{\mathsf{c}}(t||h_F) = \tau$ holds. Then, the server's advantage in deceiving the client can be expressed as

$$\mathsf{Adv}_{\mathsf{Srv}} = \mathsf{Pr}[\mathsf{R}] = \mathsf{Pr}[\mathsf{R}|\mathsf{E}] \cdot \mathsf{Pr}[\mathsf{E}] + \mathsf{Pr}[\mathsf{R}|\neg\mathsf{E}] \cdot \mathsf{Pr}[\neg\mathsf{E}].$$

It is trivial that $\mathsf{Pr}[\mathsf{R}|\mathsf{E}] = 1$, and thus the above equation can be simplified as

$$\mathsf{Adv}_{\mathsf{Srv}} \leqslant \mathsf{Pr}[\mathsf{E}] + \mathsf{Pr}[\mathsf{R}|\neg\mathsf{E}].$$

Since the server doesn't have the file $F$, the probability $\mathsf{Pr}[\mathsf{E}]$ is bounded by $\epsilon$ where $1/\epsilon = 2^m$ is the minimum-entropy of files. The right part $\mathsf{Pr}[\mathsf{R}|\neg\mathsf{E}]$ is the probability that the server sends a random $\tau$ to the client and finds a valid $t$ after receiving $h_F$ from the client, which contradicts to the *pre-image resistance* of the underlying hash function. Let $2^k$ be the complexity of the *pre-image resistance* of the underlying hash function. Then, $\mathsf{Pr}[\mathsf{R}|\neg\mathsf{E}]$ is bounded by $1/2^k$. Therefore, we can see that

$$\mathsf{Adv}_{\mathsf{Srv}} \leqslant \frac{1}{2^m} + \frac{1}{2^k}.$$

*Security against Clt.* In the protocol execution, Clt publishes three values $\{\mathsf{tag}_F, \alpha, h_F\}$. Note that, as we discussed in Subsection 4.1.1, the tag generation function is not an important component for security. We permit adversaries to obtain the tag of the target file, and thus it can be assumed that the adversary knows $\mathsf{tag}_F$. Since the random number $\alpha$ can be chosen regardless of the target file, the client succeeds in deceiving the server if he can generate valid $h_F$ without the file $F$. The client receives $\beta$ and $\tau$ from the server before generating $h_F$. The first random number can be used to derive $s$ by computing $H_{\mathsf{r}}(\alpha||\beta)$. Then, the client can use $s$ and $\tau$ to generate $h_F$ where $s$ is a part of input for $h_F$ and $\tau$ is a commitment generated based on $h_F$. Note that $\tau$ is computed by the server as $H_{\mathsf{c}}(t||h_F)$ for randomly chosen $t$. Let $\mathsf{Succ}$ be the event that the client succeeds in deceiving the server, which also can be seen as the event that the client succeeds in evaluating $h_F$ using $s$ or extract the hash value from $\tau$. Let $\mathsf{S}_1$ be the event that the client evaluates $h_F$ using $s$ and $\mathsf{S}_2$ be the event that the client extracts $h_F$ from $\tau$. Then the advantage of the client can be written as following:

$$\mathsf{Adv}_{\mathsf{Clt}} = \mathsf{Pr}[\mathsf{Succ}] \leqslant \mathsf{Pr}[\mathsf{S}_1] + \mathsf{Pr}[\mathsf{S}_2].$$

From now on, we measure each probability $\mathsf{Pr}[\mathsf{S}_i]$ for $i = 1, 2$. Recall that $h_F$ is computed by $H_{\mathsf{p}}(s||F)$ and the client knows only the random part $s$. Let $\mathsf{Cor}$ be the event that the client correctly guesses the file $F$. Then, we can express $\mathsf{Pr}[\mathsf{S}_1]$ as

$$\mathsf{Pr}[\mathsf{S}_1] = \mathsf{Pr}[\mathsf{S}_1|\mathsf{Cor}] \cdot \mathsf{Pr}[\mathsf{Cor}] + \mathsf{Pr}[\mathsf{S}_1|\neg\mathsf{Cor}] \cdot \mathsf{Pr}[\neg\mathsf{Cor}] \leqslant \mathsf{Pr}[\mathsf{Cor}] + \mathsf{Pr}[\mathsf{S}_1|\neg\mathsf{Cor}].$$

