Private Data Deduplication Protocols in Cloud Storage

Wee Keong Ng
SCE, NTU
50 Nanyang Avenue
Singapore 639798
wkn@pmail.ntu.edu.sg

Yonggang Wen
SCE, NTU
50 Nanyang Avenue
Singapore 639798
YGWEN@ntu.edu.sg

Huafei Zhu
I^{2}R, A*STAR
1 Fusionopolis Way
Singapore 138632
huafei@i2r.a-star.edu.sg

ABSTRACT

In this paper, a new notion which we call private data deduplication protocol, a deduplication technique for private data storage is introduced and formalized. Intuitively, a private data deduplication protocol allows a client who holds a private data proves to a server who holds a summary string of the data that he/she is the owner of that data without revealing further information to the server. Our notion can be viewed as a complement of the state-of-the-art public data deduplication protocols of Halevi et al [7]. The security of private data deduplication protocols is formalized in the simulation-based framework in the context of two-party computations. A construction of private deduplication protocols based on the standard cryptographic assumptions is then presented and analyzed. We show that the proposed private data deduplication protocol is provably secure assuming that the underlying hash function is collision-resilient, the discrete logarithm is hard and the erasure coding algorithm can erasure up to α-fraction of the bits in the presence of malicious adversaries in the presence of malicious adversaries. To the best our knowledge this is the first deduplication protocol for private data storage.

Keywords

Cloud computing, Data storage, Private data deduplication

1. INTRODUCTION

Data deduplication is a technique that stores only a single copy of each file on a storage server regardless of how many clients ask to store that file. In a data deduplication system, a client P sends to a storage server S only a summary string v of a file F, say a Merkle-tree hash value of F. S checks to see whether the received summary string v has stored in its database: if v is not in the database then S asks P to upload the entire file F; otherwise, it tells P that there is no need to send F itself and marks P as an owner of F.

Although the data deduplication technique is considered to be the most-impactful storage technique [11], it is vulnerable to powerful attacks. Harnik et al [8] demonstrate how data deduplication technique can be used as a side channel which reveals information about the contents of files of other users. Specially, Harnik et al consider an attacker that is able to temporarily compromise a server machine, getting access to its internal cache, which includes the hash values for all the recently accessed files. Having obtained this piece of information, the attacker is able to download all these files, which may include confidential files of others.

To overcome such attacks, Halevi et al [7] introduced and formalized the notion of proofs of ownership (the HHPS protocol), where a client P proves to a server S that it actually holds the data of file F and not just some short summary string v. As noted in [7], the data deduplication protocol are closely related to the proofs of retrievability [9, 12, 13] and proofs of data possession [1] but they are significantly different in the sense that the proofs of retrievability and data procession often use a pre-processing step that cannot be used in the data deduplication procedure.

In a nutshell, the HHPS protocol works by encoding the file F using an erasure code E which is resilient to erasure of up to α fraction of the bits, and then building a Merkle-tree over the encoded file $X = E(F)$. More precisely, let H be a collision resistant hash function with output length of λ-bit. Let $\mathrm{MT}_{H,\lambda}(X)$ be the binary Merkle-tree over buffer X using λ-bit leaves and the hash function H. The verifier computes the encoding $X = E(F)$ and the Merkle-tree $\mathrm{MT}_{H,\lambda}(X)$ and keeps the root of the tree as a summary string v. During a proof of an ownership, the verifier chooses at random u leaf indexes $l_1, \ldots, l_u$, where u is the smallest integer such that $(1-\alpha)^u < \varepsilon$, a soundness bound. The verifier asks the prover for the sibling-paths of all the leaves, and accepts if all the sibling paths are valid with respect to $\mathrm{MT}_{H,\lambda}(X)$. The HHPS protocol has been shown that it is a proof-of-the-ownership with soundness ε.

1.1 The motivation problem

The state-of-the-art deduplication technique [7] reveals leaf blocks $X_{l_1}, \ldots, X_{l_u}$ to S for u selected leaf indexes $l_1, \ldots, l_u$. This leakage causes no problem since the entire file F is assumed to be stored at S and thus allows S to learn every bit of the file F. Hence the HHPS protocol is a deduplication protocol for public data storage (public data deduplication protocol, for short).

