

Server-aided access control for cloud computing

WENG Jian, WENG Jia-si, LIU Jia-nan, HOU Lin

(College of Information Science and Technology, Jinan University, Guangzhou 510632, China)

Abstract: With the massive diffusion of cloud computing, more and more sensitive data is being centralized into the cloud for sharing, which brings forth new challenges for the security and privacy of outsourced data. To address these challenges, the server-aided access control (SAAC) system was proposed. The SAAC system builds upon a variant of conditional proxy re-encryption (CPRE) named threshold conditional proxy re-encryption (TCPRE). In TCPRE, t out of n proxies can re-encrypt ciphertexts (satisfying some specified conditions) for the delegator (while up to $t-1$ proxies cannot), and the correctness of the re-encrypted ciphertexts can be publicly verified. Both features guarantee the trust and reliability on the proxies deployed in the SAAC system. The security models for TCPRE were formalized, several TCPRE constructions were proposed and that our final scheme was secure against chosen-ciphertext attacks was proved.

Key words: threshold conditional proxy re-encryption, server-aided access control, cloud computing, chosen-ciphertext attack

doi: 10.11959/j.issn.2096-109x.2016.00104

1 Introduction

Cloud computing is an emerging computing paradigm in which IT resources and capacities are provided as services over the Internet while hiding platform and implementation details. Nowadays, many users transparently rely on outsourced storage solutions based on the cloud model, e.g., by using Dropbox and Google Docs. Unfortunately, in order to effectively protect the secrecy of their data, such users can only encrypt them locally and then upload the ciphertext to the cloud storage repository. However, if they use traditional public key encryption schemes, some shortcomings may heavily condition the sharing of such data with other authorized users. For example, suppose Alice wants to share her private data with Bob, then she has to first download the ciphertexts from the cloud, decrypt them locally with her own secret key, encrypt again the data by

using Bob's public key, and finally upload the new ciphertext to the cloud. Then Bob can access the data with his secret key. However, such a solution is highly unsatisfactory, since it introduces heavy computational cost and communication overhead and does not scale well when data must be shared with a significant number of other users.

Fortunately, the cryptographic primitive of proxy re-encryption (PRE)^[1], which enables a proxy to transform a ciphertext under a delegator's public key into another ciphertext under the delegatee's public key, without learning anything about the content of the encrypted message, can be used to efficiently address this problem. Specifically, in the above storage outsourcing scenario, Alice uses a PRE scheme to encrypt her data and then upload the ciphertext to the cloud. When she wants to share the data with Bob, she can simply give a re-encryption key to a proxy in the cloud, and then the latter can

Received date: 2016-07-12, **Revised date:** 2016-08-23. **Corresponding author:** WENG Jian, cryptjweng@gmail.com

Foundation Item: The National Natural Science Foundation of China (No.61272413, No.61472165)

efficiently transform these ciphertexts into the ciphertexts intended for Bob, who can then decrypt the ciphertext to obtain the plaintext data with his own secret key.

Nevertheless, there still exist some problems which are hard to be tackled with a traditional PRE scheme. Let us take the above cloud-based storage application as an example again. Suppose some of Alice's data are highly sensitive, and she wants to decrypt these ciphertexts only by herself. Unfortunately, in the above scenario, with the re-encryption key, the proxy can transform all of Alice's ciphertexts, including the highly sensitive ones, and thus also Bob can decrypt them by autonomously obtaining these particular contents. To address this problem, conditional proxy re-encryption (CPRE) was introduced in Refs.[2,3]. In a CPRE scheme, generated ciphertexts associated with a certain condition, and the proxy can translate only those ciphertexts satisfying the specified condition. In the above cloud-based scenario, by relying on CPRE, Alice becomes able to control the proxy to transform only non-highly-sensitive ciphertexts.

Although CPRE enables the delegator to implement fine-grained delegation of decryption rights for securing information sharing, it suffers from two limitations. The first is that CPRE cannot guarantee the correctness of the transformation done by the proxy. This is indeed a challenge for the applicability of the proxy re-encryption technology in cloud storage services. For example, in the typical pay-per-use model characterizing cloud computing services, the proxy charges the customer (e.g., Alice) for the transformation efforts. Thus, for saving time and reducing the computational cost, the proxy might simply return ciphertexts which are not really generated by using the re-encryption algorithm. Unfortunately, existing CPRE solutions do not allow the users to check for such a malicious behavior of the proxy. The second is that common CPRE architectures only involve a single proxy.

This inevitably faces with the single point of failure problem: if the proxy is out of service, the delegates lose their ability to access the data. Thus, new solutions are needed to deal with these CPRE problems.

To address the above limitations, we propose a server-aided access control system, namely the SAAC system. The SAAC system is built upon a variant of CPRE primitive, named threshold conditional proxy re-encryption (TCPRE) primitive. The ciphertext in TCPRE is associated with a specified condition, and the proxies can only successfully transform those ciphertexts satisfying the specified condition. But unlike CPRE, TCPRE involves a number n of proxies and t out of n proxies can successfully transform ciphertexts, while up to $t-1$ proxies cannot. In addition, the correctness of the transformation done by each proxy can be publicly verified.

We formalize the TCPRE primitive and define the chosen-ciphertext security of TCPRE with respect to two types of ciphertexts (i.e., original ciphertexts and transformed ciphertexts). We present three constructions of TCPRE in a "gradual" way. The first one is a CPRE scheme, that allows the owner of some data stored in a specific file, to share it through (conditionally) delegating decryption capability. The second construction, based on the first one, adds the support of threshold-based proxy re-encryption to address the single-point-of-failure limitation. The third construction (that is the final one) additionally embeds the capability of verifying the correctness of the re-encryption done by the access control servers, and achieves CCA security. We only provide security and performance analysis of the final construction, that is the most complete one.

2 Preliminaries

We briefly review some preliminaries by first summarizing the notations used in this work, and

then introducing the bilinear map technique, which serves as the foundation of the proposed approach.

2.1 Notations

We denote Z_p as the set $\{0, 1, 2, \dots, p-1\}$ and Z_p^* as $Z_p \setminus \{0\}$. For a finite set S , $x \in S$ (or $x \in_R S$) means choosing an element x from S (with a uniform distribution). We define the Lagrange coefficient $\Delta_{i,S}$ for $i \in Z_p$ and a set S of elements in Z_p : $\Delta_{i,S} = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$. For a string m , $|m|$ denotes its bit-length. For all strings x and y , let $x \parallel y$ denotes the concatenation of x and y . $A(x_1, x_2, \dots)$ denotes that A is an algorithm with the input $(x_1, x_2, \dots)$, an $y = A(x_1, x_2, \dots)$ denotes the running of $A(x_1, x_2, \dots)$ with the output y . For $A^{O_1(\cdot), O_2(\cdot), \dots}(x_1, x_2, \dots)$, it means that A is an algorithm with the input $(x_1, x_2, \dots)$ and can access to the oracles $O_1(\cdot), O_2(\cdot), \dots$. By $y = A^{O_1(\cdot), O_2(\cdot), \dots}(x_1, x_2, \dots)$, we denote the running of $A^{O_1(\cdot), O_2(\cdot), \dots}(x_1, x_2, \dots)$ with the output y .

Besides, we briefly summarize the notations in Table 1 that will be used in the following.

Table 1 Notations used in proposed system

Notations	Descriptions
(pk_i, sk_i)	i -th user's public and secret key pair
w	Condition
rk_{ij}	Re-encryption key from i -th user to j -th user
vk_{ij}	Verification key from i -th user to j -th user
θ_{ij}	Re-encryption share from i -th user to j -th user
$[\cdot]_v$	v -th share of a secret
$L(\cdot)$	Polynomial implicitly defined in our system
ct	Original ciphertext
ct'	Transformed ciphertext
f	File
k	File encryption/decryption key
(n, t)	Number of proxies and the threshold

2.2 Bilinear Maps

A Bilinear Map is a common tool used for designing cryptographic primitives. In this subsection, we give the definition of the bilinear map, which

will be used in this paper.

Definition 1 Bilinear Map

Denote G, G_T as cyclic groups of prime order p , writing the group action multiplicatively. g is a generator of G . Let $e: G \times G \rightarrow G_T$ be a map with the following properties.

- 1) Bilinearity: $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for all $g_1, g_2 \in G$, and $a, b \in_R Z_p$.
- 2) Non-degeneracy: There exists $g_1, g_2 \in G$ such that $e(g_1, g_2) \neq 1$, in other words, the map does not send all pairs in $G \times G$ to the identity in G_T .
- 3) Efficiency: There exists an efficient polynomial time algorithm to compute the bilinear map $e(\cdot, \cdot)$.

3 System model

3.1 The architecture

As shown in Figure 1, we consider three types of entities in our access control system.

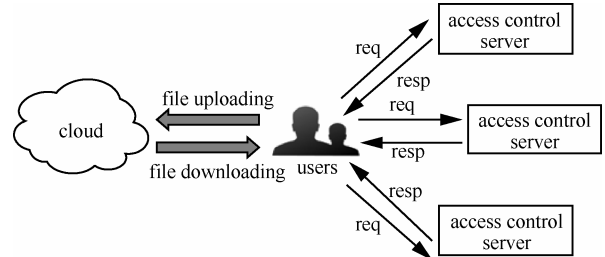


Figure 1 Access control system architecture

1) The Cloud, that is the entity providing out-sourced storage facilities. It is in charge of controlling the accesses from outside users to the data, stored by using specific servers that provide the corresponding content delivery services. In this paper, we assume that the cloud storage service provider (S-CSP) is always online and has unlimited storage capacity.

2) The users of the storage services, whose data are structured into files. For each file, we consider two types of users. The former is the owner, who decided to outsource his data into the cloud servers for storing and making them accessible in a controlled way. The latter one is the generic file user that, according to the permissions granted by the

owner, want to access some of the outsourced data files by relying on specific access control servers.

3) Multiple access control servers working as proxies and providing security services. More specifically, they transform the "raw" ciphertexts into "selectively decryptable" ones, so that only authorized users can decrypt them and then gain access to private contents.

It is worthwhile noting that, this type of architecture has several advantages. ① it does not involve any computational process, being deployed according to a thin cloud paradigm, just providing the LIST, PUT and GET interfaces^[4]. It also brings the benefits of portability and easy provider switching. ② Compared with the previous work^[5], we introduce multiple access control servers, not only balancing the computational burden, but also removing the presence of single points of failure.

3.2 System overview

Next, we give an overview of the proposed access control system. There are three phases involving its activity.

1) System setup. In this phase, the system parameters are initialized. Users are assigned their secret keys, and the access control servers are given the global private parameters.

2) File outsourcing. When a file owner wants to outsource a file for storage, he/she encrypts and uploads the file to cloud.

3) File access. When a file user wants to access an outsourced file, he/she firstly downloads the cipher text, requests access permission from file owner, and decrypts the cipher text with the help of access control servers.