The probability of correct guess is bounded by the minimum-entropy of the file, and thus we have $\mathsf{Pr}[\mathsf{Cor}] \leqslant 1/2^m$. When the client fails to correctly guess the file $F$, the event $\mathsf{S}_1$ occurs only if the hash of incorrectly guessed file is equal to $h_F$ of which probability is bounded by the probability of the pre-image resistance of the underlying hash function. Therefore, we have $\mathsf{Pr}[\mathsf{S}_1|\neg\mathsf{Cor}] \leqslant 1/2^k$, and we have

$$\mathsf{Pr}[\mathsf{S}_1] \leqslant \frac{1}{2^m} + \frac{1}{2^k}.$$

The second event $\mathsf{S}_2$ occurs only if the hash function $H_{\mathsf{c}}(\cdot)$ is broken in terms of the pre-image attack which implies that

$$\mathsf{Pr}[\mathsf{S}_2] \leqslant \frac{1}{2^k}.$$

Then, finally, we have

$$\mathsf{Adv}_{\mathsf{Clt}} \leqslant \frac{1}{2^m} + \frac{2}{2^k}.$$

Based on the above discussions, the proposed scheme is secure since the advantages of a malicious server and a client are bounded by

$$\frac{1}{2^m} + \frac{1}{2^k} \text{ and } \frac{1}{2^m} + \frac{2}{2^k},$$

respectively.

**Remark 1.** There is a gap in the proof for the security against an adversarial client since we exclude the attack scenario where the client uses both $s$ and $\tau$ to find $h_F$. However, we can ignore the gap since the increased advantage which can be obtained by performing the excluded attack is negligible. The excluded attack can be classified into two cases. The first case is that the adversary tries to evaluate $h_F$ using both $s$ and $\tau$. Only the difference compared with the event $\mathsf{S}_1$ in Theorem 1 is that the client can use $\tau$ to evaluate $h_F$. Recall that an output of a hash function looks like a random value if all input values are not known. Hence, the additional information is a useless random value in the view point of the client since he doesn't know an input $t$. The other scenario is that the adversary tries to extract $h_F$ from $\tau$ using $s$. In this case, the random value $s$ is also given differently from the event $\mathsf{S}_2$ in Theorem 1. Recall that a hash function maps a string of arbitrary length to a string of fixed length, and the property guarantees the existence of a pair $\{s', F'\}$ such that $h_F = H_{\mathsf{p}}(s'||F')$ since many values are mapped to a single value. Hence, any random value can be a part of an input for $h_F$, which implies that a part of pre-image of a committed value $h_F$ is not useful to extract $h_F$ from $\tau$. Hence, we can ignore the security gap exists in Theorem 1.

### 4.4 Performance

In our scheme, the client computes four hash values to generate a tag, a random challenge, a proof, and a commitment, and the server computes almost the same operations except the generation of a tag. Recall that the cost of hash function is determined by the size of input. Among four hash values, only one value is evaluated from a entire file and other values are computed from small inputs whose size are fixed. When the size of file is large, the cost for evaluating three hash values for small inputs is negligible compared with the cost for evaluating the hash value of the whole file. In short, if the size of file is not too small, the computational cost of our scheme is roughly one hash evaluation of the whole file.

When we does not consider the proof of ownership in client-side de-duplication, we use the hash value for the whole file as the tag. Therefore, in the client's viewpoint, the cost of our scheme is almost identical with the client-side de-duplication which does not support the proof of ownership. If we use uni-directional proof of ownership in client-side de-duplication, we can reduce the cost of tag generation as in our scheme. However, we still need to generate a proof which requires at least one hash evaluation for the whole file. Therefore, the proposed scheme supports bi-directional proof of ownership without increasing the cost of computation compared with the client-side de-duplication without the ownership proof and the client-side de-duplication with uni-directional ownership proof. Note that, in our scheme, the server generates a proof to prove its ownership, and the computation inevitably requires additional cost. The communication overhead of the proposed technique is almost identical with existing client-side de-duplication schemes supporting uni-directional ownership proof. The cost of communication overhead includes the size of messages and the number of rounds. Note that the size of transmitted messages is not completely identical with that of existing techniques, but the difference is negligible if the size of file is not too small.