This paper studies private data deduplication technique for cloud storages. Intuitively, a private data deduplication protocol allows a client who holds a private data proves to a server who holds a summary string of the data that he/she is the owner of that data without revealing further information to the server. Our motivation problem is illustrated by the following data sharing among a group members in the cloud computing scenario:

Let $\{P_1, \ldots, P_m\}$ be a set of members in a group. Let (pk_i, sk_i) be a pair of publc/secret keys of P_i, $i = 1, \ldots, m$. Each member (say P_i) is willing to share its data F among the group members by uploading an encrypted data $\mathcal{E}_{pk_i}(F, r)$ to a common cloud server S so that individual group member (say P_j) can decrypt the ciphertetxt $\mathcal{E}_{pk_i}(F, r)$ and thus enables him/her to read the message F (notice that such a decryption algorithm exists if a trusted coordinator is allowed to help P_j to decrypt the ciphertetxt generated by P_i. We refer to the reader [3] for more details).

Similar to the public data deduplication scenario, when P_i uploads an encryption $\mathcal{E}_{pk_i}(F, r)$ of a file F, she/he first sends to the server S only a summary string v who checks to see whether an encryption of F has been stored in the server (for example, a ciphertext of F has already been uploaded in the common storage server S by another group member P_j). A subtle issue in a private data deduplication protocol is a computation of summary string v. If a summary string v is computed from a probabilistic function, say, a ciphertext $\mathcal{E}_{pk_i}(F, r_F)$ of F is first computed from a semantically secure encryption scheme $\mathcal{E}_{pk_i}(\cdot, \cdot)$ and v is then computed from the Merkle-tree hash values, a check of summary v will be a difficult task since the summary string v computed from the current upload $\mathcal{E}_{pk_i}(F, r_F)$ may be different from that stored in the server.

Alternatively, one may assume that a summary string is computed according to the HHPS protocol where F is known to the server. We however do not know how to compute the common summary string v from the private data file F (*notice that the HHPS protocol cannot be applied in our scenario since F should not be revealed to S in our case, this is the very notion of private data deduplication protocols we consider in this paper*).

Research problems: A challenging research problem thus is that − how to formalize the functionality of private data deduplication protocols? how to define the security of private data deduplication protocols and how to construct private data deduplication protocols if exist?

1.2 This work

While there are known constructions of public data deduplication protocols assuming the existence of collision-resistent hash functions [7], this paper takes the first step to study the feasible result of private data deduplication protocols under the standard cryptographic assumptions and makes the following three-fold contributions:

- in the first fold, a new notion called private data deduplication protocol is first introduced and formalized (Definition 1 in Section 2) in this paper. Our notion can be viewed as a complement of the state-of-the-art public data deduplication protocols of Halevi et al [7]; The security of private data deduplication protocol is formalized in the simulation-based framework (Definition 3 in Section 2) in the context of two-party computations;

- in the second fold, a feasible result of private data deduplication protocols in the standard complexity assumptions is presented. The idea behind our construction is that − a leaf in a Merkle-tree stores a commitment of a block B_i of a private file F. A summary string v is computed from the Merkle-tree commitments (hence the computation of v is independent with an uploaded ciphertetxt, this observation is crucial for constructing our feasible result). That is, the value v_i of a leaf n_i is the commitment C_i of the block B_i and the value of an internal node n_j is the hash $v_j = H(v_l, v_r)$, where v_l and v_r are the values of left- and right- children of v_j. If v_l and v_r are leaf nodes then $v_l = C_l$ and $v_r = C_r$ (i.e., the computation of v does not involve F but a commitment of F). The root value is v. To prove an ownership of F, a verifier S randomly selects a leaf n_i; and the prover P sends a valid path from the leaf value v_i to the root v together with a proof that P knows $(B_{i,1}, \ldots, B_{i,m})$ such that $C_i = g_1^{B_{i,1}} \ldots g_m^{B_{i,m}} \bmod p$. The verifier accepts the proof if all checks are valid.

- in the third fold, we show that the proposed implementation is secure in the simulation-based framework assuming that the underlying discrete logarithm problem defined over Z_p is hard and the underlying hash function is collision resilient.

