Towards Thwarting Template Side-Channel Attacks in Secure Cloud Deduplications

Yuan Zhang, Yunlong Mao, Minze Xu, Fengyuan Xu, *Member, IEEE*, and Sheng Zhong

Abstract—As one of a few critical technologies to cloud storage service, deduplication allows cloud servers to save storage space by deleting redundant file copies. However, it often leaks side channel information regarding whether an uploading file gets deduplicated or not. Exploiting this information, adversaries can easily launch a template side-channel attack and severely harm cloud users' privacy. To thwart this kind of attack, we resort to the k-anonymity privacy concept to design secure threshold deduplication protocols. Specifically, we have devised a novel cryptographic primitive called "dispersed convergent encryption" (DCE) scheme, and proposed two different constructions of it. With these DCE schemes, we successfully construct secure threshold deduplication protocols that do not rely on any trusted third party. Our protocols not only support confidentiality protections and ownership verifications, but also enjoy formal security guarantee against template side-channel attacks even when the cloud server could be a "covert adversary" who may violate the predefined threshold and perform deduplication covertly. Experimental evaluations show our protocols enjoy very good performance in practice.

Index Terms—Cloud, secure deduplication, privacy, proofs of ownership

1 Introduction

Cloud deduplication enables the cloud server to remove redundant copies of the same file which is uploaded by different cloud users and save its storage resource. Via checking equality of files, deduplications can be straightforwardly implemented when all files are uploaded and stored in plain text. Unfortunately in this case, cloud users face a high risk of security threats since their data is fully exposed to both inside adversaries of the cloud (e.g., dishonest staff) and outside attackers (e.g., hackers) who have successfully breached the cloud. To deal with this issue, users are recommended to encrypt their files before uploading them to the cloud.

However, deduplications over encrypted files are not easy for cloud server. When different users individually encrypt the same file with their own keys, the encryptions are totally different and the cloud server cannot perform deduplications with easy equality examinations. Moreover, even if the server knows two encryptions correspond to the same file, it cannot simply delete one of them since file owners only know their own keys and how to decrypt their own encrypted files. In recent years, a number of smart protocols (e.g., [1], [10], [12], [13], [15], [17], [22], [23], [27]) have been designed by researchers to make deduplications over encrypted files possible. For example, the protocol proposed in [17] lets cloud users encrypt their files with the *convergent encryption* (CE). Since the CE uses a file's hash value as the key to encrypt it, encryptions of the same file always are the same even they are generated by different cloud users individually. In [15], previous uploaders of a file are invited to run a secure communication protocol with current uploader of the same file so that all of them eventually share the same key as the one chosen by the first uploader.

While deduplications are enabled in above protocols and encrypted files greatly enhance cloud user's security level, deduplications could become a side-channel and adversaries can utilize it to launch *template attacks*. An adversary can fill the personal information of its attacking target into a privacy-related template file (e.g., medical records, financial documents, etc.), and upload it to the cloud to see whether deduplication happens.[1] If deduplication does happen, the adversary knows the file it crafted does exist, and the private information of the target is revealed or verified. For instance, an adversary fills Alice into a prescription for some specific disease or a registration form of a clinic specialized in that disease, and finds this file triggers deduplication. It immediately knows Alice is probably suffering from that disease.

So far, the above side-channel template attack seems to be inevitable unless we give up the benefits of deduplications and inject bogus network traffics or server operations

• *The authors are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, P.R. China, and also with the Computer Science and Technology Department, Nanjing University, Nanjing 210023, P.R. China. E-mail: {zhangyuan, zhongsheng}@nju.edu.cn, njucsmyl@163.com, {cnmzxu, fengyuan.x}@gmail.com.*

Manuscript received 25 June 2018; revised 27 Dec. 2018; accepted 12 Feb. 2019. Date of publication 15 Apr. 2019; date of current version 13 May 2021. (Corresponding author: Sheng Zhong.)
Digital Object Identifier no. 10.1109/TDSC.2019.2911502

1. Adversaries can acquire this side-channel information via examining the network traffic or the server's responding behaviors. For instance, to save the network resource of the cloud server, many cloud server requires users to run a deduplication test by uploading a short hash of the file and comparing it with the hash values of existing files. In case there is a match, no more uploading needs to be done by users and the cloud server will add the link of matched file to their accounts. Otherwise, users will continue to upload the entire encrypted file. The difference between users' uploading operations in the two cases can be easily noticed by an eavesdropping adversary.

to remove the side channel. Due to this reason, we aim to design pragmatically defense solutions to thwart the template side-channel attack and mitigate its damage to cloud users' privacy. To provide rigorous theoretical privacy guarantees in our solution, we resort to the widely used privacy concept "*k-anonymity*" which is first formulated by Sweeney [25]. Basically, K-anonymity ($K \in \mathbb{N}$) allows to reveal a piece of data only if there are at least K different users sharing the same data, and greater K provides better privacy protections. Based on K-anonymity, we construct our deduplication protocols which allow the cloud server to perform deduplications on a file only if at least K different users have uploaded the same file. Therefore, even when the adversary successfully launches a template side-channel attack and knows its testing file has been uploaded to the cloud system, it can hardly link the file to a specific user.[2] Meanwhile, when the file contains sensitive data that identifies a user uniquely, no deduplication will be performed. This looks like a limitation of our solution, but actually is how deduplications are supposed to work. Deduplications are supposed to be performed only when multiple users have a same file stored, and benefit cloud server the most when the total number of copies is large. Accordingly, our k-anonymity-based-solution is a perfect choice given 1) it admits deduplications only if k ($k > 1$) copies exist; 2) the greater k is, the stronger privacy is provided. Also, we note that different users could possess the same document even when the document is highly private. Nowadays, many private documents are generated with only partial identification information for security considerations. For example, your bank e-statement or medical report generally prints only the last 4 digits of your account-number/phone-number. This could cause different persons share the same private document.

Besides the threshold deduplication mentioned above, our solution also supports data confidentiality and ownership verification. Essentially, we have devised a novel cryptographic primitive called "*dispersed convergent encryption*" (DCE) scheme. The DCE scheme can be viewed as an integration of three techniques, namely the convergent encryption, information dispersal algorithm and secure verifiable computation. The convergent encryption is used to provide the confidentiality protection to users' files. The information dispersal and the secure verifiable computation together offer the functionalities of threshold deduplication and ownership verification. By employing the DCE scheme, we propose a secure deduplication protocol which is able to defense against a dishonest cloud server in the *covert adversary* model [9], which is stronger compared with the curious and passive adversary in the semi-honest model). Our protocol guarantees that the threshold deduplication will be performed correctly even if the cloud server may covertly perform deduplications before the threshold is reached.

2. Although sometimes the adversary may control some users in the system, the number is still limited due to Sybil defensing mechanisms. We can therefore increase the value of K and still provide reasonable privacy protections to honest users. If this is not the case, there is little we can do to thwart attackers' template side-channel attack unless deduplication is forbidden.

TABLE 1
Major Notations

Notations	Meaning
K	the predefined threshold for deduplication
λ	security parameter
$\mathcal{CE}$	a convergent encryption scheme
m	input message
k	a secret key for encryption and decryption
uid, i	user's unique index
$C\langle i\rangle$	user i's DCE ciphertext
C'	secondary ciphertext
aux	auxiliary information
c	CE ciphertext
fid	unique index of a file F
$\mathcal{E}$	a regular semantically secure symmetric cipher
$\vec{C}, \vec{c}$	vector of C, vector of c, respectively
$h()$	a cryptographic hash function

Our major contributions can be summarized as follows.