3.3 Threat model and security goal

Based on the architecture proposed above, we have three threat assumptions. 1) The channels (connecting users to cloud and access control servers) can be public, and the data transferred by using these channels might be eavesdropped by outside attackers. 2) The cloud and access control servers are "honest-but-curious". They follow the proposed

protocol, but try to extract as much secret information as possible. 3) Some users might be compromised. We note that, this threat model is stronger than the one already proposed in^[5,6], which also considers the access control problem in cloud. More specifically, a fully trusted authority is assumed to exist in Refs.[6], becoming a single-point-of-failure if it is compromised, while we do not need to "put all the eggs in a single basket". In addition, the private key channel is assumed to be private in Refs.[6], but we do not need this assumption.

The security goal considered in this paper is providing semantic security under chosen ciphertext attacks (SS-CCA). In other words, it requires that any useful information about the private data stored into files, cannot be extracted by attackers, even if they have chances to access one or more known ciphertexts/plaintexts into the system and obtain the resulting plaintexts. Thus, it differentiates from the previous works^[5,6] which only achieve the same security goal for the (weaker) chosen plaintext attacks.

4 System framework

In order to realize the SAAC system, we introduce a new primitive, named threshold conditional proxy re-encryption (TCPRE). In what follows, we will present the formal definition of TCPRE and show how it is used to construct the SAAC system in section 3.

4.1 Threshold conditional proxy re-encryption

Before formalizing the definition for TCPRE, we first give a brief overview for TCPRE in the literature of proxy re-encryption. As with conditional proxy re-encryption, a TCPRE-based system runs the global setup algorithm Setup to generate the global parameters param, and each user runs the key generation algorithm KeyGen to generate his/her public/secret key pair. The ciphertext is also generated with respect to a specified condition by using the encryption algorithm Encrypt. Unlike CPRE, the user in TCPRE can designate a number n of proxies.

The user can generate n re-encryption key shares and verification key shares with the ReKeyGen algorithm. Each re-encryption key share is kept secret by a proxy, and all the verification key shares are made public. By running the ReEncShare algorithm, the proxy can generate a re-encryption share, whose validity can be publicly verified by the ShareVerify algorithm. With the ShareCombine algorithm, at least t valid re-encryption shares can be used to generate a valid transformed ciphertext, which can then be decrypted by the delegatee.

Formally, a TCPRE scheme consists of the following algorithms.

1) *Setup*(k): The global setup algorithm taking as input a security parameter k , and generating the global parameters $param$.

2) *KeyGen*($param$): The key generation algorithm taking as input the global parameters $param$, and generating user's public and secret key pair (pk_i, sk_i) .

3) *ReKeyGen*(sk_i, pk_j, w, t, n): The re-encryption key generation algorithm taking as input the delegator's secret key sk_i , a delegatee's public key pk_j , a condition w , a threshold t and a number n of proxies. It generates n shares of the re-encryption keys $\{[rk_{ij}]_v\}_{v=1}^n$ and verification keys $\{[vk_{ij}]_v\}_{v=1}^n$.

4) *Encrypt*(pk_i, m, w): The encryption algorithm taking as input user's public key pk_i , a message m and a condition w . It returns an original ciphertext ct_i .

5) *ReEncShare*($[rk_{ij}]_v, ct_i$): The re-encryption algorithm taking as input the v -th re-encryption key share $[rk_{ij}]_v$ and the ciphertext ct_i . It generates the v -th re-encryption share $[\theta_{ij}]_v$.

6) *ShareVerify*($ct_i, [vk_{ij}]_v, [\theta_{ij}]_v$): The share verification algorithm taking as input the original ciphertext ct_i , the v -th verification key share $[vk_{ij}]_v$ and the re-encryption share $[\theta_{ij}]_v$. It returns 1 if $[\theta_{ij}]_v$ is a valid re-encryption share; otherwise it returns 0 indicating $[\theta_{ij}]_v$ is invalid.

7) *ShareCombine*($ct_i, \{[\theta_{ij}]_v\}_{v \in S}, \{[vk_{ij}]_v\}_{v \in S}$):

The share combination algorithm taking as input the original ciphertext ct_i , a set of valid re-encryption shares $\{[\theta_{ij}]_v\}_{v \in S}$ with $|S| \geq t$ and a set of verification key shares $\{[vk_{ij}]_v\}_{v \in S}$. It generates a transformed ciphertext ct'_j .

8) *Decrypt*(sk_i, ct_i): The decryption algorithm, taking as input the user's secret key sk_i and an original/transformed ciphertext. It returns the original message m .

Suppose a TCPRE scheme has been initialized as $param = Setup(k)$ and $(pk_i, sk_i) = KeyGen(param)$. The correctness of TCPRE requires that, for any message m and condition w , the following equations hold

$$Decrypt(sk_i, Encrypt(pk_i, m, w)) = m \quad (1)$$

$$ShareVerify(ct_i, [vk_{ij}]_v, ReEncShare([rk_{ij}]_v, ct_i)) = 1 \quad (2)$$

$$Decrypt(sk_j, ShareCombine(ct_i, \{ReEncShare([rk_{ij}]_v, ct_i)\}, \{[vk_{ij}]_v\}_{v \in S})) = m \quad (3)$$

We explain these equations below. Equation (1) defines the consistency of the original ciphertext. It requires that the ciphertext encrypted with the i -th user's public key must be decrypted by using his/her secret key. Equation (2) defines the verifiability on encryption share: the correctness of each re-encryption share computed from a ciphertext can be verified only by using its corresponding verification key share. Equation (3) defines the consistency of transformed ciphertext: the ciphertext transformed from original ciphertext can be decrypted by using the delegatee's secret key.

4.2 Generic construction

Based on the definition of TCPRE, we give a generic construction of our SAAC system. Note that in the generic construction, all the algorithms involved in TCPRE are used as black boxes, the realization of which will be explained later.

1) System setup. In order to initialize the system, a security parameter k and the number n of proxies are chosen. The global setup algorithm

$Setup(k)$ of TCPRE is run to produce the parameters $param$. For each user in the system, the key generation algorithm $KeyGen(param)$ of TCPRE is used to assign him/her a pair of public and private key (pk_i, sk_i) . The secret key sk_i must be kept secret and the public key pk_i can be published.

2) File outsourcing. Suppose that a file f is to be uploaded by the i -th user. This user randomly pick a symmetric key k to encrypt f . Then he/she runs the encryption algorithm $Encrypt(pk_i, k, w)$ to compute the TCPRE ciphertext ct_i of k in respect to w which describes the condition with respect to f . Finally, the file ciphertext, key ciphertext, file identifier and the user identity are sent to cloud for storage.

3) File access. Any user can use the LIST operation to list the names of files in the cloud. For each needed file owned by user i , the generic user j downloads the key ciphertext ct_i and file ciphertext. The user takes the identifier of this file to request access permission from the owner. The file owner (the i -th user) runs the re-encryption key generation algorithm $ReKeyGen(sk_i, pk_j, w, t, n)$ of TCPRE and responds with the set of re-encryption key shares $\{[rk_{ij}]_v\}_{v=1}^n$ and verification key shares $\{[vk_{ij}]_v\}_{v=1}^n$. Notice that the threshold t should be chosen flexibly by the file owner for reliability consideration.

Upon receiving $\{[rk_{ij}]_v\}_{v=1}^n$ and $\{[vk_{ij}]_v\}_{v=1}^n$ the j -th user respectively sends the downloaded key ciphertext ct_i and the v -th key share $[rk_{ij}]_v$ to the v -th access control server. The v -th access control server runs the share re-encryption algorithm $ReEncShare([rk_{ij}]_v, ct_i)$ and responds with the v -th re-encryption share $[\theta_{ij}]_v$. After collecting at least t responses (the index of which constitute the set $\bar{S}$), the j -th user verifies their correctness by running the share verification algorithm $ShareVerify(ct_i, [vk_{ij}]_v, [\theta_{ij}]_v)$ for $v \in \bar{S}$. If the number of correct re-encryption shares is less than t , an error is reported. Otherwise, the j -th user selects t of the correct re-encryption shares (constituting the set

S) and runs the share combination algorithm $ShareCombine(ct_i, \{[\theta_{ij}]_v\}_{v \in S}, \{[vk_{ij}]_v\}_{v \in S})$ to get the transformed ciphertext ct'_j . Finally the j -th user decrypts ct'_j with his/her secret key sk_j to recover the file decryption key k , which is leveraged to decrypt the file ciphertext.

5 TCPRE constructions

From Section 4, we have learn how to leverage TCPRE (as a black box) to build the SAAC system. In this section, we will get into the black box of TCPRE and present several constructions. For better understanding, we use an incremental approach by providing our constructions in a "gradual" way. Our first construction is just a Conditional Proxy Re-Encryption (CPRE) scheme. Our second construction, based on the first one, adds the support of threshold-based proxy re-encryption. Our third construction (the final, and more complete one) additionally embeds verifiability on the re-encryption shares, and achieves CCA security.

5.1 First construction

The first construction, related to CPRE facilities, is a special case of the more general definition of TCPRE presented in section 4.1. Specifically, here we fix the number of access control servers $n = 1$ and the threshold value $t = 1$. Also, since we neither study verifiability nor allow multiple shares, the algorithms $ShareVerify(\cdot)$ and $ShareCombine(\cdot)$ are not considered.

We give our first construction as follows.

1) $Setup(k)$: Given a security parameter k , choose bilinear map groups (G, G_T) of prime order $p > 2^k$, and pick the generator $g \in_R G$. In addition, define the hash functions $H_1: G \times \{0,1\}^* \rightarrow G, H_2: G_T \rightarrow \{0,1\}^\lambda$. The global parameters are set to $param = (G, G_T, g, H_1(\cdot), H_2(\cdot))$.

2) $KeyGen(param)$: To generate a public and secret key pair for user i , the key generation algorithm picks $x_i \in_R Z_p^*$, and sets $pk_i = g^{x_i}$ and $sk_i = x_i$.

3) $ReKeyGen(sk_i, pk_j, w)$: Starting from the delegator's secret key sk_i , the delegatee's public

key pk_j and a condition w , the re-encryption key generation algorithm randomly picks $\alpha \in_R Z_p^*$ and returns the re-encryption key $rk_{ij} = (d_0, d_1) =$

$$(g^{\frac{x_j}{x_i}} H_1(pk_i, w)^\alpha, g^{x_i \alpha}).$$

4) $Encrypt(pk_i, m, w)$: The encryption algorithm takes as its inputs a plaintext $m \in \{0, 1\}^\lambda$, a public key pk_i and a condition w . It then picks $r \in Z_p^*$ and outputs $ct_i = (c_1, c_2, c_3)$, where $c_1 = pk_i^r$, $c_2 = H_2(e(g, g)^r) \oplus m$ and $c_3 = H_1(pk_i, w)^r$.