2. PRIVATE DATA DEDUPLICATION PROTOCOL: FUNCTIONALITY AND SECURITY

In this section, the functionality and security of deduplication protocols for private file storage (or private deduplication protocols) is introduced and formalized in the secure two-party computation model.

2.1 Private data deduplication $\mathcal{F}_{\mathrm{pdd}}$

Intuitively, a private data deduplication protocol is a task between two parties, a client P and server S, where P and S have a common summary string v and a ciphertetxt $c = \mathcal{E}_{pk}(F, r_F)$ (an encryption of a file F under public-key pk using randomness r_F) so that at the end of the protocol execution S is convinced that P is a data owner of a summary string v. Let $R_v(\cdot, \cdot)$ be a Boolean predict defined over v. Let x be a transcript of a proof and w be a witness. If $R_v(x, w) = 1$, then S should accept the proof x; otherwise it will reject. Base on this observation, we formalize the functionality of private data deduplication protocols $\mathcal{F}_{\mathrm{pdd}}$ in terms of the zero-knowledge proof framework [2].

Definition 1. *Parameterized by a binary relation R_v, an imaginary trusted third party TTP proceeds as follows*

- *Upon receiving an input $(prove, sid, x, w)$ from some party P, TTP verifies that $sid = (P, S, sid')$ for some S; else it ignores the input; Next, if $R_v(x, w)$ holds then TTP generates a public delay output $(verified, sid, x)$ to S, else, it does nothing. From now on, TTP ignores further $(prove, \ldots,)$ inputs.*

- *Up receiving a message $(corrupt - prover, sid)$ from an adversary, TTP sends w to the adversary. Furthermore, if the adversary now provides a value $(\overline{x}, \overline{w})$ such that $R_v(\overline{x}, \overline{w})$ holds, and no output was yet written to S, then TTP outputs $(verified, sid, \overline{x})$ to S.*

2.2 The security

The security of private data dedupliction protocols is therefore formalized in the standard simulation-based framework in the context of two-party computations, where a client P is called a prover while a server S is called a verifier.

Notations [5, 6]: Let $\mathcal{I}$ be a countable index set. An ensemble indexed by $\mathcal{I}$ is a sequence of random variables $\{X_i\}_{i\in\mathcal{I}}$, where each X_i is a random variable. Let $\mathcal{I}=\mathcal{N}$ and $\{X_n\}_{n\in\mathcal{N}}$ and $\{Y_n\}_{n\in\mathcal{N}}$ be sequences of distributions with X_n, Y_n ranging over $\{0,1\}^{l(n)}$ for some $l(n) = n^{O(1)}$.

$\{X_n\}_{n\in\mathcal{N}}$ and $\{Y_n\}_{n\in\mathcal{N}}$ are computationally indistinguishable, denoted by $X_n \approx Y_n$, if for every polynomial time D and polynomial bounded $p(n)$, and sufficiently large n,

$$|Pr[D(X_n) = 1] - Pr[D(Y_n) = 1]| \le \frac{1}{p(n)}$$

Execution in the ideal model: Let f=(f_1, f_2) be a two-party functionality, and P_1 and P_2 be two parties. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial time machine, and let $\mathcal{C} \subseteq \{P_1, P_2\}$ be the index of a corrupted party, controlled by an adversary $\mathcal{A}$. An ideal execution of proceeds as follows:

- Inputs: each party obtains an input (the i^{th} party's input is denoted by x_i, $i \in \{1, 2\}$). The adversary $\mathcal{A}$ receives an auxiliary input denoted by aux.
- Sending inputs to the trusted party: any honest party P_i sends its received input x_i to the trusted third party TTP. The corrupted party $\mathcal{C}$ controlled by $\mathcal{A}$ may either abort, send its received input, or some other input to the trusted party. This decision is made by $\mathcal{A}$ and may depend on the value x_j for $j \in \mathcal{C}$ and its auxiliary input aux. Denote the vector of inputs sent to the trusted party by $\overline{\omega}$=(ω_1, ω_2). Notice that $\overline{\omega}$ does not necessary equal $\overline{x}$=(x_1, x_2); If the trusted party does not receive valid messages, then it replies two parties with a special symbol $\perp$ and the ideal execution terminates; otherwise, the execution proceeds to the next step.
- TTP sends its output to $\mathcal{A}$: TTP computes $f(\overline{\omega})$ and sends $f_i(\overline{\omega})$ to the party P$_i$, for $i \in \mathcal{C}$;
- $\mathcal{A}$ instructs TTP to "continue" or "halt": $\mathcal{A}$ sends either "continue" or "halt" to the trusted party. If it sends "continue", the trusted party sends $f_j(\overline{\omega})$ to P_j, for $P_j \notin \mathcal{C}$. Otherwise, if it sends "halt", TTP sends $\perp$ to P$_j$, for $P_j \notin \mathcal{C}$.
- Outputs: an honest party always outputs the message it obtained from the trusted party. The corrupted party outputs nothing. The adversary $\mathcal{A}$ outputs any arbitrary function of the initial input x_i and the message $f_i(\overline{\omega})$, $i \in \mathcal{C}$ obtained from the trusted party.

