Secure Deduplication of Encrypted Data: Refined Model and New Constructions

Jian Liu[1], Li Duan[2], Yong Li[3], N. Asokan[1]

[1] Aalto University, Finland
jian.liu@aalto.fi, asokan@acm.org
[2] Paderborn University, Germany
liduan@mail.upb.de
[3] Ruhr-University Bochum, Germany
yong.li@rub.de

Abstract

Cloud providers tend to save storage via cross-user deduplication, while users who care about privacy tend to encrypt their files on client-side. Secure deduplication of encrypted data (SDoE) is an active research topic. In this paper, we propose a formal security model for this problem. We also propose two single-server SDoE protocols and prove their security in our model. We evaluate their deduplication effectiveness via simulations with realistic datasets.

1 Introduction

Cloud storage services are very popular. Providers of cloud storage services routinely use *cross-user deduplication* to save costs: if two or more users upload the same file, the storage provider stores only a single copy of the file. Users concerned about privacy of their data may prefer encrypting their files on client-side before uploading them to cloud storage. This thwarts deduplication since identical files are uploaded as completely different ciphertexts. Reconciling deduplication and encryption has been a very active research topic [3,10,4,15,19,13]. One proposed solution is *convergent encryption* (CE) [10] [4], which derives the file encryption key solely and deterministically from the file contents. As a result, identical files will always produce identical ciphertexts given identical public parameters. Unfortunately, a server compromised by the adversary can perform an *offline brute-force guessing attack* over the ciphertexts, due to the determinstic property of CE.

More recent solutions allow clients to encrypt their files using stronger encryption schemes while allowing the server to perform deduplication. They usually assume the presence of an independent (trusted) third parties [3] [15] [19]. However, in a cloud storage setting, like in many other commercial client-server contexts, assuming the presence of an indepdent third party is unjustified in practice [13] since it is unclear who can bear the costs of such third parties. Moreover, such schemes *cannot* prevent the *online brute-force guessing attacks* from a compromised active server.

Liu et al. proposed a single-server scheme for secure deduplication without the need for any third party [13]. Their scheme uses a per-file rate limiting strategy to prevent online brute-force guessing attacks by a compromised active server. However, their security model and proof only cover one round of the protocol (Section 9 in [14]). Consequently, their scheme is vunerable to additional attacks when considering the long-term operation of the system which involves multiple rounds of the protocol.

In this paper, we make the following contributions:

- We propose a **formal security model** for the single-server "secure deduplication of encrypted data" (SDoE) (Section 2). We claim that a deduplication scheme proved secure in this model can guarantee that, for a certain file, (1) a compromised client cannot learn whether or not this file has already been uploaded by someone else (Section 2.1), and (2) the only way for a compromised server to uniquely determine this file is by doing an online brute-force attack (Section 2.2).
- We propose two new **single-server deduplication** schemes and prove their security in our model. (Section 4)
- We show that their deduplication effectiveness is reasonable via **simulations with realistic datasets**. (Section 5)

2 Security Model

We model the threats with security games played between an adversary (attacker) and a challenger. The challenger possesses some secret targeted by the adversary. As in the real world, the adversary can interact with the challenger by using different queries to the challenger. At the end of each game, the adversary outputs what he has learned about the secret and he wins if his output is correct. The restriction on queries are used to rule out the trivial cases of breaking the security of the scheme.

2.1 Settings

We consider the generic setting for a cloud storage system where a set of clients ($\mathcal{C}$s) store their files on a single storage server ($\mathcal{S}$), and the $\mathcal{C}$s and $\mathcal{S}$ are always communicating through secured channels. The deduplication happens at server-side, i.e., the client will always upload the encrypted file and the server knows whether to discard the uploaded file or not after the protocol execution. All theses participants are generalized as *parties*. Each party has a party identifier *pid*; a flag τ indicating whether it is corrupted or not. Each $\mathcal{C}$ may have one or more sessions connecting to the $\mathcal{S}$, where each session has a session identifier *sid*.

The internal state $\Phi_{C,pid}$ of a $\mathcal{C}$ is a list of tuples $\{(\mathit{fid}_i, k_i)\}$, which stores the identifier and the encryption key of each file owned by it. The internal state Φ_S of $\mathcal{S}$ contains a list $\mathbf{DB} = \{(b_i, \mathit{fid}_i, \varrho_i, \mathbf{LO}_i)\}$ and a list of current user identifiers **PID**, where b_i is a bit indicating whether the file has been uploaded or not, fid_i

is an identifier of an encrypted file ϱ_i and $\mathbf{LO}_i$ is the list of owners of ϱ_i. Note that $\mathbf{DB}$ contains all possible files.

To initialize $\mathbf{DB}$ for the security game, the challenger first generate a list of file owners with the corresponding identifier list $\mathbf{PID}$. Let $\mathbf{PID}_h$ be the identifier set of honest file owners. Note that before interacting with adversaries, $\mathbf{PID} = \mathbf{PID}_h$, but $\mathcal{A}$ can add new malicious identities to $\mathbf{PID}$ by using $\mathsf{RegisterCorrupt}(pid)$ queries described bellow. Then for each fid_i, the challenger chooses $b_i \xleftarrow{\$} \{0,1\}$. If $b_i = 0$, the tuple would be $(0, fid_i, -, \emptyset, -)$. Otherwise, it involves a session of a $\mathcal{C}_j$ uniformly at random to upload F_i and then makes $\mathcal{C}_j$ modify its internal state accordingly. Afterwards the challenger stores the ciphertext C_i generated by $\mathcal{C}_j$ in the uploading process and adds $\mathcal{C}_j$ to $\mathbf{LO}_i$. We denote as $\mathbf{DB}_0$ the content of $\mathbf{DB}$ after initialization.

2.2 Security against a compromised client

Here, we want to model the attacks from a compromised client. The intuition is that by interacting with the server, the client must not be able to learn whether a file already exists in the cloud storage. We allow the adversary $\mathcal{A}$ that has compromised one or more clients to make the following types of oracle queries in the security experiments:

$\mathsf{RegisterCorrupt}(pid)$ The adversary $\mathcal{A}$ can register a (new) client with identifier pid and $\tau = Corrupted$. If $pid \in \mathbf{PID}$, $\mathcal{A}$ gets the state $\Phi_{C,pid}$ of pid, including all the file identifiers and the corresponding file key $\{(fid_i, k_i)\}$ owned by pid. Otherwise $\mathcal{A}$ only gets an empty state. The challenger updates $\mathbf{PID} := \mathbf{PID} \cup \{pid\}$ and marks pid as corrupted. In both cases, $\mathcal{A}$ can then perfectly impersonate pid from this moment on.

$\mathsf{Send}(pid, sid, M)$ The corresponding oracle computes on the input message M following the SDoE protocol and returns the output message in the view of all corrupted parties to $\mathcal{A}$. This oracle models any single step of the SDoE protocol.

$\mathsf{Test}()$ $\mathcal{A}$ signals the end of the security game to the challenger, ceases all the interaction with oracles, and outputs a pair (F^*, b^*).

Let λ be the security parameter. Given the queries described above, we define the security experiment $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda)$ for a SDoE protocol Π against compromised clients as follows: $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda) = 1$ if $\mathcal{A}$ replies to $\mathsf{Test}()$ with (F^*, b^*) and either of the following case happens:

- If $b^* = 0$ and $\nexists (fid_j, k_j) \in \bigcup_{pid \in PID_h} \Phi_{C,pid}$, $(b_i, fid_i, C_i, \mathbf{LO}_i) \in \mathbf{DB}_0$ s.t., $E(k_j, F^*) = C_i$. (i.e., F^* has not been uploaded before).
- If $b^* = 1$ and $\exists (fid_j, k_j) \in \bigcup_{pid \in PID_h} \Phi_{C,pid}$, $(b_i, fid_i, C_i, \mathbf{LO}_i) \in \mathbf{DB}_0$ s.t., $E(k_j, F^*) = C_i$. (i.e., F^* has been uploaded before).