5) $ReEncShare(rk_{ij}, ct_i)$: To generate the re-encryption ciphertext of an original ciphertext $ct_i = (c_1, c_2, c_3)$, the re-encryption algorithm picks $s \in_R Z_p^*$ and computes $ct'_j = (c'_1, c'_2)$, where

$$c'_1 = \frac{e(c_1, rk_{ij} d_0)}{e(rk_{ij} d_1, c_3)} = \frac{e(g^{x_j r}, g^{\frac{x_j}{x_i}} H_1(pk_i, w)^\alpha)}{e(g^{x_i \alpha}, H_1(pk_i, w)^r)} = e(g, g)^{x_j r}$$

6) $Decrypt(sk_i, ct_i)$: Starting from a secret key sk_i and an original ciphertext ct_i (or transformed ciphertext ct'_i), the decryption works according to two cases:

① If the input is an original ciphertext $ct_i = (c_1, c_2, c_3)$, compute $c_2 \oplus H_2(e(g, c_1)^{\frac{1}{sk_i}}) = H_2(e(g, g)^r) \oplus m \oplus H_2(e(g, g)^{\frac{x_j r}{x_i}}) = H_2(e(g, g)^r) \oplus m \oplus H_2(e(g, g)^r) = m$.

② If the input is a transformed ciphertext $ct'_i = (c'_1, c'_2)$, compute $c'_2 \oplus H_2((c'_1)^{\frac{1}{sk_i}}) = H_2(e(g, g)^r) \oplus m \oplus H_2(e(g, g)^{\frac{rx_j}{x_i}}) = m$.

The correctness of this construction can be verified by computing.

$$\begin{aligned} & Decrypt(sk_i, Encrypt(pk_i, m, w)) \\ &= H_2(e(g, g)^r) \oplus m \oplus H_2(e(g, g^{\frac{x_j r}{x_i}})^{\frac{1}{sk_i}}) \\ &= H_2(e(g, g)^r) \oplus m \oplus H_2(e(g, g)^r) \\ &= m \\ & Decrypt(sk_i, ReEncShare(rk_{ij}, ct_i)) \\ &= H_2(e(g, g)^r) \oplus m \oplus \end{aligned} \quad (4)$$

$$\begin{aligned} & H_2 \left(\left[\frac{e(g^{x_j r}, g^{\frac{x_j}{x_i}} H_1(pk_i, w)^\alpha)}{e(g^{x_i \alpha}, H_1(pk_i, w)^r)} \right]^{\frac{1}{x_j}} \right) \\ &= H_2 \left(e(g, g)^r \oplus m \oplus H_2(e(g, g)^{\frac{x_j r}{x_i}}) \right) \\ &= H_2(e(g, g)^r) \oplus m \oplus H_2(e(g, g)^r) \\ &= m \end{aligned} \quad (5)$$

5.2 Second construction

The second construction is based on the first one, but involves multiple access control servers for supporting threshold-based proxy re-encryption. The basic idea is to use Shamir's secret sharing^[7] on the re-encryption key rk_{ij} to generate n re-encryption key shares. Each re-encryption key share is used by an access control server for proxy re-encryption. At least t re-encryption shares are then collected and used for re-constructing the transformed ciphertext. We present the re-written $ReKeyGen(\cdot)$, $ReEncShare(\cdot)$ and the newly introduced $ShareCombine(\cdot)$ algorithms as follows.

1) $ReKeyGen(sk_i, pk_j, w, t, n)$: Such algorithm takes the following steps.

① For each index $v \in S'$ where $S' = \{1, 2, \dots, t-1\}$, pick $\alpha_v, \beta_v \in_R Z_p^*$ and set the v -th re-encryption key share to be $[rk_{ij}]_v = (d_0, d_1) = (g^{\alpha_v} H_1(pk_i, w)^{\beta_v}, g^{x_i \beta_v})$. This implicitly defines a polynomial

$$L(x) = A_{0, S'}(x) \cdot \frac{x_j}{x_i} + \sum_{v \in S'} A_{v, S'}(x) \alpha_v \quad \text{and} \quad L(0) = \frac{x_j}{x_i}.$$

② For each remaining index $v \in \{t, t+1, t+2, \dots, n\}$, pick a random variable $\beta_v \in_R Z_p^*$ and set the v -th re-encryption key share to be $[rk_{ij}]_v =$

$$(d_0, d_1) = (g^{L(v)} H_1(pk_i, w)^{\beta_v}, g^{x_i \beta_v}) = (g^{A_{0, S'}(v) \cdot \frac{x_j}{x_i} + \sum_{z=1}^{t-1} A_{z, S'}(v) \alpha_z} H_1(pk_i, w)^{\beta_v}, g^{x_i \beta_v}).$$

③ Finally return the set of re-encryption key shares $\{[rk_{ij}]_v\}_{v=1}^n$.

2) $ReEncShare([rk_{ij}]_v, ct_i)$: This algorithm picks a random variable $s \in_R Z_p^*$ and computes

$[\bar{c}_1]_v = ([rk_{ij}]_v, d_0)^s = g^{\frac{1}{s}} H_1(pk_i, w)^{\frac{\beta_v}{s}}, [\bar{c}_2]_v = [rk_{ij}]_v, d_1 = g^{x_i \beta_v}$ and $[\bar{c}_3]_v = c_1^s = g^{x_i r s}$. Finally it returns the re-encryption share $[\theta_{ij}]_v = ([\bar{c}_1]_v, [\bar{c}_2]_v, [\bar{c}_3]_v)$.

3) *ShareCombine*($ct_i, \{[\theta_{ij}]_v\}_{v \in S}$): Given the original ciphertext $ct = (c_1, c_2, c_3)$ and the set of re-encryption shares $\{[\theta_{ij}]_v\}_{v \in S}$ with $|S| \geq t$ (for convenience, we just assume $|S| = t$). This algorithm computes

$$\begin{aligned} c'_1 &= \prod_{v \in S} \left(\frac{e([\bar{c}_1]_v, [\bar{c}_3]_v)}{e(c_3, [\bar{c}_2]_v)^{A_{v,S}(0)}} \right) \\ &= \prod_{v \in S} \left(\frac{e(g^{\frac{1}{s}} H_1(pk_i, w)^{\frac{\beta_v}{s}}, g^{x_i r s})}{e(H_1(pk_i, w)^r, g^{x_i \beta_v})} \right)^{A_{v,S}(0)} \\ &= \prod_{v \in S} e(g^{\frac{1}{s}}, g^{x_i r s})^{A_{v,S}(0)} \\ &= e(g, g)^{x_i r \sum_{v \in S} A_{v,S}(0) L(v)} \\ &= e(g, g)^{x_i r \frac{x_j}{x_i}} = e(g, g)^{x_j r} \end{aligned}$$

It finally returns the transformed ciphertext $ct'_j = (c'_1, c_2)$.

The correctness of this construction could be easily verified through equation (4) and equation (5). Thus it is omitted here.

5.3 Final construction

To achieve the goal of SS-CCA, we need to further embed the "authentication code" in ciphertext. This is because most (or maybe all) CCA attacks rely on the attackers' modifying the challenge ciphertext and then feeding it to the decryption oracle. The answer of the decryption oracle will enable the attacker to tell which message is encrypted. If an authentication code is put on top of the encryption, disallowing the attacker to fiddle with the ciphertext, then it essentially renders the decryption oracle useless. Following this intuition, we present the final construction for achieving CCA security (the *KeyGen*($\cdot$) is the same as in previous constructions, so it is omitted here).

1) *Setup*(k): The setup algorithm is identical

to the same one presented in the first construction (see section 5.1) except that the hash functions in the global parameters *param* are initialized as: $H_1 : G \times \{0, 1\}^* \rightarrow G, H_2 : G_T \rightarrow \{0, 1\}^\lambda \times \{0, 1\}^{\lambda'}$, $H_3 : \{0, 1\}^\lambda \times \{0, 1\}^{\lambda'} \rightarrow Z_p^*$ and $H_4 : \{0, 1\}^{\lambda+\lambda'} \times G \rightarrow G$.

2) *ReKeyGen*(sk_i, pk_j, w, t, n): Unlike the algorithm with the same name, presented in the first two constructions, the re-encryption key generation algorithm returns a set of re-encryption key shares $\{[rk_{ij}]_v\}_{v=1}^n$ and verification key shares $\{[vk_{ij}]_v\}_{v=1}^n$. In detail, it performs the following steps.

① For each index $v \in S'$ where $S' = \{1, 2, \dots, t-1\}$, pick $\alpha_v, \beta_v \in_R Z_p^*$ and set the v -th re-encryption key share to be $[rk_{ij}]_v = (d_0, d_1) = (g^{\alpha_v} H_1(pk_i, w)^{\beta_v}, g^{x_i \beta_v})$. As in the previous construction, this way implicitly defines the polynomial $L(x) = \Delta_{0,S'}(x)$.

$\frac{x_j}{x_i} + \sum_{v \in S'} \Delta_{v,S'}(x) \alpha_v$ and $L(0) = \frac{x_j}{x_i}$. Moreover, compute the verification key share $[vk_{ij}]_v = (l_0, l_1) = (pk_i^{\alpha_v}, pk_i^{\beta_v}) = (g^{x_i \alpha_v}, g^{x_i \beta_v})$.

② For each remaining index $v \in \{t, t+1, t+2, \dots, n\}$, pick a random variable $\beta_v \in_R Z_p^*$ and set the v -th re-encryption key share to be $[rk_{ij}]_v = (d_0, d_1) = (g^{L(v)} H_1(pk_i, w)^{\beta_v}, g^{x_i \beta_v})$. The v -th verification key share could be computed as $[vk_{ij}]_v = (l_0, l_1) = (pk_i^{L(v)}, pk_i^{\beta_v}) = (g^{x_i \Delta_{0,S'}(v) \frac{x_j}{x_i} + \sum_{z=1}^{t-1} \Delta_{z,S'}(v) \alpha_z}, g^{x_i \beta_v})$.

③ Finally output the set of re-encryption key shares $\{[rk_{ij}]_v\}_{v=1}^n$ and verification key shares $\{[vk_{ij}]_v\}_{v=1}^n$. Note that the validity of each re-encryption key share $[rk_{ij}]_v$ could be checked by verifying the equation $e([rk_{ij}]_v, d_0, pk_i) \stackrel{?}{=} e(H_1(pk_i, w), [rk_{ij}]_v, d_1) e(g, [vk_{ij}]_v, l_0)$.

3) *Encrypt*(pk_i, m, w): The encryption algorithm takes input a plaintext $m \in \{0, 1\}^\lambda$, a public key pk_i and a condition w . It picks $r' \in_R \{0, 1\}^{\lambda'}$ and computes $r = H_3(m, r')$. Finally returns $ct_i = (c_1, c_2, c_3, c_4)$, where $c_1 = pk_i^r, c_2 = H_2(e(g, g)^r) \oplus$

$(m \parallel r')$, $c_3 = H_1(pk_i, w)^r$ and $c_4 = H_4(c_2, c_3)^r$.