In Table 1, we summarize the cost of the proposed technique in terms of the computational complexity and the communication overhead. Since our technique does not influence on the performance of the

**Table 1** Summary of cost

| | | Cost for the first upload | | Cost for the $i$th upload ($i \geqslant 2$) | |
|---|---|---|---|---|---|
| | | Computation | Communication | Computation | Communication |
| Generic scheme | Client | $c_{\mathsf{T}}$ | $\ell_{\mathsf{D}}$ | $c_{\mathsf{T}}+c_{\mathsf{R}}+c_{\mathsf{P}}+c_{\mathsf{V}}$ | $\ell_{\mathsf{T}}+\ell_{\mathsf{R}}+\ell_{\mathsf{P}}$ |
| | Server | – | – | $c_{\mathsf{R}}+c_{\mathsf{P}}+c_{\mathsf{C}}$ | $2\ell_{\mathsf{R}}+\ell_{\mathsf{C}}$ |
| Hash-based scheme | Client | $c_{\mathsf{H}}(\ell_{\mathsf{m}})$ | $\ell_{\mathsf{D}}$ | $c_{\mathsf{H}}(\ell_{\mathsf{m}})+c_{\mathsf{H}}(2\ell_{\mathsf{R}})+c_{\mathsf{H}}(\ell_{\mathsf{D}}+\ell_{\mathsf{R}})+c_{\mathsf{H}}(\ell_{\mathsf{R}}+\ell_{\mathsf{H}})$ | $3\ell_{\mathsf{H}}$ |
| | Server | – | – | $c_{\mathsf{H}}(2\ell_{\mathsf{R}})+c_{\mathsf{H}}(\ell_{\mathsf{D}}+\ell_{\mathsf{R}})+c_{\mathsf{H}}(\ell_{\mathsf{R}}+\ell_{\mathsf{H}})$ | $3\ell_{\mathsf{H}}$ |

download procedure, only the cost for the upload procedure will be summarized in the table. Let $c_{\mathsf{T}}$, $c_{\mathsf{R}}$, $c_{\mathsf{P}}$, and $c_{\mathsf{C}}$ be the computational complexity of the tag generation function, the pseudo random number generator, the proof generation function, and the commitment generation function, respectively, and the size of their output will be denoted by $\ell_{\mathsf{T}}$, $\ell_{\mathsf{R}}$, $\ell_{\mathsf{P}}$, and $\ell_{\mathsf{C}}$, respectively. Let $c_{\mathsf{V}}$ be the cost of the commitment verification function. Let $c_{\mathsf{H}}(\ell)$ be the cost of hash evaluation of an $\ell$-bits input value and $\ell_{\mathsf{H}}$ be the size of the output of the hash function. Let $\ell_{\mathsf{D}}$ be the size of the full data. In our scheme, two parties choose three random values, $\alpha, \beta$, and $t$, but we did not consider the cost of choosing the values since the parties can choose the values without using the pseudo random number generator. We also omit the cost of some insignificant operations. For example, we exclude the cost of sending the message "$\mathsf{N/A}$". For the simplicity of the explanation, we assume that the size of a chosen random number is identical with the size of the output of the hash function.

## 5 Conclusion

In this paper, we first discuss the necessity of bi-directional and concurrent proof of ownership for secure client-side de-duplications and give a generic solution for it. We first design a generic scheme and then give an efficient construction which can be implemented by using (only) hash functions. The proposed hash-based construction provides tight efficiency in the sense that it requires almost the same computational complexity and communication overhead compared with client-side de-duplication techniques which do not support bi-directional ownership proof. We prove the security of the proposed hash-based construction by showing that adversarial parties cannot pretend to have a file by passing the protocol execution without actually possessing the file.

**Acknowledgements** This work was supported by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government (17ZH1700, Development of Storage and Search Technologies over Encrypted Database).