But *none* of the following events happens before $\mathcal{A}$ outputs (F^*, b^*):

- $\mathcal{A}$ has issued $\mathsf{RegisterCorrupt}(pid)$ with $pid \in \mathbf{PID}_h$, i.e., $\mathcal{A}$ cannot impersonate honest file owners.

– $\mathcal{A}$ has issued $\mathsf{Send}(pid, sid, M)$ with $pid \in \mathbf{PID}_h$. ($\mathcal{A}$ cannot force an honest owner to send any messages.)

Definition 1. *We define the advantage of an adversary $\mathcal{A}$ in the experiment $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda)$ as*

$$\mathbf{Adv}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi}(\lambda) = 1] - \frac{1}{2}$$

2.3 Security against a compromised server

The intuition behind the security definition is that a SDoE scheme is secure against a compromised server, if a file can only be uniquely determined by the compromised server through an online brute-force attack for all candidate files. The queries (adversary's ability) captures the essence of concrete attacks, such as registering malicious clients, uploading and tampering with some messages. We allow the adversary $\mathcal{A}$ that has compromised the server to make the following types of queries in the security experiments:

$\mathsf{RegisterCorrupt}(pid, \theta)$ The same as that of compromised clients.
$\mathsf{Send}(pid, sid, M)$ The same as that of compromised clients.
$\mathsf{AccessDB}()$ The adversary $\mathcal{A}$ gets all the C_i and the owner list of each C_i in Φ_S with $b_i = 1$. If this is the t_a-th query made by $\mathcal{A}$, then for all the t-th queries with $t > t_a$, $\mathcal{A}$ also gets the updated Φ_s items with $b_i = 1$ in addition to the response for other queries.
$\mathsf{Execute}(pid, P, F)$ As the initiator, $\mathcal{A}$ invokes a complete (sub-)protocol P with party pid on the input file F and obtains all the messages exchanged, following the description of P.
$\mathsf{Test}()$ $\mathcal{A}$ outputs two files F_0, F_1 with equal length. Upon receiving F_0, F_1, the challenger chooses $b \xleftarrow{\$} \{0,1\}$ and replies with a ciphertext $C_b = Enc(k_{f_b}, F_b)$. $\mathcal{A}$ performs the above queries and then outputs a bit b'.

We define the security experiment $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda)$ for a SDoE protocol Π against partially compromised server as follows: $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda) = 1$ if $\mathcal{S}$ replies to $\mathsf{Test}()$ with $b' = b$, but *none* of the following events happens before $\mathcal{A}$ outputs the bit b':

– $\mathcal{A}$ has issued $\mathsf{RegisterCorrupt}(pid)$ with $pid \in \mathbf{PID}_h$.
– $\mathcal{A}$ has issued $\mathsf{Execute}(pid, P, F), F \in \{F_0, F_1\}$. ($\mathcal{A}$ have not included F_0 or F_1 in its online butte-force attacks.)
– $\mathcal{A}$ has issued $\mathsf{Send}(pid, sid, M)$ with $pid \in \mathbf{PID}_h$.

Definition 2. *We define the advantage of an adversary $\mathcal{A}$ in the experiment $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda)$ as*

$$\mathbf{Adv}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda) = \Pr[\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi}(\lambda) = 1] - \frac{1}{2}$$

3 PAKE based Deduplication

Bellovin and Merritt [7] proposed a *password authenticated key exchange* (PAKE) protocol to against offline brute-force attacks even through users choose low-entropy passwords. PAKE enables two parties to set up a session key iff they hold the same secret ("password"). Otherwise, neither party can learn anything about the key output by the other party.

Bellare et al. provided a game-based definition for the security of PAKE [5]. A random bit b is chosen at the beginning of the game. They assume that there is an adversary $\mathcal{A}$ that has complete control over the environment (mainly, the network), and is allowed to query the following oracles:

$\mathsf{Send}(U_i, M)$: causes message M to be sent to instance U_i, which computes following the protocol and gives the result to $\mathcal{A}$. If this query causes U_i to accept or terminate, this will also be shown to $\mathcal{A}$.

$\mathsf{Execute}(A_i, B_j)$: causes the protocol to be executed to completion between A_i and B_j, and outputs the transcript of the execution.

$\mathsf{Reveal}(U_i)$: output k_{U_i}, which is the session key held by U_i.

$\mathsf{Test}(U_i)$: if $b = 1$, output the session key k_{U_i}; otherwise, output a string drawn uniformly from the space of session keys.

$\mathsf{Corrupt}(U_i)$: output U_i's password.

Let $\mathsf{Succ}_{\mathcal{A}}^{\mathsf{PAKE}}(\lambda)$ be the event that $\mathcal{A}$ outputs a bit $b' = b$ but none of the following events happens:

1. a $\mathsf{Reveal}(U_i)$ query occurs;
2. a $\mathsf{Reveal}(U_j)$ query occurs where U_j is the partner of U_i;
3. a $\mathsf{Corrupt}(U_i)$ query occurs before U_i defined its key k_{U_i} and a $\mathsf{Send}(U_i, M)$ query occurred.

The advantage of $\mathcal{A}$ attacking a PAKE protocol is defined to be

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PAKE}}(\lambda) \stackrel{def}{=} 2\Pr[\mathsf{Succ}_{\mathcal{A}}^{\mathsf{PAKE}}(\lambda)] - 1.$$

The PAKE protocol is considered secure if passwords are uniformly and independently drawn from a dictionary of size n:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PAKE}}(\lambda) \leq \frac{n_{se}}{n} + \mathsf{negl}(\lambda),$$

where n_{se} is the number of Send queries (to distinct instances U_i). The intuition behind this definition is that only online brute-force attacks are allowed in a secure PAKE protocol.

PAKE-based SDoE. Liu et al. present a PAKE-based SDoE scheme that does not depend on any additional independent servers [13]. Their scheme allows an uploader to securely obtain the decryption key of another user who has previously uploaded the same file. Specifically, the uploader $\mathcal{C}$ first sends a short hash of its file (10-20 bits long) to $\mathcal{S}$. $\mathcal{S}$ finds other clients who may hold the same files based on the short hash, and lets them run a single round PAKE protocol

(routed through $\mathcal{S}$) with the long hashes of their files as inputs. At the end of the protocol, the uploader gets the key of another $\mathcal{C}$ if and only if they indeed hold the same file. Otherwise, it gets a random key. The PAKE-based SDoE scheme Π_{PAKE} is shown in Figure 1.

$\mathcal{C}$s protect themselves against online brute-force attacks by limiting the number of PAKE instances they will participate in for each file.

Security against compromised clients. As pointed out by Liu et al. themselves in [14], additional attacks are possible when considering the long-term operation of the system. For example, a malicious client can upload a file and then pretend to be offline. Later it uploads the same file using another identity. If it gets the the same key as the one it got before, it knows that the file has been uploaded be someone else. Another attack is also targeting the PAKE phase. The adversary uploads a file F with the identity of $\mathcal{C}_1$ in the first protocol run. It then uses a different identity $\mathcal{C}_2$ to upload F again. By observing whether $\mathcal{C}_1$ is involved in the PAKE phase with $\mathcal{C}_2$ for F, the adversary knows if there are other owners of F.

In the next section, we will introduce two protocols that are immune to those attacks and prove their security under our new model.

4 New SDoE Schemes

Recall that in Π_{PAKE}, there are two possible cases when an uploader uploads a file: it either gets the key of a previous uploader of the same file or gets a new random key. As described in the previous section, a malicious $\mathcal{C}$ can distinguish between these two cases.