4) *ReEncShare* $([rk_{ij}]_v, ct_i)$: The share re-encryption algorithm takes as its inputs the re-encryption key share $[rk_{ij}]_v$ and original ciphertext $ct_i = (c_1, c_2, c_3, c_4)$. The following steps are performed.

① Check the validity of the original ciphertext ct_i by verifying the equations: $e(c_1, H_3(pk_i, w)) \stackrel{?}{=} e(pk_i, c_3)$ and $e(c_1, H_4(c_2, c_3)) \stackrel{?}{=} e(pk_i, c_4)$. We note that if ct_i is a valid original ciphertext, the two equations above hold, since $e(c_1, H_3(pk_i, w)) = e(pk_i^r, H_3(pk_i, w)) = e(pk_i, H_3(pk_i, w)^r) = e(pk_i, c_3)$, $e(c_1, H_4(c_2, c_3)) = e(pk_i^r, H_4(c_2, c_3)) = e(pk_i, H_4(c_2, c_3)^r) = e(pk_i, c_4)$.

② Pick $s \in_R Z_p^*$ and compute $[\bar{c}_1]_v = pk_i^s = g^{x_i s}$, $[\bar{c}_2]_v = ([rk_{ij}]_v, d_0)^s = g^{\frac{L(v)}{s}} H_1(pk_i, w)^{\frac{\beta_v}{s}}$ and $[\bar{c}_3]_v = c_1^s = g^{x_i r s}$.

③ Finally output the v -th re-encryption share $[\theta_{ij}]_v = ([\bar{c}_1]_v, [\bar{c}_2]_v, [\bar{c}_3]_v)$.

5) *ShareVerify* $(ct_i, [vk_{ij}]_v, [\theta_{ij}]_v)$: The re-encryption share verification algorithm takes as its inputs the original ciphertext $ct_i = (c_1, c_2, c_3, c_4)$, the verification key share $[vk_{ij}]_v$ and the re-encryption share $[\theta_{ij}]_v = ([\bar{c}_1]_v, [\bar{c}_2]_v, [\bar{c}_3]_v)$. The algorithm checks whether the following equations hold: $e([\bar{c}_1]_v, c_1) \stackrel{?}{=} e(pk_i, [\bar{c}_3]_v)$ and $e([\bar{c}_2]_v, [\bar{c}_1]_v) \stackrel{?}{=} e(g, [vk_{ij}]_v, l_0) e(H_1(pk_i, w), [vk_{ij}]_v, l_1)$. If both the conditions are verified, it returns 1, otherwise it returns 0. Note that a well-formed re-encryption share generated by the algorithm *ReEncShare* $(\cdot)$ can pass the above two equations (i.e., the construction satisfies the verifiability on encryption share in equation (2), since $e([\bar{c}_1]_v, c_1) = e(pk_i^s, pk_i^r) = e(pk_i, pk_i^{rs}) = e(pk_i, [\bar{c}_3]_v)$ and $e([\bar{c}_2]_v, [\bar{c}_1]_v) = e(g^{\frac{L(v)}{s}} H_1(pk_i, w)^{\frac{\beta_v}{s}}, pk_i^s) = e(g, pk_i^{L(v)}) e(H_1(pk_i, w), pk_i^{\beta_v}) = e(g, [vk_{ij}]_v, l_0) e(H_1(pk_i, w), [vk_{ij}]_v, l_1)$.

6) *ShareCombine* $(ct_i, \{[\theta_{ij}]_v\}_{v \in S})$: Like the corresponding algorithm presented in the second con-

struction (see Section 5.2), the share combination algorithm computes $c'_1 = \prod_{v \in S} \left(\frac{e([\bar{c}_1]_v, [\bar{c}_3]_v)}{e(c_3, [\bar{c}_2]_v)} \right)^{4_{v,s}(0)} = e(pk_j, g)^r$. Finally it returns the transformed ciphertext $ct = (c'_1, c_2)$.

7) *Decrypt* (sk_i, ct_i) : The decryption algorithm works as in the second construction (section 5.2). One exception is that the validity of the ciphertext needs to be checked. More specifically,

① If the input is an original ciphertext: check whether the equations $e(c_1, H_3(pk_i, w)) \stackrel{?}{=} e(pk_i, c_3)$ and $e(c_1, H_4(c_2, c_3)) \stackrel{?}{=} e(pk_i, c_4)$ hold (as in the validity checking in *ReEncShare* $(\cdot)$ algorithm). If both the conditions are positively verified compute $m \parallel r'$ following the same strategy of *Decrypt* $(\cdot)$, in the second construction. Further, check whether $c_4 \stackrel{?}{=} H_4(c_2, c_3)^{H_1(m \parallel r')}$ holds. Finally return m .

② If the input is a transformed ciphertext: compute $m \parallel r'$ in the same way of *Decrypt* $(\cdot)$ in the second construction. Then check whether $c'_1 = e(g^{sk_i}, g)^{H_1(m \parallel r')}$ holds. Finally return m .

The consistency of original and transformed ciphertext could be easily checked in a similar way by using equation (4) and equation (5).

6 Security analysis

In this section we analyze the security of the SAAC solution proposed in section 4.2. In what follows, we at first model the capabilities of attackers against the SAAC system. Then, we formalize two security definitions, original ciphertext security (IND-oCCA) and transformed ciphertext security (IND-tCCA), associated to TCPRE. Finally we prove that the final TCPRE scheme presented in section 5.3 achieves both the security definitions.

6.1 Threat capabilities

We simplify the data flow of the proposed SAAC system as sketched in Figure 2. Notice that we omit the encrypted file content because it is protected by symmetric-key encryption, which is as-

sumed to be secure if the symmetric encryption key is not leaked.

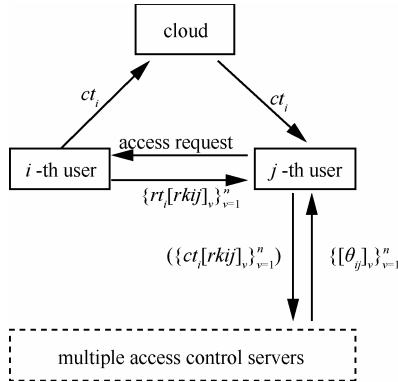


Figure 2 Data flow in server-aided access control system

Initially, the i -th user uploads the ciphertext ct_i to cloud. Later, the j -th user downloads the ciphertext ct_i and sends an access request to the owner (i.e., the i -th user). The file owner responds with a set of re-encryption keys $\{[rk_{ij}]_v\}_{v=1}^n$ that would be forwarded along with the ciphertext ct_i to the access control servers for re-encryption. After re-encryption by each access control server, the re-encryption shares are sent back to the j -th user for combination and decryption.

We point out that these data including the original ciphertext ct_i , the re-encryption key share $[rk_{ij}]_v$ and the re-encryption share $[\theta_{ij}]_v$, are transferred by using public channels and hence can be eavesdropped by attackers. For analyzing semantic security, we model these leakages (as threat capabilities against our system) in the following oracles. Notice that since the original ciphertext ct_i can be easily computed by using the user's public key, we omit modeling it as an oracle.

1) Re-encryption key share oracle $O_{[rk]}(\cdot)$. This oracle models the attackers' capabilities of eavesdropping the key shares on the channels from users to access control servers. Starting from the inputs (pk_i, pk_j, w, t, n, U) with $1 \leq t \leq n$ and $U \subseteq \{1, 2, \dots, n\}$ this oracle runs $ReKeyGen(sk_i, pk_j, w, t, n) = \{([rk_{ij}]_v, [vk_{ij}]_v)\}_{v=1}^n$ and returns $\{[rk_{ij}]_v\}_{v \in U}$ and $\{[vk_{ij}]_v\}_{v=1}^n$. Notice that the set U here models

the set of channels from users to the access control servers eavesdropped by attackers.

2) Re-encryption share oracle $O_{[\theta]}(\cdot)$. This oracle models the attackers' capabilities of eavesdropping the shares on the channels from access control servers to the users. Starting from the inputs $(pk_i, pk_j, w, t, n, ct_i, V)$ with $t \leq n$ and $V \subseteq \{1, 2, \dots, n\}$, this oracle runs $ReEncShare([rk_{ij}]_v, ct_i) = [\theta_{ij}]_v$ and returns $\{[\theta_{ij}]_v\}_{v \in V}$. Also in this case, the set V models the set of channels from access control servers to users eavesdropped by attackers.

We further strengthen threat capabilities by considering corrupted users and CCA model and define three oracles below.

1) Uncorrupted key generation oracle $O_u(\cdot)$.

This oracle models the common users which can publish their public keys. With each user i , this oracle runs the $KeyGen(\cdot)$ algorithm and obtains a public and secret key pair (pk_i, sk_i) . Finally it returns the public key pk_i .

2) Corrupted key generation oracle $O_c(\cdot)$.

This oracle models the corrupted users which have been intruded by attackers. For each user i , this oracle runs the $KeyGen(\cdot)$ algorithm and obtains a public and secret key pair (pk_i, sk_i) . Finally return the (pk_i, sk_i) .

3) Decryption oracle $O_m(\cdot)$.

This oracle models the capability of chosen ciphertext attack. On receiving the pair (pk_i, ct_i) as its input, this oracle runs $Decrypt(sk_i, ct_i)$ and returns the underlying message m . Notice that sk_i is the secret key with respect to the input public key pk_i .

6.2 Security definitions of TCPRE

Based on the threat capabilities formalized in the previous subsection, we define the semantic security of TCPRE, which captures the spirit of indistinguishability of both original and transformed ciphertext. Formally, we respectively formalize the security notions as IND-oCCA (for the security of original ciphertext) and IND-tCCA (for the security of transformed ciphertext).

The above security definitions are verified by using specific experiments involving an adversary $\mathcal{A}$. The adversary runs in find and guess stages. We give the experiment $\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-oCCA}-b}$ for IND-oCCA as follows.

Experiment $\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-oCCA}-b}(k)$
 $param = \text{Setup}(k)$
 (m_0, m_1, pk_i^*, w^*)
 $= A^{O_u(\cdot), O_c(\cdot), O_{[rk]}(\cdot), O_{[\theta]}(\cdot), O_m(\cdot)}(\text{find}, param),$
 where $|m_0| = |m_1|$
 $ct^* = \text{Encrypt}(pk_i^*, m_b, w^*)$
 $b' = A^{O_u(\cdot), O_c(\cdot), O_{[rk]}(\cdot), O_{[\theta]}(\cdot), O_m(\cdot)}(\text{guess}, param, ct^*)$
 return b' .

In the IND-oCCA experiment, the $\text{Setup}(\cdot)$ algorithm is initially executed to generate the global parameters $param$. Next, $\mathcal{A}$ executes the find stage. Specifically, it takes as its input the parameters $param$, accesses several oracles and returns the challenge messages m_0, m_1 as well as pk_i and w^* . For these oracles, we have two requirements.

