Received May 20, 2019, accepted May 31, 2019, date of publication June 5, 2019, date of current version June 21, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2920998

Secure Encrypted Data With Authorized Deduplication in Cloud

JINBO XIONG[1], (Member, IEEE), YUANYUAN ZHANG[2,3], SHAOHUA TANG[2,3], (Member, IEEE), XIMENG LIU[4], (Member, IEEE), AND ZHIQIANG YAO[1]

[1]Fujian Provincial Key Laboratory of Network Security and Cryptology, College of Mathematics and Informatics, Fujian Normal University, Fuzhou 350117, China
[2]School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China
[3]Peng Cheng Laboratory, Shenzhen 518055, China
[4]Fujian Provincial Key Laboratory of Information Security of Network Systems, College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China

Corresponding author: Yuanyuan Zhang (zyy837603010@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61872088, Grant 61632013, Grant 61872090, Grant 61702105, and Grant U1804263, and in part by the Natural Science Foundation of Fujian Province under Grant 2019J01276.

ABSTRACT In this paper, we propose a novel secure role re-encryption system (SRRS), which is based on convergent encryption and the role re-encryption algorithm to prevent the privacy data leakage in cloud and it also achieves the authorized deduplication and satisfies the dynamic privilege updating and revoking. Meanwhile, our system supports ownership checking and achieves the proof of ownership for the authorized users efficiently. Specifically, we introduce a management center to handle with the authorized request and establish a role authorized tree (RAT) mapping the relationship of the roles and keys. With the convergent encryption algorithm and the role re-encryption technique, it can be guaranteed that only the authorized user who has the corresponding role re-encryption key can access the specific file without any data leakage. Through role re-encryption key updating and revoking, our system achieves the dynamic updating of the authorized user's privilege. Furthermore, we exploit the dynamic count filters (DCF) to implement the data updating and improve the retrieval of ownership verifying effectively. We conduct the security analysis and the simulation experiment to demonstrate the security and efficiency of our proposed system.

INDEX TERMS Role re-encryption, role authorized tree, privacy leakage, authorized deduplication, proof of ownership.

I. INTRODUCTION

The rapid development of cloud computing and big data technology changes user's method and efficiency in processing information, the cloud servers provide the scalable computing and efficient storage to users in anytime and anywhere. Therefore, for enterprises and individuals, it will be a trend to outsource data to the cloud service providers (CSP) [1], [2]. A report "Internet data generated in one minute" of Excelcom shows that more than 701389 accounts log onto Facebook in one minute, more than 300 hours new video and audio are uploaded to YouTube in one minute. The users generate 2.4 million search requests in Google search, and the users post more than 24.30 million photos to the Instagram social application per minute [3], [4]. The amount of global uploaded data reached 4 billion or more per day in 2011, the data volume reached 1.8 ZB. IDC (International Data Corporation) statistic shows that the global data volume reached 4.4 ZB in 2013, reached 8.61 ZB by the end of 2015, the growth rate of data volume is more than 50%, and expected to 2020, it will unexpectedly reach 44 ZB [5]. More than half of the cloud storage space is occupied by the duplicate data, and the expenditure for managing the duplicate data is 8 times that of the original data [6]. Along with the explosive growth of cloud data, massive duplicate data occupied the storage space and the huge expenditure bring a severe challenge to the limited cloud storage space. Therefore, how to reduce the management expenditure and improve the storage efficiency in cloud is an urgent issue to be solved for the cloud service providers [7]–[9].

The associate editor coordinating the review of this manuscript and approving it for publication was Kuo-Hui Yeh.