In this section, we address the issue in Π_{PAKE} by having $\mathcal{C}$s always get random keys when they upload their files. We propose two schemes. The first scheme ($\Pi_{\mathsf{PAKE,re\text{-}enc}}$) borrows the idea from proxy re-encryption [1]. $\mathcal{S}$ only keeps a single copy of duplicated files. When $\mathcal{C}$ wants to download its file, $\mathcal{S}$ re-encrypts the file so that $\mathcal{C}$ will download the same ciphertext as the one it uploaded. However, this scheme requires public-key operations on the entire file, which is not efficient for large files. So we propose a second scheme ($\Pi_{\mathsf{PAKE,popular}}$) that only deduplicates popular files and only protects the privacy of unpopular files. For unpopular files, $\mathcal{C}$s get random keys and download the same ciphertexts as they uploaded. If those files become popular later, $\mathcal{S}$ deletes all duplicated copies and provides a way to help $\mathcal{C}$s to transform their keys to the right key.

4.1 PAKE-based Deduplication with Re-encryption

The first scheme $\Pi_{\mathsf{PAKE,re\text{-}enc}}$ is shown in Figure 2. It is similar to Π_{PAKE}. In the following description, the details of client authentication and file ownership authentication are omitted. We assume that the owners of each file are stored in an ordered list with respect to the upload time points. In the case that there are more than one owner of a candidate file, the newest checker is chosen by $\mathcal{S}$ for the PAKE phase. After PAKE, instead of masking k_{F_i} with k'_{iR}, $\mathcal{C}_i$ generates a

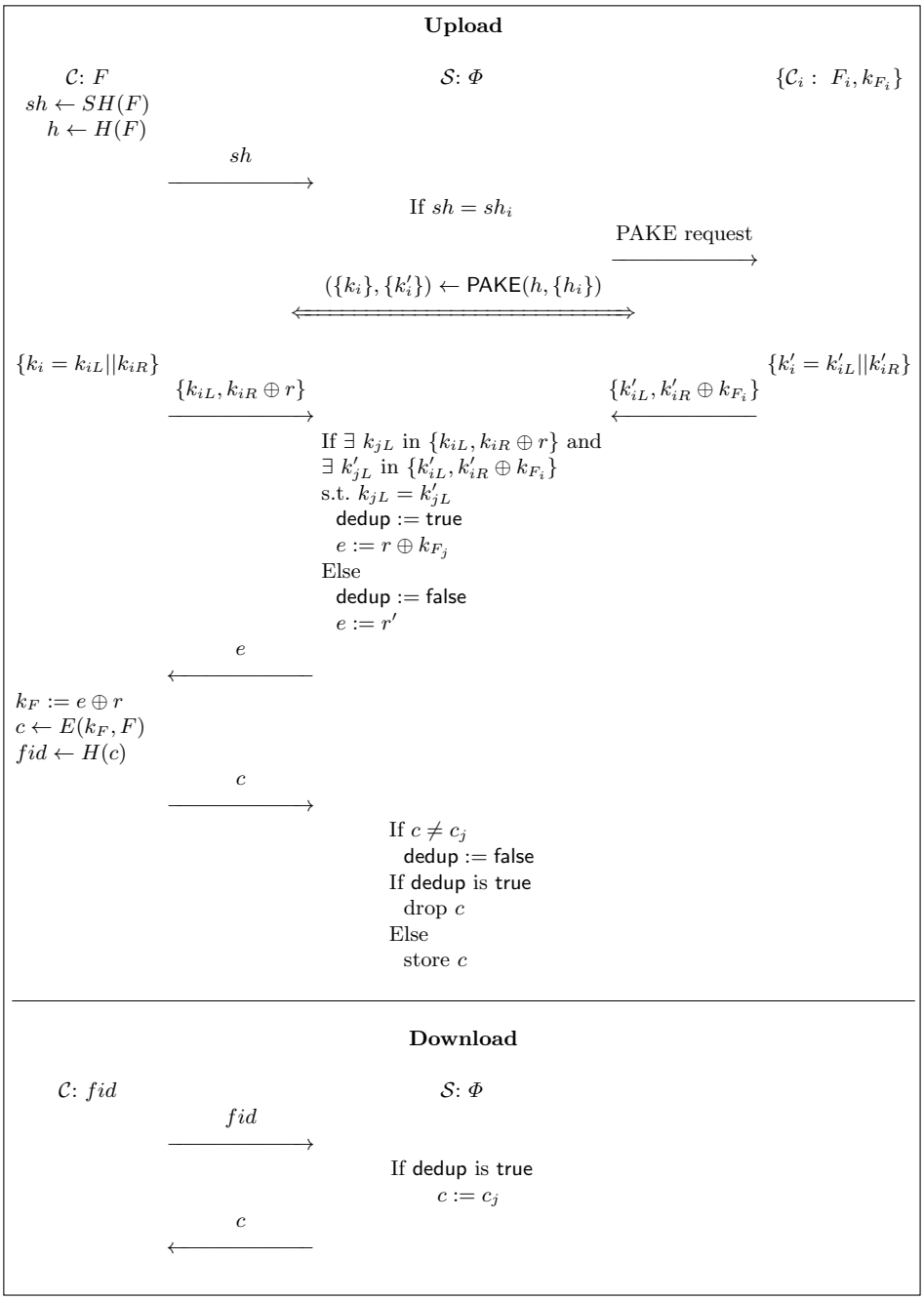


Fig. 1. PAKE-based deduplication scheme [13].

random number r_i, and masks both k_{F_i} and k'_{iR} with r_i. $\mathcal{C}$ only sends k_{iL} to $\mathcal{S}$. If there is an index j s.t. $k_{jL} = k'_{jL}$, $\mathcal{S}$ knows that $\mathcal{C}$ is uploading the same file with $\mathcal{C}_j$. Then, it keeps $(r_j - k_{F_j})$ and sends $(k'_{jR} + r_j)$ to $\mathcal{C}$. Otherwise, it sends a random number r'. $\mathcal{C}$ calculates its file key as $k_F := e - k_{jR}$ and then encrypts its file as $F \cdot g^{k_F}$. Notice that if F is detected to be duplicated, k_F is just the randomness r_j generated by $\mathcal{C}_j$. $\mathcal{S}$ can just drop this ciphertext if deduplication happens and stores the $fid = H(C)$ as an alias of the file. Later, when $\mathcal{C}$ wants to download F, $\mathcal{C}$ re-encrypts c_j to k_F: $c := c_j \cdot g^{r_j - k_{F_j}} = F \cdot g^{k_{F_j}} g^{r_j - k_{F_j}} = F \cdot g^{r_j} = F \cdot g^{k_F}$. Notice that c_j may be deduplicated already. In this case, $\mathcal{S}$ need to calculate $(r_0 - k_{F_0}) + (r_1 - k_{F_1}) +, ..., + (r_j - k_{F_j}) = (r_j - k_{F_0})$, and then transfer c_0 to $\mathcal{C}$'s ciphertext.

Security against compromised clients. Security of SDoE schemes cannot be directly reduced to the semantic security of PAKE schemes in [5]. This technical impossibility in the proof lies in the fact that the password (the hash of the file) is always known to the adversary in SDoE prior to any other interactions in the PAKE protocol. To overcome this difficulty, we expand the original definition of the model in [5] in the following way, which we call the constrained PAKE security game. Let $sk = sk_L || sk_R$ be the session key computed in the $\mathsf{Test}(i, s)$ session, where $|sk_L| = |sk_R| = \frac{1}{2}|sk|$.