1) For a public key pk_j generated from $\mathcal{O}_u$, one of them would be selected as pk_i^* for generating output. Further, if $\mathcal{A}$ has issued the queries $O_{[rk]}(pk_i^*, pk_j, w^*, t, n, U)$ and $O_{[\theta]}(pk_i^*, pk_j, w^*, t, n, ct^*, L)$ with $|U \cup L| \geq t$, then $\mathcal{A}$ cannot issue the query $O_m(pk_j, ct_j)$ such that $\text{Decrypt}(sk_j, ct_j) \in \{m_0, m_1\}$, where n is the number of proxies and t is the threshold.

2) For a public key pk_j generated from $\mathcal{O}_c$, $\mathcal{A}$ cannot issue these queries, $O_{[rk]}(pk_i^*, pk_j, w^*, t, n, U)$ with $|U| \geq t$, or, $O_{[rk]}(pk_i^*, pk_j, w^*, t, n, U)$ and $O_{[\theta]}(pk_i^*, pk_j, w^*, t, n, ct^*, L)$ with $|U \cup L| \geq t$.

We define $\mathcal{A}$'s advantage in the IND-oCCA experiment $\text{Adv}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-oCCA}}(k)$ as $|\Pr[\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-oCCA}-1}(k) = 1] - \Pr[\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-oCCA}-0}(k) = 1]|$ and formalize the IND-oCCA security as follows.

Definition 2 IND-oCCA Security

A TCPRE scheme is said to be $(t, q_u, q_c, q_{[rk]}, q_{[\theta]}, q_m, \epsilon)$ -secure, if for any t -time adversary

$\mathcal{A}$ who asks at most $q_u, q_c, q_{[rk]}, q_{[\theta]}$ and q_m queries to oracles $O_u(\cdot), O_c(\cdot), O_{[rk]}(\cdot), O_{[\theta]}(\cdot)$ and $O_m(\cdot)$ respectively, we have $\text{Adv}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-oCCA}}(k) \leq \epsilon$.

Similarly, we give the experiment $\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-tCCA}-b}$ for IND-tCCA as follows.

Experiment $\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-tCCA}-b}(k)$
 $param = \text{Setup}(k)$
 $(m_0, m_1, pk_i, pk_j^*, w^*)$
 $= \mathcal{A}^{O_u(\cdot), O_c(\cdot), O_{[rk]}(\cdot), O_{[\theta]}(\cdot), O_m(\cdot)}(\text{find}, param),$
 where $|m_0| = |m_1|$
 $ct = \text{Encrypt}(pk_i, m_b, w^*)$
 $ct^* = \text{ShareCombine}(ct,$
 $\{ \text{ReEncShare}([rk_{ij}]_v, ct) \}_{v=1}^t, \{ [rk_{ij}]_v \}_{v=1}^t)$
 $b' = A^{O_u(\cdot), O_c(\cdot), O_{[rk]}(\cdot), O_m(\cdot)}(\text{guess}, param, ct^*)$
 return b' .

As with $\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-oCCA}}(k)$, $\mathcal{A}$ takes as its input the global parameters and returns the challenge messages m_0, m_1 as well as pk_i, w^* in the find stage. But unlike $\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-oCCA}}(k)$, in this experiment, we require $\mathcal{A}$ to additionally output pk_j^* for re-encrypting the challenge messages and disallow $\mathcal{A}$ to access the re-encryption share oracle $O_{[\theta]}(\cdot)$. The challenge messages are then encrypted (with pk_i) and re-encrypted into the ciphertext under pk_j^* . Finally, in the guess stage, $\mathcal{A}$ outputs the bit b' which is the guess of b .

We define $\mathcal{A}$'s advantage in the IND-tCCA experiment $\text{Adv}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-tCCA}}(k)$ as $|\Pr[\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-tCCA}-1}(k) = 1] - \Pr[\text{Exp}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-tCCA}-0}(k) = 1]|$ and formalize the IND-tCCA security as follows.

Definition 3 IND-tCCA Security

A TCPRE scheme is said to be $(t, q_u, q_c, q_{[rk]}, q_m, \epsilon)$ -secure, if for any t -time adversary $\mathcal{A}$ who asks at most $q_u, q_c, q_{[rk]}$ and q_m queries to oracles $O_u(\cdot), O_c(\cdot), O_{[rk]}(\cdot)$ and $O_m(\cdot)$ respectively, we have $\text{Adv}_{\text{TCPRE}, \mathcal{A}}^{\text{IND-tCCA}}(k) \leq \epsilon$.

6.3 Security proof of TCPRE

We now turn to analyze the security of the final scheme presented in section 5.3 against Definition 2 and Definition 3. We firstly introduce a new

computational problem, and show that it is equivalent to the weak Decisional Bilinear Diffie-Hellman Inversion (wDBDHI) problem investigated in [8]. Then, we prove that the final scheme is secure under the assumption that the newly introduced problem is intractable.

We describe both assumptions as follows.

Definition 4 q -wDBDHI Problem

The q -wDBDHI problem in groups (G, G_T) consists in: given $(g, g^a, \dots, g^{a^q}, g^b, Z) \in G^{q+2} \times G_T$ with unknown $a, b \in_R Z_p^*$, deciding whether $Z = e(g, g)^{\frac{b}{a}}$.

Definition 5 New Problem

Given the tuple $(g, g^{\frac{1}{a}}, g^a, \dots, g^{a^{q-1}}, g^b, Z) \in G^{q+2} \times G_T$ as input, decide whether Z equals to $e(g, g)^{\frac{b}{a^2}}$ or to a random value.

We present the following theorem to demonstrate the equivalence of the problem defined in Definition 5 and the q -wDBDHI problem.

Theorem 1 The problem defined in Definition 5 is equivalent to the wDBDHI problem.

Proof: Suppose there exists a q -wDBDHI problem solver S which can solve the q -wDBDHI problem in polynomial time. We then describe how to construct an algorithm leveraging S to solve the new problem introduced in Definition 5.

Given the tuple $(g, g^{\frac{1}{a}}, g^a, \dots, g^{a^{q-1}}, g^b, Z) \in G^{q+2} \times G_T$, we set $y = g^{\frac{1}{a}}$, $y^A = g, \dots, y^{A^q} = g^{a^{q-1}}$ and $y^B = g^b$ which implicitly defines $A = a$ and $B = ab$. Then we run S with the input $(y, y^A, \dots, y^{A^q}, y^B, Z)$. Notice that $e(y, y)^{\frac{B}{A}} = e(g^{\frac{1}{a}}, g^{\frac{1}{a}})^{\frac{ab}{a}} = e(g, g)^{\frac{b}{a^2}}$, if S can distinguish $e(g, g)^{\frac{B}{A}}$ from any random element in G_T (according to q -wDBDHI assumption), it is also able to distinguish $e(g, g)^{\frac{b}{a^2}}$ from a random element, solving the new problem introduced in Definition 5.

For the inverse implication, we suppose the

existence of a solver S for the problem formalized in Definition 5. We then describe how to use S to solve the q -wDBDHI problem.

Given the q -wDBDHI tuple $(g, g^a, \dots, g^{a^q}, g^b, Z) \in G^{q+2} \times G_T$, we set $y = g^a, y^{\frac{1}{A}} = g, \dots, y^{A^{q-1}} = g^{a^q}, y^B = g^b$ which implicitly defines $A = a$ and $B = \frac{b}{a}$. We run the problem solver S with input $(y, y^{\frac{1}{A}}, y^A, \dots, y^{A^{q-1}}, y^B, Z)$. Since $e(g, g)^{\frac{b}{a^2}} = e(g^a, g^a)^{\frac{1}{a} \frac{ab}{a}} = e(y, y)^{\frac{B}{A}}$, if S can distinguish $e(g, g)^{\frac{b}{a^2}}$ from any random element in G_T , it is also able to distinguish $e(g, g)^{\frac{B}{A}}$ from a random element, solving the q -wDBDHI problem.

From Theorem 1, we can see that the introduced new problem is equivalent to the wDBDHI problem, for which the best known algorithm is to solve the standard Discrete Logarithm Problem (DLP) [9]. Thus we can present the following theorems (based on the intractability of the introduced problem) to demonstrate the IND-oCCA security of the final scheme in section 4.3.

Theorem 2 The proposed final scheme in section 5.3 is IND-oCCA secure in the random oracle model, assuming our introduced problem is intractable in groups (G, G_T) .

Proof: Suppose that the algorithm B receives as input a challenge tuple $(g, g^{\frac{1}{a}}, g^a, g^{a^2}, g^{a^3}, g^b, Z) \in G^6 \times G$ with unknown $a, b \in_R Z_p$. B 's goal is to decide whether $Z = e(g, g)^{\frac{b}{a^2}}$. For this purpose, algorithm B sets the parameters and plays the IND-oCCA experiment with adversary A in the following way.

In the IND-oCCA experiment, B gives the global parameters $param = (G, G_T, g, \{H_i(\cdot)\}_{i=1}^4)$ to A . Here $\{H_i(\cdot)\}_{i=1}^4$ are the random oracles controlled by B and can be adaptively asked by A at any time. B maintains 4 hash lists L_i for $i = 1, 2, 3, 4$, which are initially empty. B responds

to the random oracle queries from A as follows.

1) $O_{H_1}(\cdot)$: By receiving an input (pk_i, w) , if this query already appears on the L_1 , then return the predefined value. Otherwise choose, and use the Coron's proof technique^[10] to flip a biased coin, that yields 1 with probability and with probability. If define; otherwise define. Finally add tuple to and respond with.

2) $O_{H_2}(\cdot)$: By receiving an input $R \in G_T$, if this query already appears on the L_2 , then return the predefined value. Otherwise choose $\sigma \in_R \{0, 1\}^{\lambda+\lambda'}$, add tuple (R, σ) to L_2 and respond with $H(R) = \sigma$.

3) $O_{H_3}(\cdot)$: By receiving an input (m, r') , if this tuple (m, r', r) already appears on the L_3 , then return r . Otherwise choose $r \in_R Z_p$, add tuple (m, r', r) to L_3 and respond with $H_3(m, r') = r$.

4) $O_{H_4}(\cdot)$: By receiving an input (c_2, c_3) , if this query already appears on the L_4 , then return the predefined value. Otherwise choose $\gamma \in_R Z_p$, add tuple (c_2, c_3, γ) to the L_4 and respond with $H_4(c_2, c_3) = g^\gamma$.

When operating in its find stage, A issues a series of queries as in the IND-oCCA experiment. B maintains lists L_{sk}, L_{rk} and L_{vk} , which are initially empty. B answers these queries issued by A as follows.

1) $O_u(\cdot)$: When receiving the index of the user i as its input, B picks $x_i \in_R Z_p$ and defines $pk_i = (g^{a^2})^{x_i}$. Next, it sets $tag_i = 0$ and adds tuple (pk_i, x_i, tag_i) to L_{sk} . Finally it returns pk_i to A .