- We propose a practical secure deduplication protocol with theoretical privacy guarantees that can deal with template side-channel attacks in the covert adversary model, and protect data confidentiality and ownership verification.
- We devise a novel cryptographic primitive called dispersed convergent encryption scheme that can be used to construct secure deduplication efficiently without relying on trusted third parties or independent servers.
- We propose two efficient constructions of DCE schemes $\mathcal{R}-\mathcal{DCE}1$ and $\mathcal{R}-\mathcal{DCE}2$. The secure deduplication protocol $\mathcal{R}-\mathcal{DCE}1$-SDP constructed based on the first scheme is 100 percent correct, highly efficient and protects cloud users' privacy against adversaries who know up to $K-1$ valid *proof-of-ownership* (PoW) evidences. The secure deduplication protocol $\mathcal{R}-\mathcal{DCE}2$-SDP constructed based on the second scheme is a randomized solution which is correct with an overwhelming probability, and can protect users' privacy even against adversaries with no restriction on the number of known valid PoW evidences.
- We formally prove the correctness and security properties of our schemes and protocols.
- We perform experiments to evaluate our schemes, and experimental results show our constructions achieve very good performance.

2 Preliminaries

In this section, we introduce our system models, define major notations or concepts, and give a short review of tools that we use to construct our own protocol. To ease reading, we summarize major notations in Table 1.

2.1 System Models

In this paper, we consider a cloud server who performs threshold client-side deduplications over encrypted files with its cloud users to reduce its storage cost. We assume there is a publicly agreed, predefined threshold $K \in \mathbb{N}$.

For each file, before the total number of its valid uploads reaches K, server demands the encrypted file as well as a proof-of-ownership evidence when a user requests to upload it. Server keeps them in its storage and links them to this user's account. When the threshold is reached, server performs deduplications by removing redundant copies of this file from all users' accounts, and changes to demand their PoW evidences only from later uploaders. During above process, server maintains a counter for every file which records the total number of valid uploads regarding this file, and increases the counter's value by one when a valid upload is received. Here by "valid" we mean the PoW evidence is accepted by server, and from a user who has never uploaded this file before.

We assume each user has a unique index. In addition, we assume each user communicates with the server via a secure channel, which means messages between server and users are authenticated and encrypted.

2.2 Threat Models

Malicious Outside Adversary. The outside adversary may obtain some knowledge about a targeting file (e.g., some PoW evidences) and enroll to interact with the server to invade other users' privacy. Specifically, we focus on two different kinds of adversaries. The first one is the adversary who aims to launch template side-channel attacks. As we discussed in Introduction, this kind of adversary can fill some personal and private information about the attacking target in a template file, and then upload this file to the cloud to detect whether deduplication happens. This knowledge may reveal some privacy of the victim if the template file is well chosen. The other one is the adversary who aims to use its knowledge about some file's PoW evidences to cheat server to pass the PoW verification. For example, most cloud servers may simply take the hash value of a file as the evidence of ownership and deduplication. An adversary can defraud this kind of servers of some specific file by showing its corresponding hash value. Sometimes, such kind of short evidence can be easily obtained and that makes the cloud storage service in danger. Note that the "PoW attacker" is an important type of attacker and has been considered in most existing secure deduplication work (e.g., [4], [12]). When a PoW attacker's attack succeeds, the cloud falsely recognizes the attacker as the targeting file's owner, and grants it access authority. This immediate causes complete privacy leakage to the real owner.

Covert Inside Adversary. Since deduplications can greatly reduce the storage cost, cloud servers may have incentives to covertly perform deduplications even when the threshold is not reached. With the existence of this kind of server, the threshold of deduplication may degenerate to one. Unfortunately, if this behavior cannot be realized by the users efficiently, a 'clever' server will prefer to act in this way. This will immediately weaken the privacy guarantee of the cloud storage service. Therefore we model this kind of servers as a *covert adversary* [9]. A covert adversary only follow the protocol without any deviation (same as a semi-honest adversary) if its cheating behavior can be found out efficiently by others. Thus this model provides stronger security guarantees than the semi-honest model.

In the sequel, we referred to above three kinds of adversaries as Type-1, Type-2 and Type-3 adversaries respectively.

2.3 Privacy Model and Leakage Model

K-anonymity. We adopt the *K-anonymity* privacy model which requires that server only performs deduplications on a file when there are at least K different users who have uploaded this file.

Bounded retrieval model (BRM). BRM is a widely-used leakage model in leakage-resilient cryptography. In BRM, information leaked to the adversary is assumed to have an upper bound. We adopt BRM to analyze the security of our secure deduplication protocols against Type-2 adversaries.

2.4 Convergent Encryption

Convergent encryption [5] is a symmetric encrypting framework in which the secret key is deterministically computed from the message to be encrypted. Since it provides reasonable data's confidentiality and guarantees same plaintexts' ciphertexts are the same, CE schemes and their variations have been widely used in existing secure deduplication protocols (e.g., [12], [23], [26], [27], [29]).

A CE scheme $\mathcal{CE}$ generally consists of:

- $k := \mathcal{CE}.Gen(m, 1^\lambda)$: a deterministic algorithm that takes as input a message $m \in \{0,1\}^*$ and a security parameter $\lambda \in \mathbb{N}$, outputs a secret key $k \in \{0,1\}^l$;
- $c := \mathcal{CE}.Enc_k(m)$: a deterministic encryption algorithm that takes as input m (and implicitly a secret key k determined by m), outputs a ciphertext c of m;
- $m := \mathcal{CE}.Dec(c, k)$: a deterministic decryption algorithm that takes as input a ciphertext c and a secret key k, returns c's plaintext m as the output.

In our paper, we use the CE scheme as an underlying building block of our DCE schemes.

2.5 Rabin's Information Dispersal Algorithm

Rabin's information disperse algorithm (IDA) [18] is a well-known algorithm which is widely used in reliable data storage or transmission in distributed systems.

Basically, IDA slides a long message into K short messages, and generates N ($N \geq K$) new message pieces by computing N different linear combinations of these short messages. The coefficients used in all combinations are specifically chosen to make sure these linear combinations are independent (or independent with probability close to 1). Accordingly, when more than K new message pieces are collected, all K short messages can be computed by solving K linear equations. Rabin's IDA is used to construct our DCE schemes.

3 DCE-Based Secure Deduplication Protocols

In this section, we show how to construct a secure deduplication protocol (SDP) based on a DCE scheme.

3.1 The Dispersal Convergent Encryption

To construct a SDP, we assume a dispersal convergent encryption scheme consisting of the following algorithms is available:

- $(Paras, \mathcal{CE}) \leftarrow \mathcal{DCE}.Setup(\lambda)$. Given a security parameter, the *setup algorithm* chooses a secure CE scheme $\mathcal{CE}$ which will be used to construct $\mathcal{DCE}.Enc()$, and generates $Paras$ which consists of supporting parameters for other algorithms.
- $k := \mathcal{DCE}.Gen(m, 1^\lambda)$. Given as input a message m, a security parameter λ, the *key generation algorithm* outputs a secret key k for encryption and decryption.
- $(C, aux) := \mathcal{DCE}.Enc(m, uid, K)$. Given as input a message m, a user's unique index uid, and the threshold K, the *encryption algorithm* outputs a ciphertext C and auxiliary information aux that can be used by $\mathcal{DCE}.Vrfy()$ and $\mathcal{DCE}.Recr()$.
- $y := \mathcal{DCE}.Vrfy(C, uid, aux)$. Given as input a ciphertext C, a user's unique index uid and auxiliary information aux that corresponds to uid and m, the *verification algorithm* determines whether C equals the ciphertext corresponds to uid and m.
- $c := \mathcal{DCE}.Recr(\{C\}, \{uid\}, \{aux\})$. Given as input a set of K different ciphertexts $\{C\}$ of a same message m, their corresponding users' indexes $\{uid\}$ and auxiliary information $\{aux\}$, the *recovery algorithm* outputs a CE ciphertext c of m.
- $m := \mathcal{DCE}.Dec(c, k)$. Given as input a CE ciphertext c and its secret key k, the *decryption algorithm* returns c's plaintext m.