The ideal execution of f on inputs $\overline{x}$, auxiliary input aux to $\mathcal{A}$ and security parameter k, denoted by IDEAL$_{f,\mathcal{A}(aux),\mathcal{C}}(\overline{x}, k)$ is defined as the output vector of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model: The adversary sends all messages in place of the corrupted parity, and may follow an arbitrary polynomial-time strategy. In contrast, the honest party follows the instructions of the protocol π for computing f. The real execution of π on input $\overline{x}$, auxiliary information aux to $\mathcal{A}$ and security parameter k, denoted by REAL$_{\pi,\mathcal{A}(aux),\mathcal{C}}(\overline{x}, k)$ is defined as the output vector of the honest party and the adversary $\mathcal{A}$ from the real execution of π.

DEFINITION 2. *A protocol π securely computes a two-party function f with abort if for every non-uniform probabilistic polynomial time machine $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial time machine $\mathcal{S}$ for the ideal model, such that for every $\mathcal{C} \subseteq \{P_1, P_2\}$, every balanced vector $\overline{x}$ (for every i and j, $|x_i|$ =$|x_j|$), and every auxiliary input $z \in \{0, 1\}^*$:*

$$\{IDEAL_{f,\mathcal{S},\mathcal{C}}(\overline{x}, k)\}_{k\in N} \approx \{REAL_{\pi,\mathcal{A}(aux),\mathcal{C}}(\overline{x}, k)\}_{k\in N}$$

where $\approx$ indicates computational indistinguishability.

DEFINITION 3. *Let $\mathcal{F}_{\rm pdd}$ be private date deduplication functionality and π be an implementation of $\mathcal{F}_{\rm pdd}$. We π securely realizes the private data deduplication functionality $\mathcal{F}_{\rm pdd}$ if it securely computes $\mathcal{F}_{\rm pdd}$.*

3. DEDUPLICATION PROTOCOL: DESCRIPTION AND SECURITY ANALYSIS

Following [7], we assume that a server typically has to handle a huge number of files and the files themselves are stored on a secondary storage with a large access time. The server can store only a small amount of data per file in fast storage but it cannot afford to retrieve the file or parts of it from secondary storage upon every upload request. As a result, the private data deduplication scheme must allow the server to store only an extremely short information per file that will enable it to check claims from clients that they have that file without having to fetch the file contents for verification.

3.1 Notations and cryptographic assumptions

Collision-resistant hash functions: Informally, a collision-resilient hash function is a polynomial time computable function H mapping binary strings of arbitrary length into reasonably short ones, so that it is computationally infeasible to find any collision, that is any two different strings x and y for which $H(x)$=$H(y)$. In this paper, we assume that H: $\{0, 1\}^* \leftarrow \{0, 1\}^\lambda$. A hash function H: $\{0, 1\}^* \leftarrow \{0, 1\}^\lambda$ is called a random oracle if the distribution of $H(x)$ is uniformly distributed.

Merkle-tree: We define the parent of a vertex v, PARENT(v) as follows: PARENT$(vb) = v$ for any bit b. We also say vb ($v0$ or $v1$) is a child of v. We denote by T_k to be a binary tree with at most 2^k leaves at level k. We identify the vertices of T_k's by their labels, e.g., given a leaf $x = x_1 \cdots x_n$, the path from the root ϵ to x is ϵ, x_1, x_1x_2, $\cdots$, $x_1 \cdots x_n = x$.