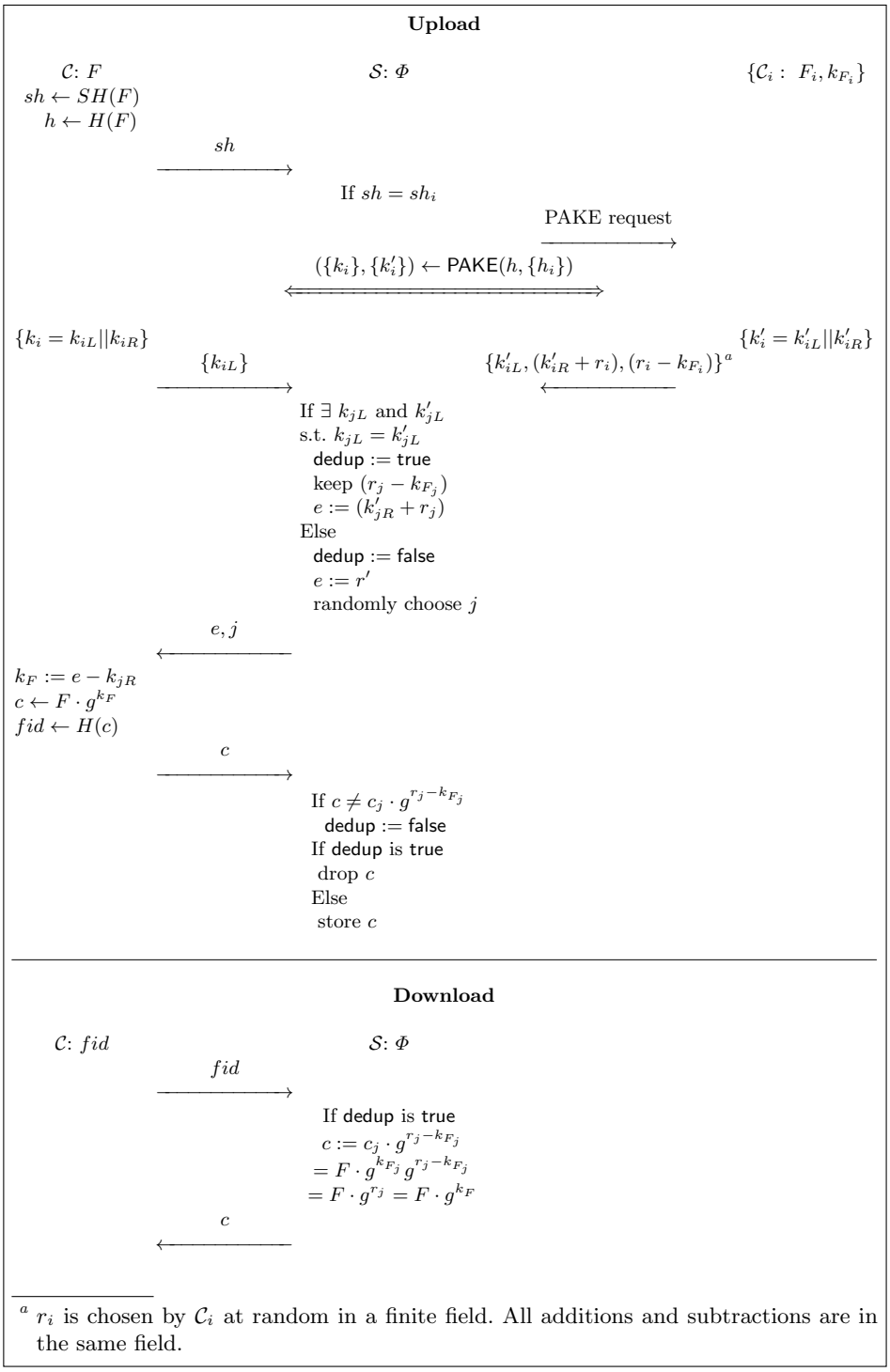
- The setup of this game is the same as in the original PAKE game except that each party now holds an additional secret $s_u \in \mathcal{K}$. A public function $f : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^{\frac{1}{2}|sk|}$ can be queried by the adversary as $f(p_i, \cdot)$.
- The $\mathsf{Test}(i, s)$-query now returns $tk = tk_L || tk_R$, where $|tk_L| = |tk_R| = \frac{1}{2}|sk|$. The first half of tk is always the same as the the first half of the real session key, i.e., $tk_L = sk_L$. If $b = 1$, $tk_R = sk_R \oplus f(s_i, T_i)$, where $T_{i,s}$ is the transcript of this session. Otherwise $tk_R = sk_R \oplus r$, where $r \xleftarrow{\$} \{0,1\}^{\frac{1}{2}|sk|}$. The adversary wins if she outputs $b' = b$.
- $\mathsf{Corrupt}(u)$ only returns the password PW_u but not the additional secret s_u.
- A session involving π_i^s and π_j^t is fresh if both the following condition holds
  - no $\mathsf{Reveal}(s, i)$ or $\mathsf{Reveal}(t, j)$ is made before $\mathsf{Test}(i, s)$.
  - no $f(p_i, T_{i,s})$ is made before $\mathsf{Test}(i, s)$.

The winning condition and the advantage of an adversary in a constrained-PAKE game is defined in the same way as in the PAKE game.

Theorem 1 *If there exists a ppt adversary $\mathcal{C}$ in $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C}, \Pi_{\mathsf{PAKE}, \text{re-enc}}}(\lambda)$ with advantage $\epsilon_{\mathcal{C}}$, then there also exists a ppt adversary $\mathcal{A}$ with advantage $\epsilon_{\mathcal{A}}$ in the underlying constrained-PAKE game against Π in the random oracle model such that*

$$\epsilon_{\mathcal{C}} \leq \frac{q_H}{2^{l_h}} + q_H \cdot \epsilon_{\mathcal{A}}$$

where Π is the PAKE oracle, l_{sh} the length of the short hash, l_h the length of the long hash and q_H is the distinct number of distinct files $\mathcal{C}$ has queried for short hash, hash or uploaded.



[a] r_i is chosen by $\mathcal{C}_i$ at random in a finite field. All additions and subtractions are in the same field.

Fig. 2. PAKE-based deduplication via ciphertext transformation.

Proof. We use the sequence of games technique introduced in [18]. Furthermore, we assume that the hash function is simulated by the challenger and all files are of equal length.
Game 0 . This is the original game $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{C},\Pi_{\mathsf{PAKE,re\text{-}enc}}}(\lambda)$.

$$\epsilon_{\mathcal{C}} = \mathbf{Adv}^{Game\ 0}_{\mathcal{C}}(\lambda) \tag{1}$$

Game 1. Let $\mathbf{F'} = \{F'_1, F'_2, \cdots, F'_{q_H}\}$ be the set of distinct files that $\mathcal{C}$ has issued H-queries or *check*()-queries before $\mathcal{C}$ queries $\mathsf{Challenge}()$. Let (F^*, b^*) be the output of $\mathcal{C}$.
If $\exists F'_j \in \mathbf{F'} : H(F'_i) = H(F^*) \wedge F_i^{'} \neq F^*$, abort the game. Therefore we have

$$\mathbf{Adv}^{Game\ 0}_{\mathcal{C}}(\lambda) \leq \frac{q_H}{2^{l_h}} + \mathbf{Adv}^{Game\ 1}_{\mathcal{C}}(\lambda) \tag{2}$$

This rule makes sure that no hash collision happens.
Game 2. Another abort rule is added in this game. The challenger makes a guess of an index $i \in \{1, \cdots, q_H\}$ and if $F^* \neq F'_i$, the challenger aborts the game. Thus we have

$$\mathbf{Adv}^{Game\ 1}_{\mathcal{C}}(\lambda) \leq q_H \cdot \mathbf{Adv}^{Game\ 2}_{\mathcal{C}}(\lambda) \tag{3}$$