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

To tackle the above issue, data deduplication is widely applied in the cloud service providers, which eliminates the multiple duplicate data to improve the storage utilization and reduce management expenditure [10]. Recent research [11] shows that data deduplication can reduce up to 83% for backup systems and 68% for memory systems [12]. According to different criteria, the classification results for data deduplication have multiple types. Based on the data processing unit, it can be divided into file-level data deduplication and block-level data deduplication; based on the data execution object, it can be divided into server-side data deduplication (target-based data deduplication), client-side data deduplication (source-based data deduplication), and cross-user data deduplication [13]–[15]. The cross-user data deduplication can save more storage space, and the deduplication rate is up to 90%-95% [16], [17]. Although data deduplication brings above benefits, it also poses some new security challenges [18]–[20]. Firstly, the data outsourced by enterprises and individuals to cloud service providers usually involves the privacy information [21]–[23]. The users lose the control over these privacy data, which exists the risk of privacy leakage by the curious cloud service providers [24]. At the same time, in order to improve the management efficient, the cloud server providers perform data deduplication to control the number of duplicate data. In the process of data deduplication, the adversary may utilize the relative attack methods to intercept the user's privacy information, which also discloses the user's identity, location, and number of duplicate data. Therefore, it is important to protect the user's privacy when performing data deduplication. Secondly, the data deduplication based on the traditional convergent encryption poses serious security problems, the unauthorized users can obtain the user's information only by supplying the hash value of the file, which makes it difficult to protect the data security, check the ownership and achieve authorization access. Only the user who has the corresponding privilege can access the specific file and perform the data deduplication in cloud, which is an urgent issue to be solved. Thirdly, the privilege of the authorized user is dynamic and flexible, it is difficult to guarantee the access permission of authorized user and achieve the key updating and revoking management when performing data deduplication.

To tackle the above issues, we propose a novel secure role re-encryption system with authorized deduplication called SRRS, which is based on the convergent encryption and the role re-encryption algorithm to achieve authorized deduplication. Up to now, we propose the first solution to prevent privacy data leakage, achieve authorized deduplication and satisfy dynamic privilege updating and revoking, meanwhile, support ownership checking. The main contributions of the proposed scheme are fourfold:

- We proposed a novel secure role re-encryption system (SRRS) with authorized deduplication in cloud, which is based on the convergent encryption to protect data privacy and achieve secure data deduplication, utilizes the role re-encryption to achieve authorized access. In SRRS, we introduce a management center to handle with the authorized request, and generate the role re-encryption key, which guarantees that only the authorized user who have the corresponding role re-encryption key used to access the specific file.
- We construct a role authorized tree to manage the user's role and implement the role re-encryption key updating and revoking efficiently, which satisfies the dynamic updating of authorized user's privilege.
- When performing the secure data deduplication, CSP can check the ownership of the authorized user. We exploit the dynamic count filters (DCF) to achieve the data updating and improve the retrieval efficiency of ownership verifying.
- Security analysis shows that our proposed system is secure under the proposed security model, and performance evaluation demonstrates the effectiveness and efficiency of our proposed system.

The rest of our paper is organized as follows: Section II gives the preliminaries in our work; and Section III describes the system model, adversary model and design goal; in Section IV, we construct the proposed system; Section V and Section VI gives the security analysis and the performance evaluation, respectively; We describe the related work in Section VII; and give the conclusion in the end.

II. PRELIMINARIES

In this section, we give the preliminaries in our work about B+ tree and DCF.

A. B+ TREE

B+ tree has two types of nodes: internal nodes and leaf nodes [25]. The internal node, including a root node, stores the index information, and the leaf node stores the elements related to data. All of the leaf nodes are linked by a doubly linked list. A B+ tree of order M satisfies the following properties:

- The root node has 2 child nodes at least, and has M child nodes at most.
- With except of the root node, each internal node has $M - 1$ key values at least, and has $\lceil M/2 - 1 \rceil$ key values at most.
- Each leaf node has $\lceil M/2 \rceil$ elements at least.

B+ tree is a tree-like data structure that stores and searches data in order, which supports data searching, sequentially reading, insert and deletion. The non-leaf nodes of the B+ tree contain only index information, and do not store actual values. All leaf nodes and the connected nodes are connected by a linked list, which is convenient for interval searching and traversal. Those properties are used in the construction of role authorization tree to implement quickly querying and retrieving user's role information in the SRRS system.