2) $O_c(\cdot)$: When receiving the index of the user j as its input, B picks $x_j \in_R Z_p$ and defines $pk_j = g^{x_j}, sk_j = x_j$ and $tag_j = 1$. Next it adds the tuple (pk_j, x_j, tag_j) to L_{sk} and returns (pk_j, sk_j) to A .

3) $O_{[rk]}$: Starting from the input (pk_i, pk_j, w, t, n, U) , B recovers the tuple $(pk_i, w, coin_{i,w}, \mu)$ from L_1 , and the tuples (pk_i, x_i, tag_i) and (pk_j, x_j, tag_j) from L_{sk} . Next it reconstructs

the re-encryption key shares $\{[rk_{ij}]_v\}_{v \in U}$ and the verification key shares $\{[vk_{ij}]_v\}_{v=1}^n$ for adversary A according to the following cases.

Case 1: $tag_i = 1$, it means that $sk_i = x_i$. In other words, the user with index i has been corrupted. Using sk_i , algorithm B can certainly run $ReKeyGen(sk_i, pk_j, w, t, n)$ to obtain n shares of re-encryption keys $\{[rk_{ij}]_v\}_{v=1}^n$ and verification keys $\{[vk_{ij}]_v\}_{v=1}^n$. Then it stores $(pk_i, pk_j, w, t, n, \{[rk_{ij}]_v\}_{v=1}^n)$ in L_{rk} and $(pk_i, pk_j, w, t, n, \{[vk_{ij}]_v\}_{v=1}^n)$ in L_{vk} . Finally it returns $\{[rk_{ij}]_v\}_{v \in U}$ and $\{[vk_{ij}]_v\}_{v=1}^n$ to A .

Case 2: $tag_i = 0 \wedge tag_j = 0$, it means that $sk_i = a^2 x_i$ and $sk_j = a^2 x_j$. In other words, both the users with index i and j are not corrupted. Algorithm B can also correctly generate n shares of re-encryption keys $\{[rk_{ij}]_v\}_{v=1}^n$ and verification keys $\{[vk_{ij}]_v\}_{v=1}^n$ by simulating the results of $ReKeyGen(\cdot)$, since it can derive $pk_j^{\frac{1}{sk_j}}$ as $g^{\frac{x_j}{x_i}}$. Then it stores $(pk_i, pk_j, w, t, n, \{[rk_{ij}]_v\}_{v=1}^n)$ in L_{rk} and $(pk_i, pk_j, w, t, n, \{[vk_{ij}]_v\}_{v=1}^n)$ in L_{vk} . Finally it returns $\{[rk_{ij}]_v\}_{v \in U}$ and $\{[rk_{ij}]_v\}_{v=1}^n$ to A .

Case 3: $tag_i = 0 \wedge tag_j = 1 \wedge coin_{i,w} = 0$, it means that $sk_i = a^2 x_i, sk_j = x_j$ and $H_1(pk_i, w) = g^{\frac{\mu}{a^2}}$.

In other words, the user with index i is not corrupted, while the one with index j has been corrupted. Moreover, the flip coin $coin_{i,w}$ in oracle $O_{H_1}(\cdot)$ related to w is 0. Algorithm B can correctly generate $t-1$ shares of re-encryption key shares $\{[rk_{ij}]_v\}_{v=1}^{t-1}$ and verification key shares $\{[vk_{ij}]_v\}_{v=1}^{t-1}$. B can further simulate the rest $n-t+1$ shares $\{[rk_{ij}]_v\}_{v=t}^n$ and $\{[vk_{ij}]_v\}_{v=t}^n$ as in $ReKeyGen(\cdot)$. Then it stores the tuple $(pk_i, pk_j, w, t, n, \{[rk_{ij}]_v\}_{v=1}^n)$ in L_{rk} and $(pk_i, pk_j, w, t, n, \{[vk_{ij}]_v\}_{v=1}^n)$ in L_{vk} . Finally it returns $\{[rk_{ij}]_v\}_{v \in U}$ and $\{[vk_{ij}]_v\}_{v=1}^n$ to A .

Case 4: $tag_i = 0 \wedge tag_j = 1 \wedge coin_{i,w} = 1$, it

means that $sk_i = a^2 x_i, sk_j = x_j$ and $H_1(pk_i, w) = g^{a^2 \mu}$.

In other words, the user with index i is not corrupted, while the one with index j has been corrupted. Moreover, the flip coin $coin_{i,w}$ in oracle $O_{H_1}(\cdot)$ related to w is 1. If $|U| \geq t$, the algorithm B generates a random bit $\beta' \in_R \{0,1\}$ and aborts.

Otherwise, B chooses a set U' such that $U \subseteq U'$ and $|U'| = t-1$. For each index $v \in U'$, it picks $\alpha_v, \beta_v \in_R Z_p$, and sets the v -th re-encryption key share and verification key share to be $[rk_{ij}]_v = (g^{\alpha_v} H_1(pk_i, w)^{\beta_v}, pk_i^{\beta_v})$ and $[vk_{ij}]_v = (pk_i^{\alpha_v}, pk_j^{\beta_v})$. Next B picks $\beta \in_R Z_p$ and sets $S' = U' \cup \{0\}$. For each remaining index $v \in \{1, 2, \dots, n\}$, set the v -th verification key share to

be $[vk_{ij}]_v \cdot I_0 = pk_j^{\Delta_{0,S'}(v)} \prod_{z \in U'} [vk_{ij}]_z \cdot I_0^{\Delta_{z,S'}(v)}$ and $[vk_{ij}]_v \cdot I_1 = (pk_i^{\beta})^{\Delta_{0,S'}(v)} \prod_{z \in U'} [vk_{ij}]_z \cdot I_1^{\Delta_{z,S'}(v)}$. Then it stores $(pk_i, pk_j, w, t, n, \{[rk_{ij}]_v\}_{v \in U'})$ in L_{rk} and $(pk_i, pk_j, w, t, n, \{[vk_{ij}]_v\}_{v=1}^n, U', \beta, \{\alpha_v, \beta_v\}_{v \in U'})$ in L_{vk} . Finally it returns $\{[rk_{ij}]_v\}_{v \in U}$ and $\{[vk_{ij}]_v\}_{v=1}^n$ to A.

4) $O_{[0]}(\cdot)$: With an input $(pk_i, pk_j, w, t, n, ct_i, V)$, the algorithm B parses ct_i as $ct_i = (c_1, c_2, c_3, c_4)$. It recover the tuples (pk_i, x_i, tag_i) and (pk_j, x_j, tag_j) from L_{sk} and $(pk_i, w, coin_{i,w}, \mu)$ from L_1 . If $tag_i = 1$ or $tag_i = tag_j = 0$ or $tag_i = 0 \wedge tag_j = 1 \wedge coin_{i,w} = 0$, algorithm B first generates all the re-encryption key shares as in the response to the re-encryption key share query, and then uses these re-encryption key shares to re-encrypt ciphertexts for adversary A. Otherwise, (i.e., $tag_i = 0 \wedge tag_j = 1 \wedge coin_{i,w} = 1$), it means that $sk_i = a^2 x_i, sk_j = x_j$ and $H_1(pk_i, w) = g^{a^2 \mu}$, and algorithm B works as follows.

① If there does not exist the tuple $(pk_i, pk_j, w, t, n, \{[rk_{ij}]_v\}_{v \in U'})$ in L_{rk} or $(pk_i, pk_j, w, t, n, \{[vk_{ij}]_v\}_{v=1}^n, U', \beta, \{\alpha_v, \beta_v\}_{v \in U'})$ in L_{vk} , algorithm B first generates these re-encryption key shares and verification key shares.

② Search whether there exists a tuple $(m, r', r) \in L_3$ such that $pk_i^r = c_1, H_2(e(g, g)^r) \oplus (m \parallel r') = c_2, H_1(pk_i, w)^r = c_3, H_4(c_2, c_3)^r = c_4$. If it does not, B generates a random bit $\beta' \in_R \{0,1\}$ and aborts.

③ For each $v \in V \setminus U'$, pick $s_v \in_R Z_p$, and set $[\theta_{ij}]_v = ([\bar{c}_1]_v, [\bar{c}_2]_v, [\bar{c}_3]_v)$ with $[\bar{c}_1]_v = pk_i^{s_v}, [\bar{c}_2]_v = \frac{1}{[rk_{ij}]_v^{s_v}}$ and $[\bar{c}_3]_v = c_1^{s_v}$.

④ For each $v \in \frac{V}{U'}$, set $S' = U' \cup \{0\}$, pick $s'_v \in_R Z_p$ and let $[\theta_{ij}]_v = ([\bar{c}_1]_v, [\bar{c}_2]_v, [\bar{c}_3]_v)$ with $[\bar{c}_1]_v = (g^a)^{x_i s'_v}, [\bar{c}_2]_v = (g^a)^{x_j s'_v}$ and $[\bar{c}_3]_v = ((g^a)^{\Delta_{0,S'}(v)} (g^a)^{\sum_{z \in U'} \Delta_{z,S'}(v)} (g^{a^3})^{tmp})^{\frac{1}{s'_v}}$ where $tmp = \mu \beta \Delta_{0,S'}(v) + \sum_{z \in U'} \mu \beta_z \Delta_{z,S'}(0)$.

⑤ Finally, return $\{[\theta_{ij}]_v\}_{v \in V}$ to A.

5) $O_m(\cdot)$: With an input (pk_i, ct_i) , the algorithm B first recovers the tuple (pk_i, x_i, tag_i) from the L_{sk} . If $tag_i = 1$ (meaning $sk_i = x_i$), B decrypts the ciphertext by using sk_i and returns the plaintext to A. Otherwise, it proceeds according to the following two cases.

① ct_i is an original ciphertext $ct_i = (c_1, c_2, c_3, c_4)$: search the list L_3 to see whether there exists a tuple (m, r') such that $pk_i^r = c_1, H_2(e(g, g)^r) \oplus (m \parallel r') = c_2, H_1(pk_i, w)^r = c_3$ and $H_4(c_2, c_3)^r = c_4$. If yes, return m to A; otherwise B returns $\perp$.

② ct_i is a transformed ciphertext $ct_i = (c'_1, c'_2)$: search L_3 to see whether there exists a tuple (m, r', r) in L_3 such that $e(pk_i, g)^r = c'_1, H_2(e(g, g)^r) \oplus (m \parallel r') = c'_2$. If yes return m to A; otherwise return $\perp$.