Game 3. In this game, the random oracle in $\Pi_{\mathsf{PAKE,re\text{-}enc}}$ is replaced (implicitly) by the random process for password generation. More specifically, we implicitly define $H : \{0,1\}^* \rightarrow PW$, where PW is the password space and all the public parameters of Π (for example, group order and generator as in EKE2 in [6]) are included in the public parameters of $\Pi_{\mathsf{PAKE,re\text{-}enc}}$. This replacement has no impact on $\mathcal{C}$'s view since all passwords and parameters in Π are also sampled uniformly at random as required. Thus

$$\mathbf{Adv}^{Game\ 2}_{\mathcal{C}}(\lambda) = \mathbf{Adv}^{Game\ 3}_{\mathcal{C}}(\lambda) \tag{4}$$

We now construct an adversary $\mathcal{A}$ using $\mathcal{C}$ against the underlying PAKE scheme Π . Let d be the nmber of distinct passwords used by parties initialized by $\mathcal{A}$'s PAKE challenger, where $d \geq q_H$ [4]. When storing the files $\{F_1, \cdots, F_n\}, q_H \leq n \leq d$, $\mathcal{A}$ maps to those files n PAKE parties $P_1, \cdots, P_n$ with different passwords. $\mathcal{A}$ then sets up the list $\mathbf{PID}_h$ and $\mathbf{PID}$ as described in the model and binds the identifiers in $\mathbf{PID}$ to each file as owners randomly. Each file identifier fid_i and encryption key k_{F_i} are chosen according to the protocol definition and then used to build each $\Phi_{C,pid}$. Finally $\mathcal{A}$ stores all the $(b_i, fid_i, F_i, k_{F_i}, \mathbf{LO}_i)$ tuples as $\mathbf{DB}_0$.
Answer the Hash queries $H(F'_j)$.

1. $\mathcal{A}$ searches for $(F'_j, \{P'_j\})$.
2. If found, $\mathcal{A}$ issues $\mathsf{Corrupt}(P'_j)$ and let the output of $\mathsf{Corrupt}(P'_j)$ be $PW_{P'_j}$.
3. $\mathcal{A}$ returns $PW_{P'_j}$ to $\mathcal{C}$.

[4] we assume a polynomial sized file space.

Answer the $\mathsf{RegisterCorrupt}(pid)$. If $pid \notin \mathbf{PID}_h$, $\mathcal{A}$ simply enrols this pid in $\mathbf{PID}$. If $pid \in \mathbf{PID}_h$, $\mathcal{C}$ fails automatically.
Answer the $\mathsf{send}(pid, M)$. If $pid \notin \mathbf{PID}_h$, $\mathcal{A}$ answers the send-queries exactly as in the SDoE protocol. The hash-value or the PAKE messages to be returned are obtained in the same way as when answering hash queries.

During every PAKE-phase for uploading F_i, $\mathcal{A}$ uses Send to involve an oracle π_s^i run by party P_i. Afterwards, except for one session involving F^*, $\mathcal{A}$ uses Reveal on each accepted process $P_{i,s}$ of party P_i to get session key $k_{i,s}$.

Denote as P^* the PAKE party bound with F^*. For the PAKE test session T^*, $\mathcal{A}$ uses the PAKE Test queries to get a challenge session key $tk*$. Then $\mathcal{A}$ queries for $f(P^*, T^*)$ and computes $k_R^* = tk_R^* \oplus f(P^*, T^*)$. Finally $\mathcal{A}$ chooses an r^* and uses k_R^* as defined in SDoE protocol $\Pi_{\mathsf{PAKE,re\text{-}enc}}$.

Let i be the original index of F^* in $\mathbf{DB}_0$. $\mathcal{A}$ outputs 1 if $b^* = b_i$ and 0 otherwise. Note that if $b = 0$ in the constrained PAKE experiment, the right half of tk^* is random bit-strings and so is $k_R^* = tk_R^* \oplus f(P^*, T^*)$. As a consequence, e is also random. Therefore in this case, $\mathcal{C}$ also has no advantage. On the other hand, if $b = 1$, k_R^* is correctly distributed as in $\Pi_{\mathsf{PAKE,re\text{-}enc}}$. The probability $\mathcal{C}$ outputs the correct b^* is the same as $\mathcal{A}$ outputs the correct b. Thus we have

$$\epsilon_{\mathcal{A}} = \mathbf{Adv}_{\mathcal{C}}^{Game\ 3}(\lambda) \tag{5}$$

By combining (1) to (5), we have proved Theorem 1.

Security against compromised server. Next, we prove the security of $\Pi_{\mathsf{PAKE,re\text{-}enc}}$ against a compromised server, which leads to the following theorem.

Theorem 2 *If there exists a ppt adversary $\mathcal{S}$ in $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi_{\mathsf{PAKE,re\text{-}enc}}}(\lambda)$ with advantage $\epsilon_{\mathcal{S}}$ when $sh(F_0) = sh(F_1)$, then there also exist a ppt adversary $\mathcal{A}$ with advantage $\epsilon_{\mathcal{A}}$ in the underlying IND-KPA game against Π_{enc} in the random oracle model and a ppt but passive adversary $\mathcal{B}$ against the PAKE-protocol Π with advantage ϵ_B such that*

$$\epsilon_{\mathcal{S}} \leq \frac{2C \cdot \mathbf{N}_e}{|\mathbf{K}|} + \epsilon_B + \left(\frac{2^{l_{sh}}}{q_H}\right)^2 \left(2|\mathbf{F}|^2 \cdot \epsilon_{\mathcal{A}}\right)$$

where C is the maximal number of owners of each file, $\mathbf{K}$ the key space of Π_{enc}, N_e the number of $\mathsf{Execute}$ *queries, $\mathbf{F}$ the file space, l_{sh} the length of the short hash, q_H the number of distinct files that $\mathcal{S}$ has queried for its hash or short hash and Π_{enc} is the encryption scheme for files.*

Proof. First we consider two different cases for $\mathcal{S}$ to win.

1. $\mathcal{S}$ has issued $\mathsf{Execute}(pid, P, F)$ and seen at least one file keys collide into any of the file keys of the equivalent ciphertexts of F_0 or F_1.
2. $\mathcal{S}$ has not seen any colliding keys by issuing $\mathsf{Execute}(pid, P, F)$.

In case 1, since each $\mathsf{Execute}(pid, P, F)$ reveals at most one real file key $k \in \mathbf{K}$. On the other hand, since there are at most C owners of each file, each of whom

has an equivalent file key, seeing one key increases the probability of $\mathcal{S}$ by at most $\frac{C}{|\mathbf{K}|}$ to decrypt each F_b correctly. Let the advantage of $\mathcal{S}$ in case 2 be $\epsilon'_{\mathcal{S}}$. With the union bound we have

$$\epsilon_{\mathcal{S}} \leq \frac{2C \cdot \mathbf{N}_e}{|\mathbf{F}|} + \epsilon'_{\mathcal{S}} \tag{6}$$

To further analyze $\epsilon'_{\mathcal{S}}$, two types of adversaries are considered in the following proof.

1. Adversaries that recover the complete session key generated after the PAKE phase involving at least one honest client. We call these adversary as type 1 adversaries.
2. Adversaries do not recover any complete session keys generated by one honest clients but respond to the $\mathsf{Test}()$. We call these adversary as type 2 adversaries.

With a simple probability argument, it can be deducted that

$$\epsilon'_{\mathcal{S}} \leq \epsilon_1 + \epsilon_2 \tag{7}$$

where ϵ_1 is the advantage of type 1 adversary and ϵ_2 the advantage of type 2 adversary. Furthermore, we assume that the hash function is simulated by the challenger and all files are of equal length.

With the analysis above, we prove Theorem 2 by proving the following 2 lemmas.

Lemma 1 *(Bounding of the advantage of the type 1 adversary) If there exists any type 1 adversary $\mathcal{A}_1$ with advantage ϵ_1 and running time t_1, then there also exists a passive PAKE adversary $\mathcal{B}$ with advantage ϵ_B and running time $t_B \approx t_1$ such that $\epsilon_1 \leq \epsilon_B$.*

Proof. (lemma 1) $\mathcal{B}$ can give the transcript of the Test-session to $\mathcal{A}_1$ and obtain the session key sk_{A_1}recovered by $\mathcal{A}_1$. $\mathcal{B}$ then outputs ($sk_{A_1} = k_b$).