Specifically, DCE scheme's recovery algorithm allows server to recover a CE ciphertext of a message after K DCE ciphertexts are collected. This property can be utilized to implement a threshold deduplication protocol that thwarts template side-channel attacks. In addition, DCE scheme's verification algorithm can be utilized to implement the PoW part of our SDP.

In Sections 4 and 5, we present two different constructions of the DCE scheme to handle two different adversary settings.

3.2 DCE-Based Secure Deduplication Protocols

Here we demonstrate how to construct a secure deduplication protocol with the DCE scheme.

Notice DCE scheme's recovery algorithm allows one to recover a CE ciphertext of a file from K different DCE ciphertexts of the same data. In addition, a DCE ciphertext's correctness can be verified by the verification algorithm. Therefore, to construct our SDP, we let the user upload its DCE ciphertext to sever. After K DCE ciphertexts are collected, server can apply the recovery algorithm to recover the CE ciphertext and then perform deduplications. Furthermore, by carefully design the DCE scheme to make sure a user cannot generate a valid DCE ciphertext for itself unless it knows the CE ciphertext,[3] the DCE ciphertext and the verification algorithm can be used as the PoW evidence and verification algorithm respectively in our SDP.

Following the idea above, we showcase our DCE-based secure deduplication protocol (DCE-SDP)'s workflow as follows. The security analyses of DCE-SDP are postponed till we give detailed constructions of underlying DCE schemes in the next two sections.

3.2.1 User-Side Uploading Operations

When a user i wants to upload a file F, it first *sends F's unique index*[4] *fid as its uploading request* to the server. Depending on the total number of valid uploads regarding this file, server may request the user to *either perform a complete upload or a reduced upload*. The former requires to upload a DCE ciphertext together with its corresponding auxiliary information (which can be used as both the PoW evidence), and a *secondary ciphertext* of F (which allows the user to retrieve its file before server performs deduplications on this file). While the latter only requires to upload the DCE ciphertext.

Specifically, to perform a complete upload, user i computes the secret key k, the ciphertext of F and the auxiliary information aux

$$k := \mathcal{DCE}.Gen(m, 1^\lambda); \tag{1}$$

$$(C\langle i \rangle, aux) := \mathcal{DCE}.Enc(F, i, K), \tag{2}$$

and a *secondary ciphertext* by encrypting F with a regular semantically secure symmetric cipher $\mathcal{E}$ (e.g., AES-CTR or AES-CBC)

$$C'\langle i \rangle := \mathcal{E}.Enc(F, k_i), \tag{3}$$

and uploads $(C\langle i \rangle, C'\langle i \rangle, aux)$ to the server. Here k_i is a random secret key that is generated by user i privately. For later decryption, user i stores k, k_i in its local storage together with F's index.

To perform a reduced upload, the user simply generates F's ciphertext $C\langle i \rangle$, and uploads it to the server.

3.2.2 Server-Side Responding Operations to Uploads

Server maintains a private counter array $Ctr[]$, and an auxiliary information array $Aux[]$ to record total number of valid uploads, and auxiliary verification information for all files respectively. In addition, server maintains a ciphertext array $Cpr[]$ to store files' ciphertexts.

When server receives an uploading request regarding file fid from user i, it *checks whether the total number of valid uploads $Ctr[fid]$ has exceeded the threshold K*. If not, server requires the user to perform a complete upload; otherwise, server notifies the user to perform a reduced upload.

Upon receiving a complete upload $(C\langle i \rangle, C'\langle i \rangle, aux)$, server adds the auxiliary information into $Aux[fid]$ and *verifies the correctness of $C\langle i \rangle$* by running the verification algorithm

$$y := \mathcal{DCE}.Vrfy(C\langle i \rangle, i, aux). \tag{4}$$

• If the verification algorithm accepts, server stores the two ciphertexts and their sender's index by adding them to the ciphertext array

$$Cpr[fid] \leftarrow (i, C\langle i \rangle, C'\langle i \rangle), \tag{5}$$

3. Here we assume if an adversary knows the CE ciphertext of a file, it must have the file. This assumption is reasonable since the CE ciphertext's length is no less than the file's length, thus acquiring the CE ciphertext of a file should be no easier than acquiring the file itself from some channel.

4. For example, the index can be computed by applying a publicly-agreed collision resistant hash function on F.

updates the counter array[5]

$$Ctr[fid] := Ctr[fid] + 1, \tag{6}$$

and adds the secondary ciphertext's link/address to user i's account. In addition, *if the counter reaches the threshold now, server recovers file fid's CE ciphertext*

$$c := \mathcal{DCE}.Recr(\{(j, C\langle j\rangle, aux)\}). \tag{7}$$

Once server has this ciphertext, *it performs the deduplication* by removing all stored ciphertexts of file fid in $Cpr[fid]$, storing C'' to it, replacing file fid's secondary ciphertext's link with C'''s link to every existing file owner account.

• Otherwise the algorithm rejects, server does nothing.

Upon receiving a reduced upload $(C\langle i\rangle)$, server verifies its correctness as

$$y := \mathcal{DCE}.Vrfy(C\langle i\rangle, i, Aux[fid]). \tag{8}$$

Note that here the auxiliary information is retrieved from server's local storage, and the reduced upload only happens after the deduplication.

• If the verification algorithm accepts, server discards $C\langle i\rangle$ links file's convergent encryption's ciphertext to user's account.

• Otherwise the algorithm rejects, server does nothing.

3.2.3 User-Side Downloading Operations

If a user i requests a file fid, server checks if the user's account has any link of file fid's ciphertext. If yes, server return a copy of the ciphertext to the user; otherwise, server rejects this user's request.

Note that the ciphertext received by the user could be the secondary ciphertext C', which can be decrypted using file fid's corresponding key k_i, or the recovered convergent encryption ciphertext c, which can be decrypted using k . Once the user receives the latter, it can remove k_i from its local storage.

Remark: Notice that our DCE-SDR protocol relies on auxiliary information that is uploaded by users to verify the correctness of their ciphertexts that they upload. Although a malicious user can launch a *pollution attack* by uploading a false auxiliary information and using it to pass the verification, this would not cause a big problem in our protocol. Since for all users, the auxiliary information for the same file should always be the same. Therefore, whenever server finds two pieces of different auxiliary information regarding to a same file, it can wait for more uploads to make its judgement or can ask the two users to provide more evidences to justify their uploads.

Another major problem in the application of our DEC-SDR protocol is how to choose the value of K. It is impossible to find a fixed, proper value of K for all situations. We regard this as a tradeoff of the privacy and the efficiency of a cloud storage system. The greater K is, the stronger privacy guarantees we will have. But with a too large K, the cloud server needs to save many copies of a same file, and this will waste much storage space. So in most cases, we think the K is big enough if we know an adversary can hardly control more than K corrupted users. For example, if the server strictly controls the number of user accounts registered by any person and if users do not collude, then a relatively small K like tens is enough in general. But in other cases, we may have to set K to thousands or even millions to obtain sufficient privacy guarantees. Above all, it is vital to have a good measurement or estimation about the demographics especially the malicious or corrupted users in the entire system before setting a proper K.

5. Here we neglect the case where a user performs more than one correct uploads. If such a case happens, server can easily recognize it (by checking the ciphertext array), and does not make any change to the system.

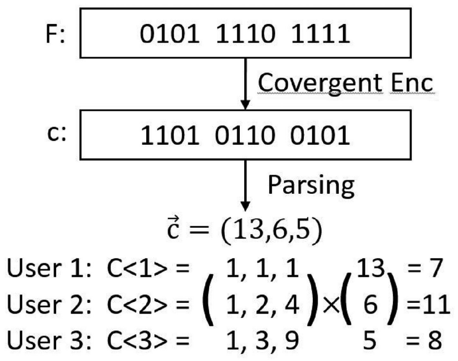

F:
0101 1110 1111
Covergent Enc
c:
1101 0110 0101
Parsing
c⃗ = (13,6,5)
User 1: C<1> = 1, 1, 1 × 13 = 7
User 2: C<2> = 1, 2, 4 6 =11
User 3: C<3> = 1, 3, 9 5 = 8

Fig. 1. An illustrative example of $\mathcal{R}{-}\mathcal{DCE}1.Enc$ with $K = 3, l = 5, p = 17$. All matrix computations are performed over $\mathbb{Z}_{17}$.