Given a collision-resilient hash function H, a subtree T of T_k is turned into a Merkle-tree M$_{H,\lambda}(X)$ by storing in every node v of T_k a value V_v in the following manner[10]: any childless node can store any non-empty string, but any other node must store the value $H(ab)$ whenever its left child stores a and its right child stores b, that is $v = H(v0v1)$.

The discrete logarithm problem [4]: Let p be a large prime number such that $p = 2q + 1$, where p, q are two prime numbers. Let $G \in Z_p^*$ be a cyclic group of prime order

q and g is assumed to be a generator of G. The discrete logarithm problem is defined below:

- input: $g \in G$, $h \in_r G$
- output: $\log_g(h)$

An algorithm that solves the the discrete logarithm problem is a probabilistic polynomial time Turing machine, on input $g \in G$, $h \in_r G$, outputs $\log_g(h)$ with non-negligible probability. The discrete logarithm assumption means that there is no such a probabilistic polynomial time Turing machine. This assumption is believed to be true for many cyclic groups, such as the prime sub-group of the multiplicative group of finite fields.

3.2 A description of deduplication storage protocol

System parameters: Let p and q be two large prime numbers such that $p = 2q + 1$ (such a p is called a safe prime number). Let $G \subset Z_p$ be cyclic group with order q. Let $g_1, \ldots, g_m$ be m generators of G. Let H be a collision resistant hash function with output length of λ-bit. Let $\mathrm{MT}_{H,\lambda}(X)$ be the binary Merkle-tree over buffer X using λ-bit leaves and the hash function H.

Let E: $\{0,1\}^M \rightarrow \{0,1\}^{M'}$ be an erasure code, resilient to erasure of up to α fraction of the bits (for some constant $\alpha > 0$). Namely, form any $(1-\alpha)M'$ bits of $E(F)$ it is possible in principle to completely recover the original $F \in \{0,1\}^M$. Let X= $E(F)$, where $X = \{B_1, \ldots, B_s\}$ and $B_i = (B_{i,1}, \ldots, B_{i,m})$.

Computations of summary string: Given X, the prover P computes a committed Merkle-tree $\mathrm{MT}_{H,\lambda}(X)$ as follows (a committed Merkle-tree is a Merkle-tree, where each leaf value v_i is a commitment)

- assigning data block $B_i = (B_{i,1}, \ldots, B_{i,m})$ to a leaf node v_i: Let $C_i = g_1^{B_{i,1}} \ldots g_m^{B_{i,m}} \bmod p$ be a commitment of B_i (notice that $B_{i,j} \in Z_q$, $j = 1, \ldots, m$). Let v_i =C_i for the ith leaf value.
- assigning value to an internal node v_i: let v_i =$H(v_{i0}, v_{i1})$ where v_{ib} is a child of node v_i, $b \in \{0,1\}$ and v_{ib} is node value of v_{ib}. Let v be the root value of the tree.

Notice that the commitment scheme in our model is a deterministic algorithm, it follows that the summary string v depends only on the original data F.

Interactive proof system: To prove ownership of $< v, \mathcal{E}_{pk}(F) >$, P and S runs the following interactive proof protocol (*notice that the following sub-routine will be invoked u times on leaves indexes* $l_1, \ldots, l_u$)

- S chooses a random $i \in [s]$ and sends the leaf index i to P;
- P provides a sibling path from the selected leaf index i with value C_i and then proves its knowledge $B_i = (B_{i,1}, \ldots, B_{i,m})$ such that
$$C_i = g_1^{B_{i,1}} \ldots g_m^{B_{i,m}} \bmod p$$
according to the following procedure

1) P selects $\widetilde{B}_i = (\widetilde{B}_{i,1}, \ldots, \widetilde{B}_{i,m})$ at random, and computes $\widetilde{C}_i = g_1^{\widetilde{B}_{i,1}} \ldots g_m^{\widetilde{B}_{i,m}}$. P sends $\widetilde{C}_i$ to S;

2) Upon receiving $\widetilde{C}_i$, S selects $e \in Z_q$ uniformly at random and sends e to P;

3) Upon receiving e, P computes $B'_{i,1} = \widetilde{B}_{i,1} + eB_{i,1} \bmod q$ $(i = 1, \ldots, m)$ and sends $B'_i = (B'_{i,1}, \cdots, B'_{i,m})$ to S;

4) S outputs 1 if $g_1^{B'_{i,1}} \ldots g_m^{B'_{i,m}} = \widetilde{C}_i\ C_i^e \bmod p$.