Note that in our protocol, if the session key is leaked to $\mathcal{A}_1$, then the encryption key k_F is also leaked to $\mathcal{A}_1$ and vise versa. The confidentiality of k_F is the basis of the remaining proof.

Lemma 2 *(Bounding of the advantage of the type 2 adversary) If there exists any type 2 adversary $\mathcal{A}_2$ with advantage ϵ_2 and running time t_2, then there also exists a IND-KPA adversary $\mathcal{A}$ with advantage ϵ_A and running time $t_A \approx t_2$ such that $\epsilon_2 \leq \left(\frac{2^{l_{sh}}}{q_H}\right)^2 \left(2|\mathbf{F}|^2 \cdot \epsilon_{\mathcal{A}}\right)$.*

Proof. (lemma 2) ***Game 0*** . This is the original game $\mathbf{Exp}^{\mathrm{SDoE}}_{\mathcal{S},\Pi_{\mathsf{PAKE,re\text{-}enc}}}(\lambda)$.

$$\epsilon_{\mathcal{A}_2} = \mathbf{Adv}^{Game\ 0}_{\mathcal{S}}(\lambda) \tag{8}$$

Game 1. If any of the files F_0, F_1 chosen by $\mathcal{S}$ when querying $\mathsf{Test}()$ has a unique short hash value, abort the game. We add this rule since in our protocol since the short hash is also stored as part of the ciphertext. If any $sh(F_j)$ is unique, $\mathcal{S}$ can simply learn F_j by computing and comparing the short hash values of all file candidates. Fix F_j and let **CollSH** be the event that $sh(F_j)$ does equal to some other $sh(F_i), F_i \in \mathbf{F}$. Then $\Pr[\mathbf{CollSH}] = \frac{q_H}{2^{l_{sh}}}$. Thus we have

$$\begin{aligned} &Pr[\exists F_i, F_k \in \mathbf{F}, F_i \neq F_0 \wedge F_k \neq F_1 : \\ &\quad sh(F_i) = sh(F_0) \wedge sh(F_k) = sh(F_1)] \geq \left(\frac{q_H}{2^{l_{sh}}}\right)^2 \end{aligned} \tag{9}$$

Therefore we have

$$\mathbf{Adv}_{\mathcal{S}}^{Game\ 0}(\lambda) \leq \left(\frac{2^{l_{sh}}}{q_H}\right)^2 \mathbf{Adv}_{\mathcal{S}}^{Game\ 1}(\lambda) \tag{10}$$

Note that l_{sh} is sub-polynomial in λ so the loss factor is not exponential.
Game 2. We add another abort rule. The challenger guesses 2 files j, k. If $\{F_j, F_k\} \neq \{F_0, F_1\}$, abort the game. Thus

$$\mathbf{Adv}_{\mathcal{S}}^{Game\ 1}(\lambda) \leq (|\mathbf{F}|)^2\, \mathbf{Adv}_{\mathcal{S}}^{Game\ 2}(\lambda) \tag{11}$$

Now we show how to construct $\mathcal{A}$ against Π_{Enc} from $\mathcal{S}$. $\mathcal{A}$ can guess $\{F_0, F_1\}$ since ***Game 2*** does not abort. At the setup phase, $\mathcal{A}$ includes the public parameters of Π_{Enc} in $\Pi_{\mathsf{PAKE,re\text{-}enc}}$ public parameters, queries for her own challenge ciphertext C_b with $\{m_0 = F_0, m_1 = F_1\}$ in her IND-KPA game. $\mathcal{A}$ fixes this C_b as the ciphertext of F_0 and use other random keys (conformant to security parameter) to encrypt all other files as described in the model and this protocol.

The Send and $\mathsf{RegisterCorrupt}$ queries can be answered as in the proof for security against compromised client. For $\mathsf{AccessDB}$, $\mathcal{A}$ simply gives $\mathcal{S}$ all the ciphertexts and owner lists at that time. Whenever a query from $\mathcal{S}$ results in an observable database change (i.e., new ciphertexts are added or new owners are added to files), $\mathcal{A}$ updates the server state and gives the ciphertext and/or the changed owner lists to $\mathcal{S}$. For $\mathsf{Execute}(\mathsf{pid}, \mathsf{P}, \mathsf{F})$ with $F \notin \{F_0, F_1\}$, $\mathcal{A}$ can use the homomorphic property of Π_{Enc} to correctly generate all the transcript. Since $\mathcal{A}$ knows all other keys and ciphertexts, $\mathcal{A}$ can answer all the queries from $\mathcal{S}$.

If $\mathcal{S}$ queries $\mathsf{Test}()$, $\mathcal{A}$ replies with C_b of her own and outputs whatever $\mathcal{S}$ outputs. Since the probability that $\mathcal{A}$ correctly simulates the SDoE-game for $\mathcal{S}$ is exactly $\frac{1}{2}$, we have

$$\mathbf{Adv}_{\mathcal{S}}^{Game\ 2}(\lambda) \leq 2\epsilon_{\mathcal{A}} \tag{12}$$

By combining (8) to (12), we have proved Lemma 2.

By combining (6), (7) and the two lemmas, we have proved Theorem 2.