4 A DCE Construction Based on Rabin's IDA

4.1 Construction of $\mathcal{R}{-}\mathcal{DCE}1$

In this section, we present $\mathcal{R}{-}\mathcal{DCE}1$, our first DCE scheme's construction based on Rabin's IDA. The main idea of this construction is as follows. To generate a ciphertext of the DCE scheme which supports PoW verification and recovery, we first use a CE scheme to encrypt the file, then apply Rabin's IDA on the CE ciphertext to get an information piece, and finally use the piece as the DCE ciphertext.

Note that Rabin's IDA can be set to allow a recovery from at least K information pieces, so it allows us to thwart template side-channel attacks or server's covert deduplication. Although Rabin's IDA provides no secrecy protection to the dispersed information, this does not cause a problem to the file's confidentiality since we use it to disperse the ciphertext of the file in our construction. Lastly, due to the fact that the computation of information pieces is linear, as we will show shortly, it is easy to construct secure verification scheme which can be used to implement a secure PoW protocol. An illustrative example of our construction is presented in Fig. 1.

Below we introduce $\mathcal{R}{-}\mathcal{DCE}1$ in more detail.

4.1.1 $\mathcal{R}{-}\mathcal{DCE}1.Setup()$

Given the security parameter $\lambda \in \mathbb{N}$ as input, the setup algorithm generates a random, λ-bits-long prime number p and a generator g of a multiplicative cyclic group G with order p such that the discrete-logarithm problem is hard relative to g and G. In addition, choose a secure convergent encryption scheme $\mathcal{CE}$. Return $(g, p, \mathcal{CE})$ as the output.

4.1.2 $\mathcal{R}-\mathcal{DCE}1.Gen()$

Given as input λ and the file to be encrypted F, $\mathcal{R}-\mathcal{DCE}1.Gen()$ computes the secret key as

$$k := \mathcal{CE}.Gen(F, 1^\lambda). \tag{9}$$

4.1.3 $\mathcal{R}-\mathcal{DCE}1.Enc()$

Given as input $F \in \{0,1\}^l$, a user's index $i \in \{1, 2, \ldots, N\}$, a secret key k, and a threshold $K \in \mathbb{N}$, $\mathcal{R}-\mathcal{DCE}1.Enc()$ first uses the convergent encryption scheme $\mathcal{CE}$ to encrypt the file and gets its ciphertext as

$$c := \mathcal{CE}.Enc_k(F). \tag{10}$$

Next, compute the DCE's ciphertext as the dot product of two K-dimension vectors

$$C\langle i\rangle := \vec{c} \cdot (1, i, i^2, \ldots, i^{K-1}) \bmod p, \tag{11}$$

where vector $\vec{c}$ is generated by parsing c into K d-bit integers[6] ($d = \lambda - 1$) as

$$\vec{c} = (\overline{c[1:d]}, \overline{c[d+1:2d]}, \ldots, \overline{c[(K-1)d+1:Kd]}), \tag{12}$$

where $\overline{c[j:j']}$ denotes the integer whose binary representation is

$$c[j]c[j+1]...c[j'].$$

Then, compute the auxiliary information

$$\vec{I} = (g^{\vec{c}[1]}, g^{\vec{c}[2]}, \ldots, g^{\vec{c}[K]}). \tag{13}$$

Finally, return $(C\langle i\rangle, \vec{I})$ as the output of $\mathcal{R}-\mathcal{DCE}1.Enc()$.

4.1.4 $\mathcal{R}-\mathcal{DCE}1.Vrfy()$

The verify algorithm takes a DCE ciphertext C, the index of user who uploads this ciphertext i, and the auxiliary information $\vec{I}$ as input, verifies the correctness of C by checking whether

$$g^C = \vec{I}[1] \times \vec{I}[2]^i \times \vec{I}[3]^{(i^2)} \times \ldots \times \vec{I}[K]^{(i^{K-1})} \tag{14}$$

holds. If yes, the algorithm outputs *Accept*, otherwise outputs *Reject*.

4.1.5 $\mathcal{R}-\mathcal{DCE}1.Recr()$

The recovery algorithm's input consists of two parts. The first one has K DCE ciphertexts of F. Denote the vector of these ciphertexts by

$$\vec{C} = (C\langle i_1\rangle, C\langle i_2\rangle, \ldots, C\langle i_K\rangle). \tag{15}$$

The second one contains the indexes of K different users who generate these ciphertexts

$$i_1, i_2, \ldots, i_K \in \{1, 2, \ldots, N\}.$$

According to the Rabin's IDA, the recovery algorithm first computes the product of the ciphertext and the inverse of a Vandermonde matrix

$$\vec{c} = \vec{C} \cdot \begin{pmatrix} 1 & 1 & \ldots & 1 \\ i_1 & i_2 & \ldots & i_K \\ \ldots & \ldots & \ldots & \ldots \\ i_1^{K-1} & {}_2^{K-1} & \ldots & i_K^{K-1} \end{pmatrix}^{-1}. \tag{16}$$

From $\vec{c}$, the corresponding CE ciphertext c can be easily recovered by concatenating the binary representations of $\vec{c}$'s elements.

4.1.6 $\mathcal{R}-\mathcal{DCE}1.Dec()$

The decryption algorithm takes a CE ciphertext c and a private key k as input, outputs

$$F := \mathcal{CE}.Dec(c, k). \tag{17}$$

4.2 Correctness

Theorem 1. *Given K different $\mathcal{R}-\mathcal{DCE}1$ ciphertexts of m , the recovery algorithm $\mathcal{R}-\mathcal{DCE}1.Recr$ always returns the convergent encryption ciphertext of m.*

Proof. According to Equation (11), we know

$$\vec{C} = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ i_1 & i_2 & \ldots & i_K \\ \ldots & \ldots & \ldots & \ldots \\ i_1^{K-1} & {}_2^{K-1} & \ldots & i_K^{K-1} \end{pmatrix} \times \vec{c}. \tag{18}$$

Since the $K \times K$ matrix above is a Vandermonde matrix which is always full-rank, we know it has an inverse. Thus, Equation (16) correctly computes the convergent encryption ciphertext's all pieces. □

Theorem 2. *With correct auxiliary information $\vec{I}$, the verification algorithm $\mathcal{R}-\mathcal{DCE}1.Very$ accepts and only accepts a ciphertext that is correctly computed.*

Proof. When the ciphertext C is correctly computed by user i, we know

$$C = C\langle i\rangle = \vec{c} \cdot (1, i, i^2, \ldots, i^{K-1}) \bmod p \tag{19}$$

$$= \vec{c}[1] + \vec{c}[2]i + \ldots + \vec{c}[K]i^{K-1} \bmod p. \tag{20}$$

In addition, we have

$$\vec{I}[k] = g^{\vec{c}[k]} \text{ for } k = 1, \ldots, K. \tag{21}$$

Therefore, we know

$$g^C = g^{\vec{c}[1]+\vec{c}[2]i+\ldots+\vec{c}[K]i^{K-1}} \tag{22}$$

$$= g^{\vec{c}[1]} \times g^{\vec{c}[2]i} \times \ldots g^{\vec{c}[K]i^{K-1}} \tag{23}$$

$$= \vec{I}[1] \times \vec{I}[2]^i \times \vec{I}[3]^{(i^2)} \times \ldots \times \vec{I}[K]^{(i^{K-1})}. \tag{24}$$

To see why the verification only accepts the correctly computed $C < i >$, please note that $C < i >$ is the only value in Z_p that makes the Equation (14) holds. □

6. Here we assume c's length equals Kd, this can be achieved by applying a padding scheme to the file before encryption. For the ease of presentation, we omit the steps of adding paddings and removing paddings in our protocols.