This ends the description of private data deduplication protocol.

3.3 Correctness and soundness for our knowledge proof system

We want to show that our protocol π is a proof of knowledge (i.e., there exists a knowledge extractor K^P such that K^P outputs all data blocks $B_1, \ldots, B_s$ stored in the committed Merkle-tree with non-negligible probability in the presence of malicious adversaries specified in Section 2).

Correctness: One can verify that if a honest prover P who holds F (and hence it knows all data blocks $B_i = (B_{i,1}, \ldots, B_{i,m})$, $i \in [s]$) and a honest verifier S follow the protocol π, then the honest verifier always accepts. As a result, the protocol π is complete.

Knowledge extractor: We want to show that our protocol is a proof of knowledge. That is, assuming that a prover P succeeds in answering u leaves $\{l_1, \ldots, l_u\}$ with probability $(1-\alpha)^u + \delta$ for a fixed X and H, then there exists a black-box extractor K^P such that K^P outputs all data blocks $B_1, \ldots, B_s$ stored in the committed Merkle-tree with non-negligible probability. Our construction of the knowledge extractor K^P is almost same as that presented in [7] (the only difference is that the knowledge extractor defined in this paper works in the committed Merkle-tree while the knowledge extractor presented in [7] works in the plain Merkle-tree):

PROOF. Let $P^*(l_1, \ldots, l_u)$ be a prover that succeeds in answering u leaves $\{l_1, \ldots, l_u\}$ with probability $(1-\alpha)^u + \delta$ for a fixed X and H. Let $e_{P^*}(l_1, \ldots, l_u)$ be an event that P^* replies with sibling paths from all the leaves when queried on leaf indexes $l_1, \ldots, l_u$ and $e_{P^*}(l_1, \ldots, l_u) = 1$ if and only if $P^*(l_1, \ldots, l_u)$ is valid, i.e., P^* replies with valid sibling paths from all the leaves when queried on leaf indexes $l_1, \ldots, l_u$ and $e_{P^*} = 0$ otherwise.

Let $[n] = [1, \ldots, n]$. For a leaf index $l \in [s]$, and a query index $i \in [u]$, let $e_i(l)(l_1, \ldots, l_u)$ be an event $e_{P^*}(l_1, \ldots, l_u)$ where the i-th index is l while the others are chosen at random. We denote $e_i(l)(l_1, \ldots, l_u) = 1$ if and only if P^* replies with valid sibling paths from all the leaves when queried on random leaf indexes $l_1, \ldots, l_u$, where ith index l_i is replaced by l.

Let E_i be the event that $e_i(l)(l_1, \ldots, l_u) = 1$ happens with probability $\Pr_{l_1,\ldots,l_u}[e_i(l)(l_1, \ldots, l_u) = 1] \geq \frac{\delta}{u}$. For each $i \in [u]$, we define $E_i^+ = \{l \in [s]\colon E_i \text{ happens}\}$. We now show that $|E_i^+| \geq (1-\alpha)s$. By contradiction, if not then we have
$\Pr_{l_1,\ldots,l_u}\ [e_{P^*}(l_1, \ldots, l_u) = 1]$
$= \Pr_{l_1,\ldots,l_u}\ [e_{P^*}(l_1, \ldots, l_u) = 1 \wedge \text{ all } l_i \in E_i]$
$+ \Pr_{l_1,\ldots,l_u}\ [e_{P^*}(l_1, \ldots, l_u) = 1 \wedge \text{ not all } l_i \in E_i]$
$\leq \Pr[\text{ all } l_i \in E_i^+] + \sum_{i=1}^u \Pr[e_{P^*}(l_1, \ldots, l_u) = 1 \wedge l_i \notin E_i^+]$
$< (1-\alpha)^u + u\frac{\delta}{u} = (1-\alpha)^u + u$

Notice that we assume that P^* succeeds in answering u leaves $\{l_1, \ldots, l_u\}$ with probability $(1-\alpha)^u + \delta$ for a fixed X and H. As a result, we have contradiction. This means that $|E_i^+| \geq (1-\alpha)s$.