When A decides that the initial stage is over, it emits a target public key pk_r^* , a condition w^* and two equal-length messages $m_0, m_1 \in \{0,1\}^\lambda$. Algorithm B responds as follows. ① Recover tuple (pk_i^*, x_i^*, c_i^*) from the L_{sk} and tuple $(pk_i^*, w^*, coin_{i^*,w^*}, \mu)$ from L_1 , where i^* corresponds to the index i of pk_i^* . If $coin_{i^*,w^*} = 0$, generate a random bit $\beta' \in_R \{0,1\}$ and abort. Otherwise, it means

that $H_1(pk_i^*, w^*) = g^{a^2 \mu}$. ② Flip a random coin $\delta \in_R \{0, 1\}$ and pick $r^* \in_R \{0, 1\}^{\lambda'}$. Define $c_1^* = (g^b)^{x_i^*}$, $c_2^* = H_2(Z) \oplus (m_\delta \| r^*)$ and $c_3^* = (g^b)^\mu$. ③ Issue a query of (c_2^*, c_3^*) to oracle $O_{H_4}(\cdot)$ to obtain the tuple (c_2^*, c_3^*, γ^*) . Define $c_4^* = (g^b)\gamma^*$ and finally give $ct^* = (c_1^*, c_2^*, c_3^*, c_4^*)$ to A. Note that in the above construction, if $Z = e(g, g)^{\frac{b}{a^2}}$, ct^* is indeed a valid ciphertext for m_δ under pk_i^* and w^* . On the other hand, if Z is uniform and independent in G_T , the challenge ciphertext ct^* is independent of δ in the adversary's view.

A continues to issue the rest of queries, with the restrictions described in the IND-oCCA experiment. Algorithm B responds to these queries as in the initial stage. Eventually, adversary A returns a guess $\delta' \in \{0, 1\}$ to B. If $\delta' = \delta$, B outputs $b = 1$ to guess $Z = e(g, g)^{\frac{b}{a^2}}$; otherwise, B outputs $b = 0$ to guess that Z is a random element in groups (G, G_T) .

Theorem 3 The final scheme presented in Section 5.3 is IND-tCCA secure in the random oracle model, assuming the 1-wDBDHI problem is intractable in groups (G, G_T) .

Proof: Suppose B is given as input a 1-wDBDHI challenge tuple $(g, g^a, g^b, Z) \in G^3 \times G_T$ with unknown $a, b \in_R Z_p$. Algorithm B's goal is to decide whether $Z = e(g, g)^a$. Algorithm B acts as challenger and plays the IND-tCCA game with adversary A in the following way.

B gives $param = (G, G_T, g, \{H_i(\cdot)\}_{i=1}^4)$ to A, where $\{H_i(\cdot)\}_{i=1}^4$ are the random oracles controlled by B and can be adaptively asked by A at any time. Algorithm B maintains four hash lists $\{L_i\}_{i=1}^4$, where L_i is initially empty. The oracles $O_{H_2}(\cdot)$, $O_{H_3}(\cdot)$ and $O_{H_4}(\cdot)$ respond in the same way as in the proof of Theorem 2. For queries to $O_{H_1}(\cdot)$, it responds as follows.

$O_{H_1}(\cdot)$: By receiving an input (pk_i, w) , if this query already appears on the list L_1 , then return the

predefined value. Otherwise, choose $\mu \in_R Z_p$, define $H_1(pk_i, w) = g^\mu$. Finally add the tuple $(pk_i, w, coin_{i,w}, \mu)$ to L_1 and respond with μ .

In A's find stage, A issues a series of queries as in the IND-oCCA experiment. B maintains lists L_{sk} , L_{rk} and L_{vk} , which are initially empty, and answer these queries for A as follows.

$O_u(\cdot)$: When receiving the index of the user i as its input, algorithm B first picks $x_i \in_R Z_p$, and defines $pk_i = (g^a)^{x_i}$. Next, it sets $tag_i = 0$ and adds tuple (pk_i, x_i, tag_i) to the L_{sk} . Finally, it returns pk_i to the adversary A.

$O_c(\cdot)$: This oracle works in the same way of that in the proof of Theorem 2.

$O_{[rk]}(\cdot)$: With an input (pk_i, pk_j, w, t, n, U) , B first recovers the tuples (pk_i, x_i, tag_i) and (pk_j, x_j, tag_j) from L_{sk} . Next, it constructs the re-encryption key shares $\{[rk_{ij}]_v\}_{v \in U}$ and the verification key shares $\{[vk_{ij}]_v\}_{v=1}^n$ for adversary A according to the following cases.

Case 1: $tag_i = 1$, it means that $sk_i = x_i$. Using sk_i , algorithm B can certainly run $ReKeyGen(\times)$ to obtain n shares of re-encryption keys $\{[rk_{ij}]_v\}_{v=1}^n$ and verification keys $\{[vk_{ij}]_v\}_{v=1}^n$. Then it stores $(pk_i, pk_j, w, t, n, \{[rk_{ij}]_v\}_{v=1}^n)$ in L_{rk} and $(pk_i, pk_j, w, t, n, \{[vk_{ij}]_v\}_{v=1}^n)$ in L_{vk} . Finally, it returns $\{[rk_{ij}]_v\}_{v \in U}$ and $\{[vk_{ij}]_v\}_{v=1}^n$ to A.

Case 2: $tag_i = 0 \wedge tag_j = 0$, it means that $sk_i = ax_i$ and $sk_j = ax_j$. Algorithm B can also correctly generate the first t shares $\{[rk_{ij}]_v\}_{v=1}^{t-1}$, $\{[vk_{ij}]_v\}_{v=1}^{t-1}$ and simulate the remaining shares $\{[rk_{ij}]_v\}_{v=t}^n, \{[vk_{ij}]_v\}_{v=t}^n$, since it can derive $pk_j^{\frac{1}{sk_i}}$ as $g^{\frac{x_j}{sk_i}}$. Then it stores $(pk_i, pk_j, w, t, n, \{[rk_{ij}]_v\}_{v=1}^n)$ in L_{rk} and $(pk_i, pk_j, w, t, n, \{[vk_{ij}]_v\}_{v=1}^n)$ in L_{vk} . Finally it returns $\{[rk_{ij}]_v\}_{v \in U}$ and $\{[vk_{ij}]_v\}_{v=1}^n$ to A.

Case 3: $tag_i = 0 \wedge tag_j = 1$, it means that $sk_i = ax_i$ and $sk_j = x_j$ and $H_1(pk_i, w) = g^\mu$. Algorithm B can correctly generate $t-1$ shares of

re-encryption keys $\{[rk_{ij}]_v\}_{v=1}^{t-1}$ and verification keys $\{[vk_{ij}]_v\}_{v=1}^{t-1}$. It can also simulate the remaining shares $\{[rk_{ij}]_v\}_{v=t}^n$ and $\{[vk_{ij}]_v\}_{v=t}^n$ following the way of algorithm $ReKeyGen(\cdot)$. Next, it stores $(pk_i, pk_j, w, t, n, \{[rk_{ij}]_v\}_{v=1}^n)$ in L_{rk} and $(pk_i, pk_j, w, t, n, \{[vk_{ij}]_v\}_{v=1}^n)$ in L_{vk} . Finally, it returns $\{[rk_{ij}]_v\}_{v \in U}$ and $\{[vk_{ij}]_v\}_{v=1}^n$ to A .

$O_m(\cdot)$: This oracle works in the same way as in the proof of Theorem 2.

When A decides that the initial stage is over, it outputs two public keys pk_i and pk_j^* , a condition w^* and two equal-length messages $m_0, m_1 \in \{0, 1\}^\lambda$. Algorithm B responds as follows. ① Recover tuple (pk_j^*, x_j^*, tag_j^*) from L_{sk} . ② Flip a random coin $\delta \in_R \{0, 1\}$ and pick $r^* \in_R \{0, 1\}^{\lambda'}$. ③ Define $c_1^* = e((g^b)^{x_j^*}, g)$, $c_2^* = H_2(Z) \oplus (m_\delta \| r^*)$. ④ Finally, give $ct^* = (c_1^*, c_2^*)$ to A .

Note that by the above construction, if $Z = e(g, g)^{\frac{b}{a}}$, ct^* is indeed a valid transformed ciphertext for m_δ under pk_j^* . On the other hand, when Z is uniform and independent in G_T , the challenge ciphertext ct^* is independent of δ in the adversary's view.

A continues to issue the rest of queries as in initial and find Stage, with the restrictions described in experiment IND-tCCA. Algorithm B responds to these queries as described above. Eventually, adversary A returns a guess $\delta' \in \{0, 1\}$ to B . If

$\delta' = \delta$, B outputs 1 to guess $Z = e(g, g)^{\frac{b}{a}}$; otherwise, B outputs 0 to guess that Z is a random element in group G_T .

7 Performance analysis

In this section, we at first analyze the computational complexity of the core scheme in SAAC, and then present an experimental simulation assessing its performance.

During the system setup phase, the basic operating parameters are initialized. The main computa-

tion overhead for Initialization is introduced by the s modular exponentiations in G , needed for generating each user's public key, where s is the number of users in the system.

The main computation overhead characterizing the filestorage operations, outsourced on the cloud, is the encryption of the file performed by using the symmetric-key encryption as well as the encryption of the symmetric encryption key involving TCPRE. The complexity of the former one depends on the size of the underlying file. The overhead for the latter consists of four modular exponentiations (in particular, three of them are in G while one is in G_T).

Regarding file access, most of the operations are carried out by the file owner, as well as by the requesting user and the involved access control servers. In particular, the file owner executes the $ReKeyGen(\cdot)$ algorithm to generate the re-encryption key share for each access control server, which represents $5n$ modular exponentiations in G , where n is the number of access control servers involved; each access control server executes the $ReEncrypt(\cdot)$ algorithm to produce a re-encryption share, which costs 3 exponentiations in G , the requesting user combines at least t shares together to get the re-encrypted ciphertext for the involved file, and finally decrypts this ciphertext. The accessing user's computation overhead includes $2t$ bilinear pairings and t modular exponentiations in G_T for $ShareCombine(\cdot)$, and one modular exponentiation in G_T for $Decrypt(\cdot)$.

We summarize the main computational overhead of our proposed SAAC in Table 2, where E_G and E_{G_T} respectively denote the modular exponentiation in G and G_T , P denotes the bilinear pairing operation, $\xi_1 |f|$ and $\xi_2 |f|$ respectively denote the computational complexity for encrypting and decrypting the file f , both of which depend on its size $|f|$. Notice that in this analysis, we omit the additional computation needed for verifying the correctness of the original data stored in the file since it does not concern our specific access control framework.