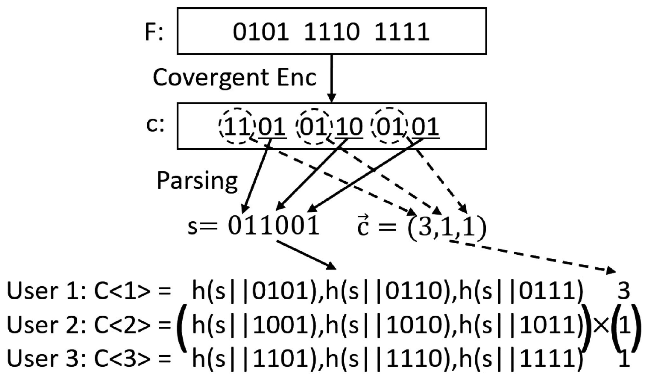

F: 0101 1110 1111
Covergent Enc
c: 1101 0110 0101
Parsing
s= 011001 c⃗ = (3,1,1)
User 1: C<1> = h(s||0101),h(s||0110),h(s||0111) 3
User 2: C<2> = h(s||1001),h(s||1010),h(s||1011) × 1
User 3: C<3> = h(s||1101),h(s||1110),h(s||1111) 1

Fig. 2. A user's computational overhead in $\mathcal{R}-\mathcal{DCE}1$-SDP, compared with Xu's scheme [27] (File size equals 1024K bit).

4.3 Security

Denote by $\mathcal{R}-\mathcal{DCE}1$-SDP our secure deduplication protocol that is constructed based on $\mathcal{R}-\mathcal{DCE}1$ scheme.

Theorem 3. *Assuming the maximum number of users that a Type-1 adversary can control equals δ, $\mathcal{R}-\mathcal{DCE}1$-SDP achieves $(K-\delta)$-anonymity against the template side-channel attack of this adversary.*

The correctness of this theorem is straightforward since the communication between each user and the server is authenticated. Therefore, the adversary cannot send a honest user's PoW evidence (even it can compute the evidence) to server to cheat.

Proof. Assuming the adversary $\mathcal{A}_1$ already has $K-1$ valid ciphertexts $(C\langle i_1\rangle,\ldots,C\langle i_{K-1}\rangle)$ and it tries to generate a new valid ciphertext $C\langle i_K\rangle$. It is easy to see computing $C\langle i_K\rangle$ is equivalent to finding out the correct $\vec{c}[1],\ldots,\vec{c}[K-1]$. Thus, to proof this theorem, we can prove the probability that $\mathcal{A}_1$ outputs $\vec{c}[1],\ldots,\vec{c}[K-1]$ correctly is negligible.

Let $C_j = C\langle i_j\rangle - i_j^{K-1}\cdot\vec{c}[K]$. Due to Eq. 16, we know

$$\begin{pmatrix}\vec{c}[1]\\ \ldots\\ \vec{c}[K-1]\end{pmatrix} = \begin{pmatrix}1 & i_1 & \ldots & i_1^{K-2}\\ 1 & i_2 & \ldots & i_2^{K-2}\\ \ldots & \ldots & \ldots & \ldots\\ 1 & i_2 & \ldots & i_{K-1}^{K-2}\end{pmatrix}^{-1} \times \begin{pmatrix}C_1\\ \ldots\\ C_{K-1}\end{pmatrix}. \tag{25}$$

From above, we know for each possible value of $\vec{c}[K]$, we have a different $\vec{c}[1],\ldots,\vec{c}[K-1]$. Notice there is only one correct $\vec{c}[K]$, and it is uniformly distributed in $[0,2^{\lambda-1}-1]$ (since we model $\mathcal{CE}$ as a random oracle), we know $Pr[\mathcal{A}\text{ succeeds}] = 1/2^{\lambda-1}$. □

Theorem 4. *For every polynomial-time bounded Type-3 adversary, the probability for it to successfully cheat $\mathcal{R}-\mathcal{DCE}1$-SDP (i.e., perform deduplication before receiving at least K ciphertexts without being able to be detected) is negligible.*

Proof. Suppose the Type-3 adversary $\mathcal{A}_3$ already has collected $K-1$ ciphertexts (and the auxiliary information for verification). For a Type-3 adversary $\mathcal{A}_3$ to successfully cheat, it has to be able to recover the underlying convergent encryption ciphertext, otherwise a honest can detect its cheating behavior by requesting to download the ciphertext. Notice that this is equivalent to require $\mathcal{A}_3$ to generate a new ciphertext $C\langle i\rangle$. Suppose $\mathcal{A}_3$ can generate correct $C\langle i\rangle$ with a non-negligible probability, this means $\mathcal{A}_3$ can solve the following discrete logarithm problem regarding g and G:

$$g^C = \vec{I}[1]\times\vec{I}[2]^i\times\vec{I}[3]^{(i^2)}\times\ldots\times\vec{I}[K]^{(i^{K-1})}, \tag{26}$$

where C can be similarly proved to be uniformly distributed in $[0,2^{\lambda-1}-1]$ under the constraints set by $K-1$ previously collected ciphertext. However, this is impossible according to our assumption about g and G. □

5 A DCE Construction Based on Randomized Rabin's IDA

Notice that our first construction $\mathcal{R}-\mathcal{DCE}1$ uses the fixed Vandermonde matrix of users' indexes to compute ciphertexts, therefore it can only defend the attack from Type-2 adversary who knows at most $K-1$ PoW evidences. In this section, we propose another construction that is secure against a Type-2 adversary who knows an arbitrary number of valid PoW evidences. Our main idea is to replace the Vandermonde matrix with a random matrix that is generated using (part of) the CE ciphertext. Since the adversary does not know the CE ciphertext, it does not know the matrix and cannot generate new valid evidence from the old ones. An illustrative example of our idea is presented in Fig. 2.

5.1 Construction of $\mathcal{R}-\mathcal{DCE}2$

5.1.1 $\mathcal{R}-\mathcal{DCE}2.Setup()$

$\mathcal{R}-\mathcal{DCE}2.Setup$ outputs what the setup algorithm $\mathcal{R}-\mathcal{DCE}1.Setup$ outputs, and a cryptographic hash function $h:\{0,1\}^*\rightarrow\{0,1\}^{\lambda-1}$.

5.1.2 $\mathcal{R}-\mathcal{DCE}2.Enc()$

Same as $\mathcal{R}-\mathcal{DCE}1.Enc()$, given a file F, $\mathcal{R}-\mathcal{DCE}2.Enc()$ encrypts the file as

$$k := \mathcal{CE}.Gen(F,1^\lambda), \tag{27}$$

$$c := \mathcal{CE}.Enc_k(F). \tag{28}$$

Let $l/K = u$ and $d=\lambda-1$.[7] Note that $\mathcal{R}-\mathcal{DCE}2$ adopts a new method to generate the two vectors that compute its ciphertext. Specifically, $\mathcal{R}-\mathcal{DCE}2$ generates the vector $\vec{c}$ by extracting K d-bit integers from c as follows.

$$\vec{c} = (\overline{c[1:d]},\overline{c[u+1:u+d]},\ldots,\overline{c[(K-1)u:(K-1)u+d]}). \tag{29}$$

Next, it computes an $(l-K\lambda)$-bits-long integer s by concatenating all remaining bits as

$$s = \overline{c[l,u]||c[u+l,2u]||\ldots||c[(K-1)u+l,Ku]}, \tag{30}$$

7. Making u an integer can be easily achieved by adopting a padding scheme or by setting K as a divisor of 1024 in practice.