The knowledge extractor $K^{\mathcal{O}}$ is defined below

1. for $i = 1$ to u and l =1 to s
2. repeat $\tau = \lceil u(\log(s)+1) \rceil$
   - choose $l_1, \ldots, l_u \in [s]$ uniformly at random
   - query $P^*(l_1, \ldots, l_{i-1}, l, l_{i+1}, \ldots, l_u)$
3. output sibling paths for all the leaves

One can check if $l \in E_i^+$, then during the loop with i, l, the prover will return a valid answer with probability at least $1 - e^{-\tau}$. Hence we arrive that the index i for which $|E_i^+| \geq (1-\alpha)s$, we will find a valid sibling paths for an $(1-\alpha)$-fraction of the leaves with probability at least $(1 - 1/e)$.

Given $C_{l_1}, \ldots, C_{l_{(1-\alpha)s}}$ commitments stored in the Merkle-tree, if we are able to extract all these committed values $B_{l_1}, \ldots, B_{l_{(1-\alpha)s}}$ (*this statement is true, see Soundness below*), then we are able to recover the whole file F since we assume that the erasure code E resistant up to α-fraction of the bits. $\square$

Soundness: The protocol π securely computes the deduplication functionality $\mathcal{F}_{\text{pdd}}$ in the presence of malicious prover P assuming that the discrete logarithm problem defined over G is hard, H is collision-resistant and the underlying erasure code E can erasure up to α-fraction of the bits in the presence of malicious adversaries (in the Definition 2).

Proof. Let C_i be a value stored at a leaf n_i. Let $\widetilde{C}_i$ be the initial message generated by the adversary $\mathcal{A}$ on behalf of P; Upon receiving $\widetilde{C}_i$, the simulator $\mathcal{S}$ randomly chooses a string $e \in Z_q$ as a challenging message and gets a response message $B_i' = (B_{i,1}', \ldots, B_{i,m}')$ such that $g_1^{B_{i,1}'} \ldots g_m^{B_{i,m}'} = \widetilde{C}_i$ C_i^e mod p.

The simulator $\mathcal{S}$ rewinds the adversarial P by sending a different challenging message $e' \neq e$. Let $B_i^* = (B_{i,1}^*, \ldots, B_{i,m}^*)$ be a corresponding response message such that $g_1^{B_{i,1}^*} \ldots g_m^{B_{i,m}^*}$ $= \widetilde{C}_i$ $C_i^{e^*}$ mod p. $\mathcal{S}$ computes blocks $B_{i,1}, \ldots, B_{i,m}$ committed in the leaf n_i by the following procedure $(j = 1, \ldots, m)$

- computing $\Delta_{i,j} = B_{i,j}' - B_{i,j}^*$ mod q;
- extracting $B_{i,j} \leftarrow \frac{\Delta_{i,j}}{e-e^*}$ mod q.

The extracted blocks $B_i = (B_{i,1}, \ldots, B_{i,m})$ are then sent to the deduplication functionality $\mathcal{F}_{\text{pdd}}$. We now argue that the extracted blocks $B_i = (B_{i,1}, \ldots, B_{i,m})$ must be an input value of P; otherwise, we can use P to attack the discrete logarithm problem. More precisely, let g be a random generator of $G \subset Z_p$, where $p = 2q+1$, p and q are two large prime number. Let g_i $=g^{x_i}$ mod p. Let $C \in G$ be a target element chosen uniformly at random. Let $C_i \leftarrow C$ (the node value is assumed to be the target element C). For different challenge messages e and $e^* \neq e$, we obtain the corresponding response messages $B_i' = (B_{i,1}', \ldots, B_{i,m}')$ such that $g_1^{B_{i,1}'} \ldots g_m^{B_{i,m}'} = \widetilde{C}_i$ C_i^e mod p and $B_i^* = (B_{i,1}^*, \ldots, B_{i,m}^*)$ such that $g_1^{B_{i,1}^*} \ldots g_m^{B_{i,m}^*} = \widetilde{C}_i$ $C_i^{e^*}$ mod p. Consequently, we have the following equation