Table 2 Computational overhead in SAAC

Phase	Computational overhead
System step	sE_G
File storage	$3E_G + P + E_{G_r} + \xi_1 f $
File access	$5nE_G$ (file owner)
	$3E_G$ (access controller server)
	$2tP + (t+1)E_{G_r} + \xi_2 f $ (file user)

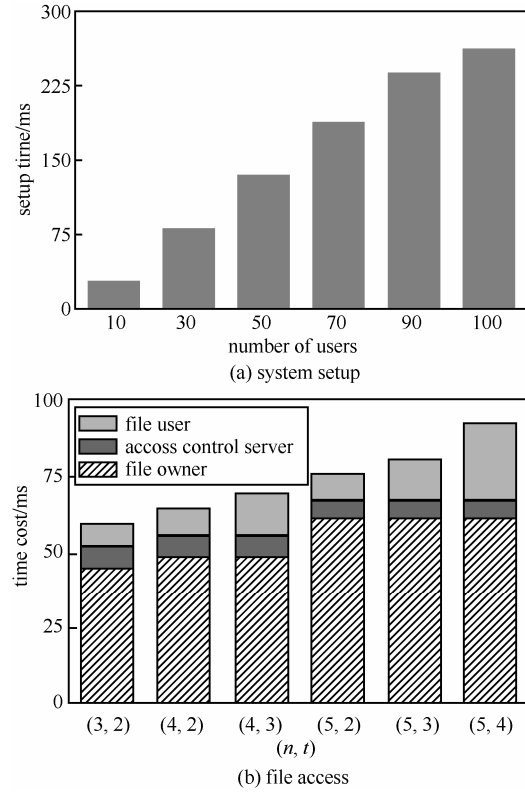
To be more clear on the overall efficiency of the proposed framework, we simulated its core scheme (in particular, the TCPRE scheme) according to the considerations reported in Table 2. The simulation, run on a Mac OS X laptop equipped with an 1.7 GHz Intel Core i7 processor and 8 GB 1 600 MHz DDR3 memory, uses the Pairing-Based Cryptography library^[11] to implement both the users and access control servers activities. We did not consider time measurements for file encryption and decryption operations, since encrypting and decrypting files with symmetric-key encryption in an IND-CPA secure way was a well studied problem for which plenty of performance data was available. Furthermore, some clients may be equipped with highly specialized hardware for accelerating symmetric-key encryption operations.

Figure 3(a) shows the simulated efficiency in terms of system setup in the cases of 10, 30, 50, 70, 90 and 100 users in system. We can observe that the elapsed time is limited by 250 ms even for the maximum number (i.e., 100 in this case) of users. Regarding the outsourced file storage operations, we simulated the average cloud access time as 8.202 ms. Figure 3(b) describes the simulated efficiency for file access by varying the number n of access control servers and the threshold value t . The total elapsed time is limited by 100 ms.

8 Related work

In 1997, Mambo and Okamoto^[12] initially introduced the concept of delegation of decryption rights, as a better-performance alternative to the trivial approach of decrypting-then-encrypting of ciphertexts. In Eurocrypt'98, Blaze et al.^[1] introduced the concept of proxy re-encryption, and pre-

sented the first bidirectional PRE scheme. Ateniese et al.^[5] presented unidirectional PRE schemes that are secure against chosen-plaintext attacks (CPA).

**Figure 3** Time cost in simulation

The first chosen-ciphertext secure bidirectional PRE schemes and unidirectional PRE scheme were proposed by Canetti et al.^[14] and Libert et al.^[14,15] respectively, and both schemes rely on the bilinear pairings. Deng et al.^[16,17] proposed a CCA-secure bidirectional PRE scheme without pairings. Shao et al.^[18] tried to propose a unidirectional PRE scheme without pairings that was improved later by Chow et al.^[19] Subsequently, several bidirectional and unidirectional PRE schemes have been proposed in Refs.[20~23]. Proxy re-encryption has also been studied in identity-based scenarios, such as the Refs.[24~26].

Several variants of PRE have also been proposed in the past few years. Libert et al.^[27] introduced the notion of traceable proxy re-encryption, in which a proxy who leaks its re-encryption key can be identified by the delegator. Ateniese et al.^[28] also introduced the concept of key-private proxy

re-encryption, in which the anonymity of the sender and receiver's identities can be protected. In TCC'12, Chandran et al.^[29] introduced the notion of functional re-encryption, which can transform an encryption of a message m under an "input public key" pk into an encryption of the same message m under one of the n "output public keys", namely the public key index by function $F(m)$. Conditional proxy re-encryption (CPRE) was introduced in Ref.[2,3]. In a CPRE scheme, ciphertexts are generated associated with a certain condition, and the proxy can translate those ciphertexts satisfying the specified condition.

In this work, the core of the proposed SAAC system is the TCPRE scheme, which significantly improves the existing works^[2,3] by quipping the capabilities of guaranteeing the correctness of transformations done by proxies.

9 Conclusion

Aiming at addressing the security and privacy issues in file outsourcing, we propose a server-aided access control system, namely SAAC. It is based on a new proxy re-encryption primitive, namely TCPRE. Compared with traditional CPRE, TCPRE enjoys two features: 1) TCPRE involves a number n of proxies where t out of n proxies can successfully re-encrypt ciphertexts, while up to $t-1$ proxies cannot; 2) The correctness of re-encryption done by each proxy can be publicly verified. We "gradually" provide three TCPRE constructions, and prove that the security of our final construction could be reduced to a new problem which is at least as strong as the wDBDHI problem. The performance of TCPRE construction has been analyzed as well, through simulation, by resulting in a limited overhead also in presence of a large number of users for each single outsourced file.

References:

- [1] BLAZE M, BLEUMER G, STRAUSS M. Divertible protocols and atomic proxy cryptography[C]//The International Conference on the Theory and Applications of Cryptographic Techniques. Berlin

Heidelberg, 1998: 127-144.

- [2] WENG J, DENG R H, DING X, et al. Conditional proxy re-encryption secure against chosen-ciphertext attack[C]//The 4th International Symposium on Information, Computer, and Communications Security, ACM. 2009: 322-332.
- [3] TANG Q. Type-based proxy re-encryption and its construction[C]//The International Conference on Cryptology in India. 2008: 130-144.
- [4] VRABLE M, SAVAGE S, VOELKER G M. Cumulus: filesystem backup to the cloud[J]. ACM Transactions on Storage (TOS), 2009, 5(4): 14.
- [5] ATENIESE G, FU K, GREEN M, et al. Improved proxy re-encryption schemes with applications to secure distributed storage[J]. ACM Transactions on Information and System Security (TISSEC), 2006, 9(1): 1-30.
- [6] LI J, CHEN X, LI J, et al. Fine-grained access control system based on outsourced attribute-based encryption[C]//European Symposium on Research in Computer Security. 2013: 592-609.
- [7] SHAMIR A. How to share a secret[J]. Communications of the ACM, 1979, 22(11): 612-613.
- [8] BONEH D, BOYEN X, GOH E J. Hierarchical identity based encryption with constant size ciphertext[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. 2005: 440-456.
- [9] Final report on main computational assumptions in cryptography[EB/OL]. <http://www.ecrypt.eu.org/documents/D.MAYA.6.pdf>, 2013.
- [10] CORON J S. On the exact security of full domain hash[C]//Annual International Cryptology Conference. 2000: 229-235.
- [11] Pairing-based cryptography library[EB/OL]. <http://crypto.stanford.edu/pbc/>.
- [12] MAMBO M, OKAMOTO E. Proxy cryptosystems: delegation of the power to decrypt ciphertexts[J]. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 1997, 80(1): 54-63.
- [13] CANETTI R, HOHENBERGER S. Chosen-ciphertext secure proxy re-encryption[C]//The 14th ACM conference on Computer and Communications Security. 2007: 185-194.
- [14] LIBERT B, VERGNAUD D. Unidirectional chosen-ciphertext secure proxy re-encryption[J]. IEEE Transactions on Information Theory, 2011, 57(3): 1786-1802.
- [15] LIBERT B, VERGNAUD D. Unidirectional chosen-ciphertext secure proxy re-encryption[C]//International Workshop on Public Key Cryptography. 2008: 360-379.
- [16] WENG J, DENG R H, LIU S, et al. Chosen-ciphertext secure bidirectional proxy re-encryption schemes without pairings[J]. Information Sciences, 2010, 180(24): 5077-5089.
- [17] DENG R H, WENG J, LIU S, et al. Chosen-ciphertext secure proxy re-encryption without pairings[C]//International Conference on Cryptology and Network Security. 2008: 1-17.
- [18] SHAO J, CAO Z. CCA-secure proxy re-encryption without pairings[C]//International Workshop on Public Key Cryptography. 2009:

357-376.

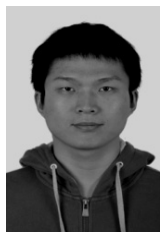
- [19] CHOW S S M, WENG J, YANG Y, et al. Efficient unidirectional proxy re-encryption[C]//International Conference on Cryptology in Africa. 2010: 316-332.
- [20] WENG J, ZHAO Y, HANAOKA G. On the security of a bidirectional proxy re-encryption scheme from PKC 2010[C]// International Workshop on Public Key Cryptography. 2011: 284-295.
- [21] HANAOKA G, KAWAI Y, KUNIHIO N, et al. Generic construction of chosen ciphertext secure proxy re-encryption[C]// Cryptographers' Track at the RSA Conference. 2012: 349-364.
- [22] WENG J, CHEN M, YANG Y, et al. CCA-secure unidirectional proxy re-encryption in the adaptive corruption model without random oracles[J]. Science China Information Sciences, 2010, 53(3): 593-606.
- [23] MATSUDA T, NISHIMAKI R, TANAKA K. CCA proxy re-encryption without bilinear maps in the standard model[C]// International Workshop on Public Key Cryptography. 2010: 261-278.
- [24] GREEN M, ATENIESE G. Identity-based proxy re-encryption[C]// Applied Cryptography and Network Security. 2007: 288-306.
- [25] MATSUO T. Proxy re-encryption systems for identity-based encryption[C]//International Conference on Pairing-Based Cryptography. 2007: 247-267.
- [26] CHU C K, TZENG W G. Identity-based proxy re-encryption without random oracles[C]//International Conference on Information Security. 2007: 189-202.
- [27] LIBERT B, VERGNAUD D. Tracing malicious proxies in proxy re-encryption[C]//The International Conference on Pairing-Based Cryptography. 2008: 332-353.
- [28] ATENIESE G, BENSON K, HOHENBERGER S. Key-private proxy re-encryption[C]//Cryptographers' Track at the RSA Conference. 2009: 279-294.
- [29] CHANDRAN N, CHASE M, VAIKUNTANATHAN V. Functional re-encryption and collusion-resistant obfuscation[C]//Theory of Cryptography Conference. 2012: 404-421

About the authors:

WENG Jian(1976-), born in Guangdong. In 2008, he received his Ph.D. degree in Computer Science and Engineering from Shanghai Jiaotong University. Currently, he is a professor and vice dean with the School of Information Technology, Jinan University. His research interests include cryptography and information security.



WENG Jiasi(1994-), born in Guangdong. She is a master in Jinan University. Her research interests include cryptography and information security.



LIU Jianan(1992-), born in Henan. He is a Ph.D student in Jinan University. His research interests include cryptography and information security.



HOU Lin (1991-), born in Hubei. She is a Ph.D student in Jinnan University. Her research interests include cryptography and information security.