To generate vector $\vec{e}$, $\mathcal{R}-\mathcal{DCE}2$ compute

$$e\langle\vec{i}\rangle = (h(s||i||1), h(s||i||2), \ldots, h(s||i||K)). \tag{31}$$

Next, compute the dot product of two K-dimension vectors

$$C\langle i\rangle := \vec{c} \cdot e\langle\vec{i}\rangle \bmod p. \tag{32}$$

Then, compute the auxiliary information $\vec{I}$ as the vector

$$\vec{I} = ((g^{\vec{c}[1]}, g^{\vec{c}[2]}, \ldots, g^{\vec{c}[K]}) \bmod p, s). \tag{33}$$

Finally, return $(k, C\langle i\rangle, I)$ as the output of $\mathcal{R}-\mathcal{DCE}1.Enc()$.

5.1.3 $\mathcal{R}-\mathcal{DCE}2.Vrfy()$

The verification algorithm takes a DCE ciphertext C, the index of user who uploads this ciphertext i, and the auxiliary information $\vec{I}$ as input, computes

$$s = \vec{I}[K+1], \tag{34}$$

$$e\langle\vec{i}\rangle[j] = h(s||i||j) \tag{35}$$

for every $j \in \{1, 2, \ldots, K\}$, and verifies the correctness of C by checking whether the following equation holds.

$$g^C = I[1]^{e\langle\vec{i}\rangle[1]} \times I[2]^{e\langle\vec{i}\rangle[2]} \times I[3]^{e\langle\vec{i}\rangle[3]} \times \ldots \times I[K]^{e\langle\vec{i}\rangle[K]} \tag{36}$$

5.1.4 $\mathcal{R}-\mathcal{DCE}2.Recr()$

Different from $\mathcal{R}-\mathcal{DCE}1$, $\mathcal{R}-\mathcal{DCE}2$'s recovery algorithm's input consists of three parts. The first one has K DCE ciphertexts of F

$$\vec{C} = (C\langle i_1\rangle, C\langle i_2\rangle, \ldots, C\langle i_K\rangle). \tag{37}$$

The second one contains the indexes of K different users who generate these ciphertexts $i_1, i_2, \ldots, i_K$. And the third one is s, which is the last element of the auxiliary information $\vec{I}$.

$\mathcal{R}-\mathcal{DCE}2.Recr()$ computes

$$e\langle\vec{i}\rangle[j] = h(s||i||j) \tag{38}$$

for every $j \in \{1, 2, \ldots, K\}$, and recovers a part of the CE ciphertext as

$$\vec{c} = \vec{C} \cdot \begin{pmatrix} e\langle\vec{i_1}\rangle[1] & e\langle\vec{i_2}\rangle[1] & \ldots & e\langle\vec{K}\rangle[1] \\ e\langle\vec{i_1}\rangle[2] & e\langle\vec{i_2}\rangle[2] & \ldots & e\langle\vec{K}\rangle[2] \\ \ldots & \ldots & \ldots & \ldots \\ e\langle\vec{i_1}\rangle[K] & e\langle\vec{i_2}\rangle[K] & \ldots & e\langle\vec{K}\rangle[K] \end{pmatrix}^{-1}. \tag{39}$$

From $\vec{c}$ and s, the corresponding CE ciphertext c can be easily recovered by concatenating the binary representations of $\vec{c}$'s elements and s.

5.1.5 $\mathcal{R}-\mathcal{DCE}2.Dec()$

Same as $\mathcal{R}-\mathcal{DCE}1.Dec()$.

5.2 Correctness

Theorem 5. *Given K different $\mathcal{R}-\mathcal{DCE}2$ ciphertexts of m, the recovery algorithm $\mathcal{R}-\mathcal{DCE}2.Recr$ outputs correct convergent encryption ciphertext of m with a probability greater than $1 - K/2^{l-1}$.*

Proof. $\mathcal{R}-\mathcal{DCE}2.Recr$ returns the correct output if and only if the $K \times K$ matrix is full-rank. Denote by E_K this matrix and by E_{K-1} the $(K-1) \times (K-1)$ matrix at the lower right corner of E_K. In a similar manner, we define matrices $E_{K-2}, \ldots, E_1$ recursively. Denote by $d_K, \ldots, d_1$ the determinants of $E_K, \ldots, E_1$ respectively. Specifically, the determinant of E_K can be computed as

$$d_K = e\langle i_1\rangle[1] \cdot d_{K-1} + b, \tag{40}$$

where b equals the sum of other $K-1$ elements in the formula of matrix determinant computation. Based on the above equation, we know

$$Pr[Det(E_K) = 0] \tag{41}$$

$$= Pr[d_{K-1} = 0 \wedge b = 0] \tag{42}$$

$$+ Pr[d_{K-1} \neq 0 \wedge e\langle i_1\rangle[1] = -b/d_{K-1}] \tag{43}$$

$$\leq Pr[d_{K-1} = 0] + Pr[e\langle i_1\rangle[1] = -b/d_{K-1}] \tag{44}$$

$$\leq Pr[d_{K-1} = 0] + 1/2^{l-1} \tag{45}$$

$$\leq Pr[d_{K-2} = 0] + 2/2^{l-1} \tag{46}$$

$$\leq \ldots \tag{47}$$

$$\leq Pr[d_1 = 0] + (K-1)/2^{l-1} \tag{48}$$

$$\leq K/2^{l-1}, \tag{49}$$

where Equations (43) $\leq$ (44) and Equations (44) $\leq$ (45) because elements in the matrix E_K are independently and uniformly distributed, and the rest inequations are due to recursion of Equation (45), given each element in E_K is an evaluation of h on a different input and h is a cryptographic hash. □

Theorem 6. *With correct auxiliary information $\vec{I}$, the verification algorithm $\mathcal{R}-\mathcal{DCE}2.Very$ accepts and only accepts a ciphertext that is correctly computed.*

Theorem 6's correctness is straightforward to see and the proof of it is basically the same as the proof to Theorem 2. To avoid redundancy, we omit the proof here.

5.3 Security

Denote by $\mathcal{R}-\mathcal{DCE}2$-SDP our secure deduplication protocol that is constructed based on $\mathcal{R}-\mathcal{DCE}2$ scheme.

Theorem 7. *Assuming the maximum number of users that a Type-1 adversary can control equals δ, $\mathcal{R}-\mathcal{DCE}2$-SDP achieves $(K-\delta)$-anonymity against the template side-channel attack of this adversary.*

Theorem 8. *For every polynomial-time bounded Type-3 adversary, the probability for it to successfully cheats $\mathcal{R}-\mathcal{DCE}2$-SDP (i.e., perform deduplication before receiving at least K ciphertexts without being able to be detected) is negligible.*

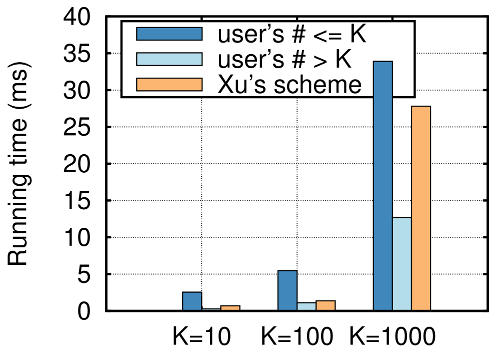


Fig. 3. An illustrative example of $\mathcal{R}-\mathcal{DCE}2.Enc$ with $K=3, l=3, p=5$. All matrix computations are performed over $\mathbb{Z}_5$.

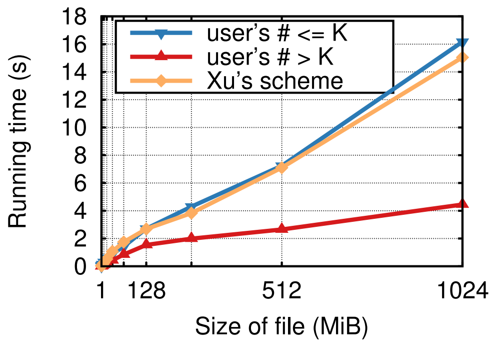


Fig. 4. A user's computational overhead in $\mathcal{R}-\mathcal{DCE}2$-SDP, compared with Xu's scheme [27] (K = 100).