**Conflict of interest** The authors declare that they have no conflict of interest.

## References

1 Bellare M, Keelveedhi S, Ristenpart T. Message-locked encryption and secure deduplication. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, 2013. 296–312
2 Bellare M, Keelveedhi S, Ristenpart T. DupLESS: server-aided encryption for deduplicated storage. In: Proceedings of the 22nd USENIX Conference on Security, Washington DC, 2013. 179–194
3 Douceur J R, Adya A, Bolosky W J, et al. Reclaiming space from duplicate files in a serverless distributed file system. In: Proceedings of International Conference on Distributed Computing Systems, Vienna, 2002. 617–624
4 Harnik D, Pinkas B, Shulman-Peleg A. Side channels in cloud services: deduplication in cloud storage. IEEE Secur Privacy Mag, 2010, 8: 40–47
5 Li J, Chen X F, Li M Q, et al. Secure deduplication with efficient and reliable convergent key management. IEEE Trans Parall Distrib Syst, 2014, 25: 1615–1625
6 Li J, Chen X, Xhafa F, et al. Secure deduplication storage systems with keyword search. In: Proceedings of IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), Gwangju, 2014. 971–977
7 Li J, Li Y K, Chen X F, et al. A hybrid cloud approach for secure authorized deduplication. IEEE Trans Parall Distrib Syst, 2015, 26: 1206–1216

8 Marques L, Costa C. Secure deduplication on mobile devices. In: Proceedings of the 2011 Workshop on Open Source and Design of Communication, Lisbon, 2011. 19–26

9 Shin Y, Kim K. Efficient and secure file deduplication in cloud storage. IEICE Trans Inf Syst, 2014, E97-D: 184–197

10 Storer M, Greenan K, Long D, et al. Secure data deduplication. In: Proceedings of the 4th ACM International Workshop on Storage Security and Survivability, Alexandria, 2008. 1–10

11 Xu J, Chang E C, Zhou J Y. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In: Proceedings of ASIA-CCS 2013, Hangzhou, 2013. 195–206

12 Ateniese G, Kamara S, Katz J. Proofs of storage from homomorphic identification protocols. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, 2009. 319–333

13 Ateniese G, Pietro R D, Mancini L, et al. Scalable and efficient provable data possession. In: Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks, Istanbul, 2008. 9

14 Bowers K D, Juels A, Oprea A. Proofs of retrievability: theory and implementation. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, Chicago, 2009. 43–54

15 Blasco J, Pietro R D, Orfila A, et al. A tunable proof of ownership scheme for deduplication using bloom filters. In: Proceedings of the IEEE Conference on Communications and Network Security (CNS), San Francisco, 2014. 481–489

16 Chen J, Zhang L H, He K, et al. Message-locked proof of ownership and retrievability with remote repairing in cloud. Secur Commun Netw, 2016, 9: 3452–3466

17 Dodis Y, Vadhan S, Wichs D. Proofs of retrievability via hardness amplification. In: Proceedings of Theory of Cryptography Conference, San Francisco, 2009. 109–127

18 Halevi S, Harnik D, Pinkas B, et al. Proofs of ownership in remote storage systems. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, Chicago, 2011. 491–500

19 Husain M I, Ko S Y, Uurtamo S, et al. Bidirectional data verification for cloud storage. J Netw Comput Appl, 2014, 45: 96–107

20 Juels A, Kaliski B. PORs: proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, 2007. 584–597

21 Pietro R D, Sorniotti A. Boosting efficiency and security in proof of ownership for deduplication. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, Seoul, 2012. 81–82

22 Rass S. Dynamic proofs of retrievability from Chameleon-Hashes. In: Proceedings of International Conference on Security and Cryptography (SECRYPT), Reykjavik, 2013. 1–9

23 Xu J, Zhou J. Leakage resilient proofs of ownership in cloud storage, revisited. In: Proceedings of International Conference on Applied Cryptography and Network Security, Lausanne, 2014. 97–115

24 Yu C-M, Chen C-Y, Chao H-C. Proof of ownership in deduplicated cloud storage with mobile device efficiency. IEEE Netw, 2015, 29: 51–55

25 Wang H Y, Zhu L H, Xu C, et al. A universal method for realizing non-repudiable provable data possession in cloud storage. Secur Commun Netw, 2016, 9: 2291–2301

26 Armknecht F, Bohli J-M, Karame G O, et al. Outsourced proofs of retrievability. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, 2014. 831–843

27 Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, 2007. 598–609

28 Shacham H, Waters B. Compact proofs of retrievability. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, 2008. 90–107