$$g_1^{\Delta_{i,1}} \ldots g_m^{\Delta_{i,m}} = C^{e-e^*} \bmod p$$

It follows that

$$x_1\Delta_{i,1} + \cdots + x_m\Delta_{i,m} = \log_g(C)(e-e^*) \bmod q$$

As a result,

$$\log_g(C) = \frac{x_1\Delta_1 + \cdots + x_m\Delta_m}{(e-e^*)} \bmod q$$

We thus arrive at the contradiction (the hardness assumption of the discrete logarithm problem defined over $G \subset Z_p$). $\square$

Theorem: The proposed private data deduplication protocol is provably secure assuming that the underlying hash function is collision-resilient, the discrete logarithm is hard and the underlying erasure code E can erasure up to α-fraction of the bits in the presence of malicious adversaries.

Proof. Let party P be a corrupted prover. We now define the following simulator for the ideal-world computation below. Whenever the simulator S obtains a transcript t of a proof that is accepted by the honest verifier S, it rewinds the corrupted P to obtain an input of corrupted party P (this is guaranteed by the soundness of the protocol assuming that the discrete logarithm problem is hard and the hash function is collision-resilient). Let F^* be an extracted file using the *knowledge extractor algorithm* $K^{\mathcal{O}}$. The simulator $\mathcal{S}$ then sends F^* to the private data deduplication functionality $\mathcal{F}_{\text{pdd}}$.

From the simulator described above (described in the soundness and knowledge extractor proofs), one can verify that $\{\text{IDEAL}_{\mathcal{F}_{\text{pdd}},\mathcal{S},\mathcal{C}}(\overline{x}, k)\}_{k \in N}$ is computationally indistinguishable from $\{\text{REAL}_{\pi,\mathcal{A}(aux),\mathcal{C}}(\overline{x}, k)\}_{k \in N}$. $\square$

4. CONCLUSION

In this paper, a new notion which we call private data deduplication protocols is introduced and formalized in the context of two-party computations. A feasible result of private data deduplication protocols has been proposed and analyzed. We have shown that the proposed private data deduplication protocol is provably secure in the simulation-based framework assuming that the underlying hash function is collision-resilient, the discrete logarithm is hard and the erasure coding algorithm E can erasure up to α-fraction of the bits in the presence of malicious adversaries.

5. REFERENCES

[1] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, Dawn Xiaodong Song: Provable data possession at untrusted stores. ACM Conference on Computer and Communications Security 2007: 598-609

[2] Ran Canetti: Universally Composable Security: A New Paradigm for Cryptographic Protocols. FOCS 2001: 136-145.

[3] Ernesto Damiani, Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, Pierangela Samarati: Key management for multi-user encrypted databases. StorageSS 2005: 74-83

[4] Whitfield Diffie and Martin Hellman. New directions in cryptography. IEEE Transactions on Information Theory, IT No.2(6):644-654, November 1976.

[5] Oded Goldreich, Foundations of Cryptography, Cambridge University Press, 2001.

[6] Oded Goldreich, Foundations of Cryptography, Cambridge University Press, 2004.

[7] Shai Halevi, Danny Harnik, Benny Pinkas and Alexandra Shulman-Peleg: Proof of ownership in remote storage systems. eprint, IACR, 2011/207.

[8] Danny Harnik, Benny Pinkas, Alexandra Shulman-Peleg: Side Channels in Cloud Services: Deduplication in Cloud Storage. IEEE Security & Privacy 8(6): 40-47, 2010

[9] Ari Juels, Burton S. Kaliski Jr.: Pors: proofs of retrievability for large files. ACM Conference on Computer and Communications Security 2007: 584-597.

[10] Ralph C. Merkle: A Certified Digital Signature. CRYPTO 1989: 218-238.

[11] Dave Russell: Data Deduplication Will Be Even Bigger in 2010, Gartner, 8 February 2010

[12] Hovav Shacham, Brent Waters: Compact Proofs of Retrievability. ASIACRYPT 2008: 90-107

[13] Qian Wang, Cong Wang, Jin Li, Kui Ren, Wenjing Lou: Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing. ESORICS 2009: 355-370