The proofs of Theorems 7 and 8 are basically the same as the proofs for Theorem 3 and Theorem 4. To avoid redundancy, we omit their proofs.

Theorem 9. *For every polynomial-time bounded Type-2 adversary who have unlimited access to an oracle that outputs a valid ciphertext or PoW evidence, the probability for it to successfully cheat $\mathcal{R}-\mathcal{DCE}2$-SDP (i.e., generate a new valid ciphertext) is negligible.*

Proof. To see why Theorem 9 is correct, please note that, different from $\mathcal{R}-\mathcal{DCE}1$, the elements in the vector or matrix which we use to compute the ciphertext of DCE from the CE ciphertext are uniformly random (since they are generated by a cryptographic hash). Therefore the ciphertexts that the adversary receives from the oracle are no different from random numbers. □

6 Evaluations

In this section, we conduct experiments to evaluate the performance of our $\mathcal{R}-\mathcal{DCE}1$-SDP and $\mathcal{R}-\mathcal{DCE}2$-SDP protocols. All non-cryptographic computations are implemented using C++ programming language, and cryptographic computations are implemented using Crypto++ library. Security parameter λ is set to 1024, hash functions are implemented as SHA256, and encryptions are implemented as AES256 in CBC mode. Programs are run on a Inter E5 CPU @3.50 GHz server running Ubuntu 12.04 64-bit OS with 16GB memory. Experiments results presented below have been averaged over 50 runs.

TABLE 2
Ratio of Running Time to Uploading Time, K=100

File size + SPD protocol	1Mb/s	5Mb/s
100Kibit + $\mathcal{R}-\mathcal{DCE}1$-SDP	5.3%	26.7%
16MiB + $\mathcal{R}-\mathcal{DCE}2$-SDP	1.2%	6.0%
1024MiB + $\mathcal{R}-\mathcal{DCE}2$-SDP	0.2%	1.0%

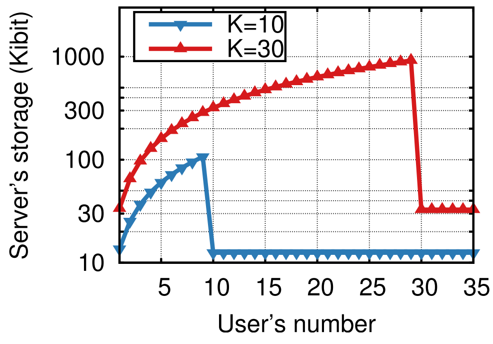


Fig. 5. Storage cost of the cloud storage server with $\mathcal{R}-\mathcal{DCE}1$-SDP.

Efficiency. we first evaluate our protocols' efficiency by measuring a user's computational overhead. To provide an clear idea, we also include one state-of-art secure deduplication protocol that is recently proposed by Xu et al. [27] in our test. Fig. 3 shows the comparison of a user's computational overhead in $\mathcal{R}-\mathcal{DCE}1$-SDP and in Xu's scheme. We can see two protocols have a comparable efficiency. Especially, we can see the user's overhead in $\mathcal{R}-\mathcal{DCE}1$-SDP before deduplications is slightly greater than its overhead in Xu's scheme before deduplication, and becomes much smaller after deduplication. This is because the user has to compute relatively expensive exponentiations to generate auxiliary information for verification before deduplication happens. Similar results can be found in Fig. 4 which shows the comparison between $\mathcal{R}-\mathcal{DCE}2$-SDP and Xu's scheme. We notice that user's computational overhead in $\mathcal{R}-\mathcal{DCE}2$-SDP is heavier compared with the overhead in $\mathcal{R}-\mathcal{DCE}1$-SDP which is mainly due to the additional hash value computations in $\mathcal{R}-\mathcal{DCE}2$-SDP.

Overall, both our protocols and Xu's scheme are highly efficient. In fact, as shown in Table 2, when comparing user's computation time in our protocols with its uploading time of the file, user's computational overhead becomes negligible especially when file size is over 1 GB.

Effectiveness of Deduplication. Another important factor of evaluation is the effectiveness of deduplication. Here we observe deduplication from server's storage status. We point out that our results here consider the storage overhead which is introduced by our scheme. Necessary cost in all cloud storage (like linkages or pointers that work for data management) is not considered by us.

Specifically, Fig. 5 shows the change of server's storage that is occupied as more and more users upload a same file. We can see server has a large amount of space released when the K-th user finishes uploading. Also, occupied storage's increasing speed is more relevant to the size of file, than to the value of K. In Fig. 6, increasing speed is not significant because of the large range of Y-axis. But we can

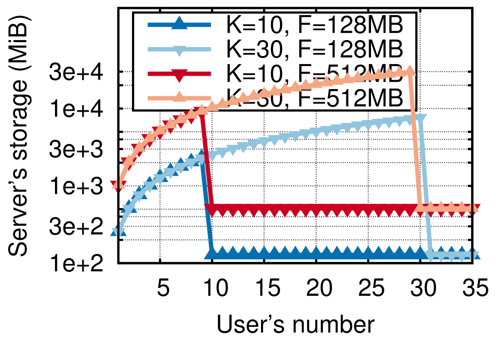


Fig. 6. Storage cost of the cloud storage server in $\mathcal{R}-\mathcal{DCE}2$-SDP.

still tell that after server deletes redundancy, the storage cost is even smaller than the first user's cost. This is because server needs to store remaining bits of c for $\mathcal{R}-\mathcal{DCE}2$-DSP users. It is also possible for $\mathcal{R}-\mathcal{DCE}1$-DSP to have less storage cost than the first user's cost if the server stores vector $\vec{C}$ instead of encrypted file because some information of file's ciphertext is implied by coefficients which can be easily recovered from users' indexes.

7 Related Works

Many works including most early ones (e.g [5], [6], [14], [24]) on secure deduplication focus on server-side deduplications. These works generally aim to increase the opportunity for deduplications in different storage environments by designing specific encryption and uploading schemes that are used by data uploaders. For example, in [14] authors consider a multi-cloud storage system, and design a novel information dispersal scheme called convergent dispersal that assigns same dispersed pieces to a same cloud server, so that deduplications can be easily on each cloud server locally. Different from these works, we consider the client-side deduplication.

Client-side deduplication can happen within each user's account or across multiple users' accounts. For the previous case, simple solutions such as per-user encryption key [19] exist. For the latter case, designing secure protocols is much more complicated. Specifically, one of the key problems for cross-user secure deduplication is how to let all uploaders of the same file choose or acquire the same encrypting/decryption key. Based on how this problem is solved, existing secure cross-user client-side deduplication can be divided into two categories. The first one which is also the most widely used is based on convergent encryption (e.g., [12], [17], [23]) or its variation (e.g., [1], [13], [27]). The second one is based on secure communication protocols between different users (e.g., [15]). Compared with solutions in the first category, the solution proposed in [15] can achieve semantic security guarantee without introducing independent servers. However, it requires previous uploaders to be online and participate in communication protocol with current uploader.

Another key problem for cross-user deduplication is how to authenticate deduplications, or how to make sure the user do have the file it requests to upload when the file upload would get deduplicated. Failing to achieve this goal may allow adversaries to perform attacks such as hash-only attack (e.g., [16]) or content-distributed-network (CDN) attack [8]. To solve this problem, Halevi et al. [7] first propose a solution called "proofs of ownership". After this work, many researchers have proposed their PoW protocols focusing either on improving the efficiency (e.g., [2], [28]) and/or improve the security strength (e.g., [27]).