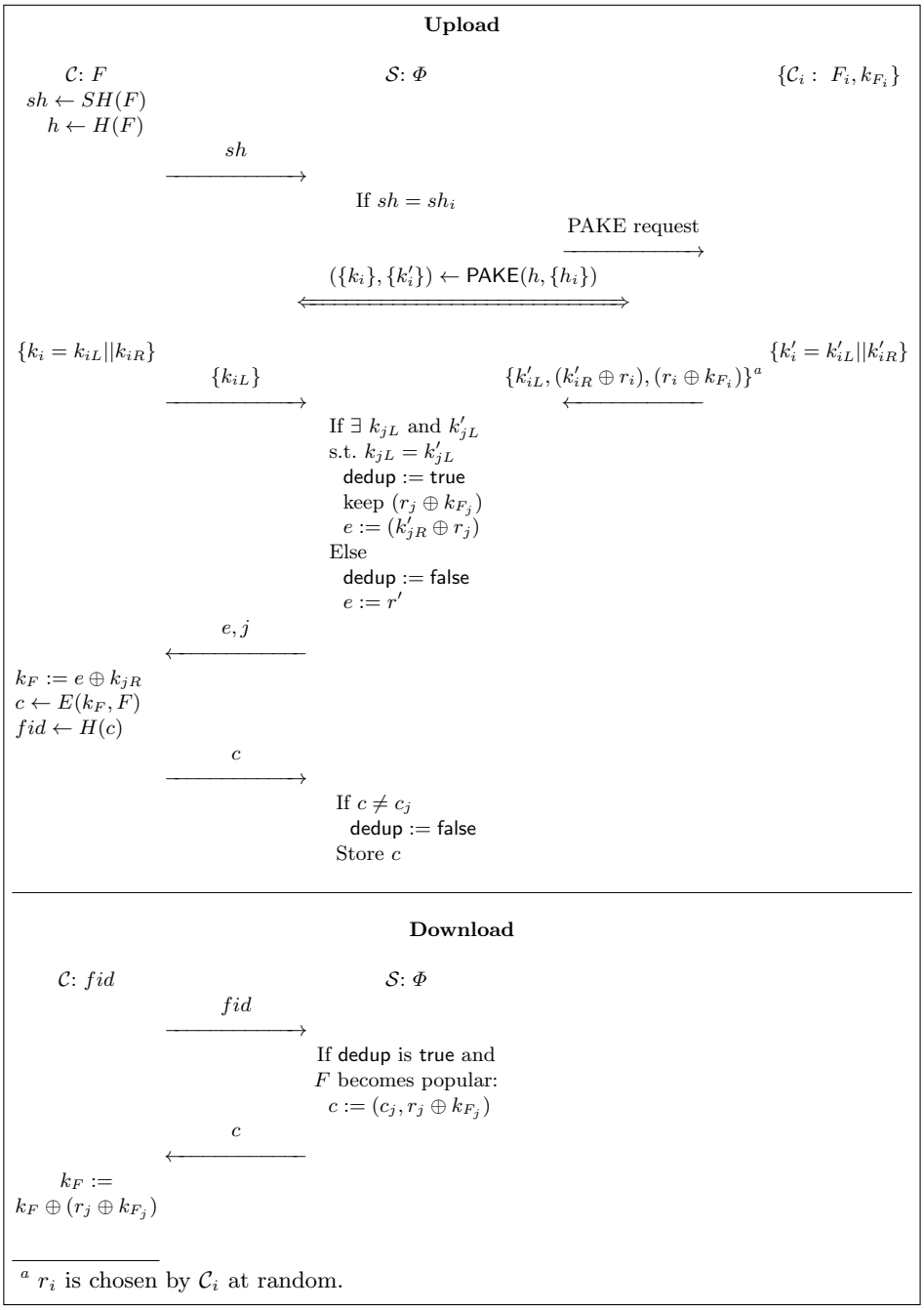


Fig. 3. PAKE-based deduplication on popular files.

4.2 PAKE-based deduplication on popular files

Our second scheme ($\Pi_{\mathsf{PAKE,popular}}$) is shown in Figure 3. It tries to avoid using public-key operations to encrypt the entire file. The penalty is that it only deduplicates popular files. The idea is the same as $\Pi_{\mathsf{PAKE,re\text{-}enc}}$, except that instead of deleting the duplicated files directly, $\mathcal{S}$ keeps them until they become popular.

Note that, for unpopular files, the views of both $\mathcal{S}$ and $\mathcal{C}$ are similar to those in $\Pi_{\mathsf{PAKE,re\text{-}enc}}$, except that XOR is used to replace addition and subtraction and a symmetric-key encryption scheme $E()$ is used to replace $F \cdot g^{k_F}$. So the security argument for $\Pi_{\mathsf{PAKE,re\text{-}enc}}$ still holds for unpopular files here.

Deduplication effectiveness will be negatively affected if only popular files are deduplicated. In the next section, we show that this affection is small via simulations with realistic datasets.

5 Simulation

The authors in [13] did a realistic simulation to measure the deduplication effectiveness of Π_{PAKE}. They used a dataset comprising of Android application popularity data to represent the predominance of media files. We follow their simulation but with two improvements.

First, we expanded the data set to a more reasonable size since their dataset is relatively small (7 396 235 "upload requests" in total, of which 178 396 are for distinct files). In order to measure how the system behaves as the number of unique files increases, a larger dataset is needed. Since such data was not available, we used the Synthetic Minority Over-sampling (SMOTE) Technique [8] to generate extra samples. Given a set of input samples and the amount of required over-samplings, SMOTE performs the following for each input sample:

1. Compute x nearest neighbors for the input sample.
2. Randomly choose a neighbor and a point on the line segment joining the input sample to the selected neighbor. This point is a new, generated sample.
3. Repeat step 2 until the requested amount of over-sampling has been reached. For example, if the amount of needed over-sampling is 200%, it will be repeated twice.

We used the (file size, popularity) pairs of the original dataset as the input samples in the SMOTE algorithm. The amount of over-sampling was 500% and for each input sample five nearest neighbors were considered when the new samples were computed. The hashes for the synthetic samples were chosen randomly. These new samples were combined with the samples from the original dataset into a *expanded* dataset. The expanded dataset contains 110 942 571 files of which 2 675 917 are unique. See Figure 4 for the file popularities of the original dataset and the expanded dataset.

Second, we adjust the distribution of upload request to better reflect the real world cases. In [13], they map the dataset to a stream of upload requests by generating the request in random order, i.e., a file that has x copies generates x upload requests that are uniformly distributed during the simulation. We argue

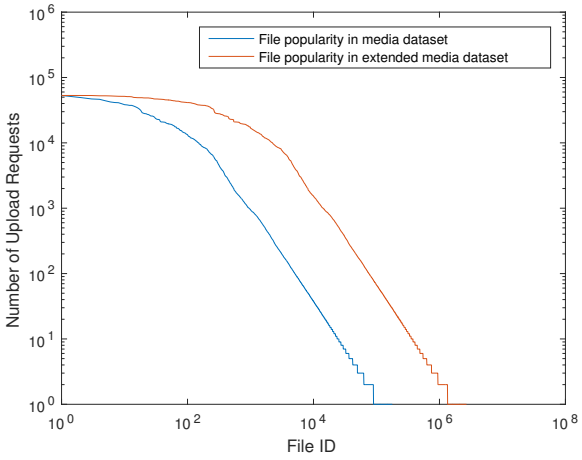


Fig. 4. File popularity.

that this cannot precisely capture the upload stream in real world: a file usually has less upload requests when it was generated, and becomes increasingly popular (more and more people hold it). To capture this case, we assume the upload requests of a single file follows normal distribution $\mathcal{N}(\mu, \sigma^2)$ where μ and σ are chosen randomly. Specifically, for a file F_i that has x_i total copies, the number of copies of F_i uploaded at time point t is $y_i = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(t-\mu_i)^2}{2\sigma_i^2}} x_i$. Then the total number of files uploaded at time point t is $\sum y_i$ and we assume that they are uploaded in random order. We do this for all time points and measure the final deduplication percentage.

Parameters. We follow [13], setting the number of possible files as 825 000, $l_{sh} = 13$ and $(n_{\mathsf{RL}} + n'_{\mathsf{RL}}) = 100$ (i.e., a $\mathcal{C}$ will run PAKE at most 100 times for a certain file as both uploader and checker). We use these parameters in our simulations and measure deduplication effectiveness using the *dedpulication percentage* ρ:

$$\rho = (1 - \frac{Number\ of\ all\ files\ in\ storage}{Total\ number\ of\ upload\ requests}) \cdot 100\% \tag{13}$$

Rate limiting. We first assume that all $\mathcal{C}$s are online during the simulation and all files will be deduplicated (not limited to popular files). We run simulations with different combinations of RL_u and RL_c that satisfies $RL_u + RL_c = 100$, to see how selecting specific values for rate limits affects the deduplication effectiveness. Figure 5 shows that setting $RL_u = RL_c = 50$ maximises ρ to be 94.85%, which is close to the perfect deduplication percentage of 97.59%.

Offline rate. Note that $\mathcal{C}$s cannot participate in the deduplication protocol if they are offline, which may negatively affect deduplication effectiveness. To estimate this impact, we assign an *offline rate* to each $\mathcal{C}$ as its probability to be

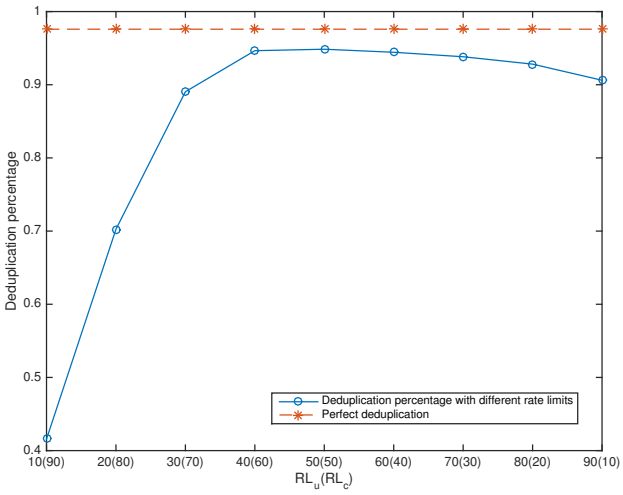


Fig. 5. Dedup. percentage VS. rate limits.

offline during one run of the deduplication protocol. We set rate limits $RL_u = 50$ and $RL_c = 50$, and measured ρ by varying the offline rate. Figure 7 shows that ρ is still reasonably high even for for relatively high offline rates of up to 70%, but drops quickly beyond that.