In our paper, besides the above two problems, we also take the template side-channel attack which exploits an inherent weakness of deduplication systems (i.e., the side-channel information regarding whether a deduplication happens is easy to acquired by adversaries) into consideration when designing our secure deduplication protocols. Our solution is constructed based on the well known k-anonymity privacy concept and threshold-based deduplication. We note that introducing a threshold into deduplication system to mitigate side-channel information leakage is not our invention. There have been a few works [8], [11], [20], [23] that propose to introduce a threshold and/or probabilistic upload into deduplication protocols. However, either they assume there is a honest server (e.g., [8], [11]) or a trusted independent server (e.g., [20], [23]) who is willing to help to perform the threshold deduplication.

Finally, we note there have been also many works studying miscellaneous topics related to secure deduplication and providing interesting results in recent years. For example, the secure deduplication of encrypted data in the attribute-based cloud storage is studied in [3], the convergent encryption key management problem is studied in [12], secure deduplication systems that are specifically designed for cloud media centers are proposed in [29], [30]. For more detailed introductions of recent secure deduplication proposals, we refer readers to two excellent recent survey works [19], [21].

8 Conclusion

In this paper, we study the problem of thwarting template side-channel attack in client-side secure duplication systems on the cloud with a covert server adversary who may trigger the deduplication before the predefined threshold is reached. By introducing the k-anonymity privacy concept into our design, we have devised a novel cryptographic primitive called dispersal convergent encryption scheme which can be used to construct efficient secure deduplication protocol satisfying our requirements. We also provide two practical constructions of DCE schemes, and theoretically prove their excellent security guarantees against three kinds of important adversaries. Experiment results show our secure deduplication protocols achieve very good performance.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2018YFB1004301), NSFC-61872179, NSFC-61321491, NSFC-61425024, NSFC-61872176, NSFC-61872180, Jiangsu "Shuang-Chuang" Program, Jiangsu "Six-Talent-Peaks" Program, and Ant Financial through the Ant Financial Science Funds for Security Research.

References

[1] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Tech.*, 2013, pp. 296–312.
[2] J. Blasco, R. Di Pietro, A. Orfila, and A. Sorniotti, "A tunable proof of ownership scheme for deduplication using bloom filters," in *Proc. IEEE Conf. Commun. Netw. Secur.*, 2014, pp. 481–489.

[3] H. Cui, R. H. Deng, Y. Li, and G. Wu, "Attribute-based storage supporting secure deduplication of encrypted data in cloud," *IEEE Trans. Big Data*, 2017.

[4] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proc. 7th ACM Symp. Inf. Comput. Commun. Secur.*, 2012, pp. 81–82.

[5] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst.*, 2002, pp. 617–624.

[6] J. R. Douceur, W. J. Bolosky, and M. M. Theimer, "Encryption systems and methods for identifying and coalescing identical objects encrypted with different keys," U.S. Patent 6983365, Jan. 3, 2006.

[7] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 491–500.

[8] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Secur. Privacy*, vol. 8, no. 6, pp. 40–47, Nov./Dec. 2010.

[9] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols: Techniques and Constructions*. New York, NY, USA: Springer Science & Business Media, 2010.

[10] T. Jiang, X. Chen, Q. Wu, J. Ma, W. Susilo, and W. Lou, "Secure and efficient cloud data deduplication with randomized tag," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 3, pp. 532–543, Mar. 2017.

[11] S. Lee and D. Choi, "Privacy-preserving cross-user source-based data deduplication in cloud storage," in *Proc. Int. Conf. ICT Convergence*, 2012, pp. 329–330.

[12] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1615–1625, Jun. 2014.

[13] J. Li, Y. K. Li, X. Chen, P. P. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1206–1216, May 2015.

[14] M. Li, C. Qin, P. P. Lee, and J. Li, "Convergent dispersal: Toward storage-efficient security in a cloud-of-clouds," in *Proc. 6th USENIX Workshop Hot Topics Storage File Syst.*, 2014, pp. 1–5

[15] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 874–885.

[16] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. R. Weippl, "Dark clouds on the horizon: Using cloud storage as attack vector and online slack space," in *Proc. USENIX Secur. Symp.*, 2011, pp. 65–76.

[17] P. Puzio, R. Molva, M. Önen, and S. Loureiro, "Cloudedup: Secure deduplication with encrypted data for cloud storage," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci.*, 2013, vol. 1, pp. 363–370.

[18] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM (JACM)*, vol. 36, no. 2, pp. 335–348, Apr. 1989.

[19] V. Rabotka and M. Mannan, "An evaluation of recent secure deduplication proposals," *J. Inf. Secur. Appl.*, vol. 27, pp. 3–18, 2016.

[20] Y. Shin and K. Kim, "Differentially private client-side data deduplication protocol for cloud storage services," *Secur. Commun. Netw.*, vol. 8, no. 12, pp. 2114–2123, 2015.

[21] Y. Shin, D. Koo, and J. Hur, "A survey of secure data deduplication schemes for cloud storage systems," *ACM Comput. Surveys*, vol. 49, no. 4, 2017, Art. no. 74.

[22] J. Stanek and L. Kencl, "Enhanced secure thresholded data deduplication scheme for cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 4, pp. 694–707, Jul./Aug. 2018.

[23] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, "A secure data deduplication scheme for cloud storage," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2014, pp. 99–118.

[24] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure data deduplication," in *Proc. 4th ACM Int. Workshop Storage Secur. Survivability*, 2008, pp. 1–10.

[25] L. Sweeney, "k-anonymity: A model for protecting privacy," *Int. J. Uncertainty Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, 2002.

[26] C. Wang, Z.-g. Qin, J. Peng, and J. Wang, "A novel encryption scheme for data deduplication system," in *Proc. Int. Conf. Commun. Circuits Syst*, 2010, pp. 265–269.

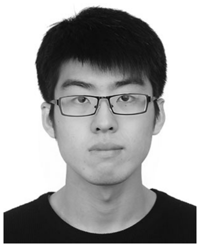
[27] J. Xu, E.-C. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *Proc. 8th ACM SIGSAC Symp. Inf. Comput. Commun. Secur.*, 2013, pp. 195–206.

[28] C.-M. Yu, C.-Y. Chen, and H.-C. Chao, "Proof of ownership in deduplicated cloud storage with mobile device efficiency," *IEEE Netw.*, vol. 29, no. 2, pp. 51–55, Mar./Apr. 2015.

[29] Y. Zheng, X. Yuan, X. Wang, J. Jiang, C. Wang, and X. Gui, "Enabling encrypted cloud media center with secure deduplication," in *Proc. 10th ACM Symp. Inf. Comput. Commun. Secur.*, 2015, pp. 63–72.

[30] Y. Zheng, X. Yuan, X. Wang, J. Jiang, C. Wang, and X. Gui, "Toward encrypted cloud media center with secure deduplication," *IEEE Trans. Multimedia*, vol. 19, no. 2, pp. 251–265, Feb. 2017.

Yuan Zhang received the BS degree in automation from Tianjin University, in 2005, the MSE degree in software engineering from Tsinghua University, in 2009, and the PhD degree in computer science from State University of New York at Buffalo, in 2013. His research interest include security, privacy, and economic incentives.



Yunlong Mao received the BS and PhD degrees in computer science from Nanjing University, in 2013 and 2018, respectively. He is currently an assistant researcher with the Department of Computer Science and Technology. His current research interests include security, privacy and machine learning.

Minze Xu is working toward the PhD degree at Nanjing University, majoring in computer science and technology. His central research interest includes information security and privacy.

Fengyuan Xu received the PhD degree, with the Distinguished Dissertation Award, from the College of William and Mary. He is currently a professor with the Computer Science Department, Nanjing University. His research interests include the broad areas of systems and security, with a focus on data driven security analytics, deep learning security & applications, mobile & edge computing, and trusted execution environments. He is a member of the IEEE.

Sheng Zhong received the BS and MS degrees from Nanjing University, in 1996 and 1999, respectively and the PhD degree from Yale University, in 2004, all in computer science. His research interest include security, privacy, and economic incentives.