Popularity threshold. By far the simulation results is for $\Pi_{\mathsf{PAKE,re\text{-}enc}}$. Recall that $\Pi_{\mathsf{PAKE,popular}}$ only deduplicates popular files which have a number of copies that are larger than a threshold, called *popularity threshold*. To investigate how this strategy affects deduplication effectiveness, we set rate limits $RL_u = 50(RL_c = 50)$, offline rate as 0.5, and run the simulation with different popularity thresholds. Figure 7 shows that ρ drops quickly if the popularity thresholds is larger than 32.

6 Related Work

The first proposed SDoE scheme is *convergent encryption* (CE) [10], which uses $H(F)$ as a key to encrypt F, where $H()$ is a publicly known cryptographic hash function. In this way, different copies of F result in the same ciphertext. However, a compromised passive $\mathcal{S}$ can easily perform an offline brute-force attack over a predictable file. Bellare et al. recently formalized CE and proposed *message-locked encryption* (MLE) and its interactive version (iMLE) [2], which uses a semantically secure encryption scheme but produces a deterministic tag [4]. So it still suffers from the same attack. More recent work has attempted to improve MLE in several respects. Qin et al. [16] and Lei et al. [12] made MLE support *Rekeying* to protect key compromise and enable dynamic access control

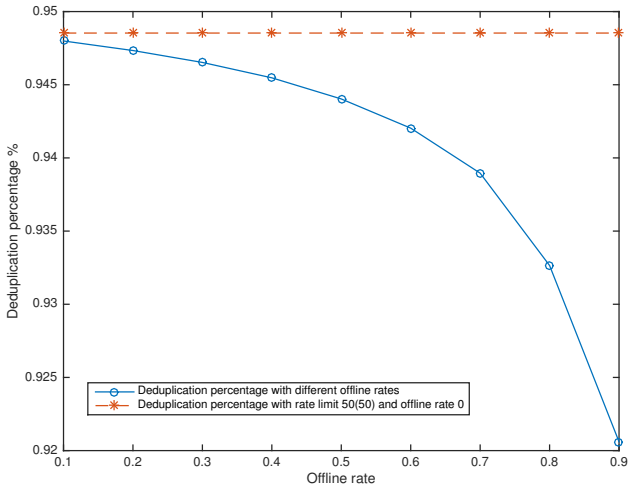


Fig. 6. Dedup. percentage VS. offline rates.

in the cloud storage. Zhao and Chow [20] proposed updatable MLE so that an encrypted file F can be efficiently updated with $O(log|F|)$ computational cost. However, none of these improvements is able to make MLE secure against offline brute-force attack.

DupLESS is a SDoE scheme that improves the security of CE against offline brute-force attacks [3]. In the key generation phase of CE, they introduce another secret which is provided by an third party and identical for all $\mathcal{C}$s. It adopts oblivious PRF to make sure that $\mathcal{C}$s generate their keys without revealing their files, or letting the $\mathcal{C}$s learn anything about the secret. Duan [11] and Shin [17] later used decentralized architectures to distributed the trust of the third party in DupLESS.

Cloudedup is a SDoE scheme that introduces a third party for encryption and decryption [15]. Stanek et al. propose another SDoE scheme that only deduplicates popular files [19]. Alternatively, Dang and Chang [9] proposed a trusted hardware based SDoE that encrypts files using a randomized encryption scheme under the seal key inside enclave, and uses a privacy-preserving comparison scheme to remove duplicates in the storage. They further use differential privacy to guarantee that the aggregate information is not exposed by the traffic analysis. However, all the above approaches are vulnerable to the online brute-force attacks.

7 Conclusions

In this paper, we revisited the problem of secure deduplication of encrypted data (SDoE). We proposed a formal security model for this problem. We also proposed

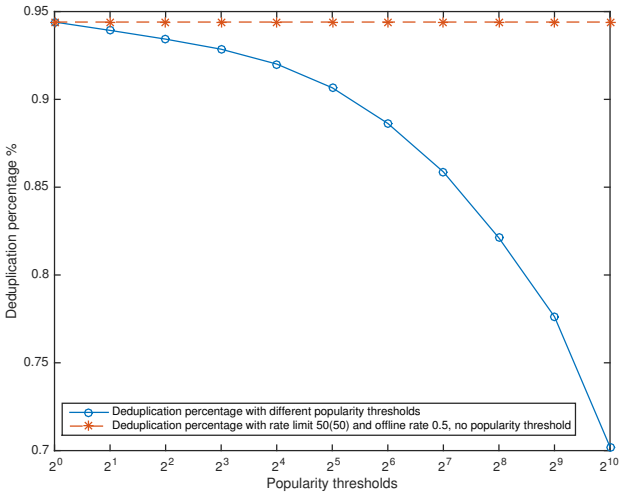


Fig. 7. Dedup. percentage VS. popularity thresholds.

two single-server SDoE protocols and proved their security in our model. We showed that both of them can achieve reasonable deduplication effectiveness via simulations with realistic datasets.

References

1. G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, Feb. 2006.
2. M. Bellare and S. Keelveedhi. Interactive message-locked encryption and secure deduplication. In *IACR International Workshop on Public Key Cryptography*, pages 516–538. Springer, 2015.
3. M. Bellare, S. Keelveedhi, and T. Ristenpart. DupLESS: Server-aided encryption for deduplicated storage. In *USENIX Security*, pages 179–194. USENIX Association, 2013.
4. M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. In *EUROCRYPT*, volume 7881 of *LNCS*, pages 296–312. Springer, 2013.
5. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings*, pages 139–155, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
6. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, pages 139–155, 2000.

7. S. M. Bellovin and M. Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 72–84, May 1992.
8. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.
9. H. Dang and E. C. Chang. Privacy-preserving data deduplication on trusted processors. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 66–73, June 2017.
10. J. Douceur, A. Adya, W. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624, 2002.
11. Y. Duan. Distributed key generation for encrypted deduplication: Achieving the strongest privacy. In *CCSW*, pages 57–68. ACM, 2014.
12. L. Lei, Q. Cai, B. Chen, and J. Lin. *Towards Efficient Re-encryption for Secure Client-Side Deduplication in Public Clouds*, pages 71–84. Springer International Publishing, Cham, 2016.
13. J. Liu, N. Asokan, and B. Pinkas. Secure deduplication of encrypted data without additional independent servers. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 874–885, New York, NY, USA, 2015. ACM.
14. J. Liu, N.Asokan, and B. Pinkas. Secure deduplication of encrypted data without additional independent servers. Cryptology ePrint Archive, Report 2015/455, 2015. `http://eprint.iacr.org/2015/455`.
15. P. Puzio, R. Molva, M. Onen, and S. Loureiro. Cloudedup: Secure deduplication with encrypted data for cloud storage. In *CloudCom*, pages 363–370. IEEE Computer Society, 2013.
16. C. Qin, J. Li, and P. P. C. Lee. The design and implementation of a rekeying-aware encrypted deduplication storage system. *Trans. Storage*, 13(1):9:1–9:30, Feb. 2017.
17. Y. Shin, D. Koo, J. Yun, and J. Hur. Decentralized server-aided encryption for secure deduplication in cloud storage. *IEEE Transactions on Services Computing*, PP(99):1–1, 2017.
18. V. Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.
19. J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl. A secure data deduplication scheme for cloud storage. In *FC*, pages 99–118, 2014.
20. Y. Zhao and S. S. Chow. Updatable block-level message-locked encryption. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 449–460, New York, NY, USA, 2017. ACM.