B. CONVERGENT ENCRYPTION

Convergent encryption is a specific symmetrical encryption and provides data confidentiality in deduplication process [18], [26].

The key of convergent encryption is a value of a cryptographically strong hash calculating from the original file content. Therefore, two users with the identical plaintext files can get the identical hashes and identical keys to obtain identical ciphertext files without considering their encryption keys.

Formally, given a symmetric-key encryption function *Enc*, a plaintext file f and a cryptographically strong hash function h, the convergent key is $k = h(f)$, and the ciphertext $C = Enc_k(f)$.

In the SRRS system, we employ the convergent encryption to protect data privacy and achieve secure data deduplication.

C. DYNAMIC COUNT FILTERS

A dynamic count filters (DCF) [27] is used to check whether a specified element is in a set, and consists of two different vectors and a series of random mapping functions. Compared with the standard bloom filter, DCF can handle deletes and inserts on multisets with less consumption and higher efficiency. Each element is mapped to an array used the hash functions. For the retrieved element, if all the mapping results are equal to 1, it exists in the set. Otherwise, it has not been in the set. However, the retrieved result has a certain probability of false positive.

The process of initialization, insertion, retrieval and deletion are described as follows:

- $InitDCF(\alpha, \beta, t)$: $CBF \leftarrow \{CU_i\}_{i\in[1,t]}$, $OFV \leftarrow \{OF_i\}_{i\in[1,t]}$. Initialize a DCF, where t is the number of counters, each CU_i has size of α bits, each OF_i has size of β, where β varies dynamically its bit length.
- $InseDCF(EM, \gamma) : CBF \leftarrow \{EM_{h_j}\}_{j\in[1,\gamma]}$. Insert a new element EM, where $\{h_j\}_{j\in[1,\gamma]}$ are γ independent hash functions, then, updating the CBF and OFV.
- $RetrDCF(EM, DCF)$: $\perp \Leftrightarrow \exists i \in [1, t] : V_i = (2^\alpha \times OF_i + Cu_i) \leftarrow 0$. Retrieve whether the element EM exists in the set, the value of the i-th counters is V_i.
- $DeleDCF(EM, DCF)$: $CBF \leftarrow \{EM_{h_j}\}_{j\in[1,\gamma]}$. Delete an element EM, then, update the CBF and OFV.

In the SRRS system, we obtain the encrypted blocks and insert the results to DCF to achieve the data updating and improve the retrieval efficiency of ownership verifying.

III. PROBLEM DESCRIPTION

In this section, we firstly give a system model. Then, we describe an adversary model and our design goal.

A. SYSTEM MODEL

The system model of SRRS consists of three entities: Users (U), a Cloud Service Provider (CSP) and a Management Center (MC), as shown in Figure 1. A user sends a request to MC, and encrypts the original file, then, outsources the ciphertext to CSP.

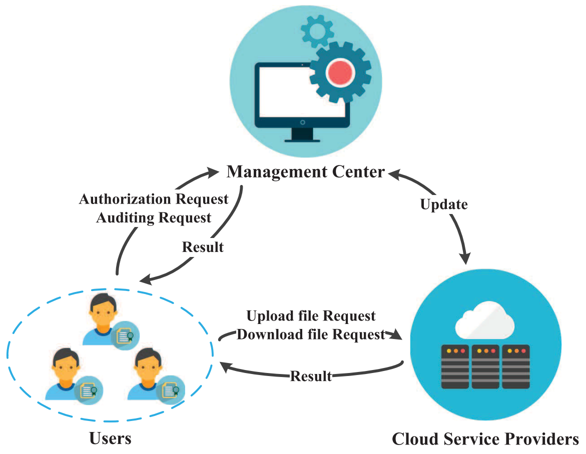

FIGURE 1. System model of SRRS.

Users who belong to different role groups owning the corresponding role keys. According to the role keys and access control policies, the user can upload or download the specific files from CSP. The creator of a file is unique, and is also a special user.

A **CSP** is responsible for data storage, management and verification. CSP stores and manages the uploaded files from users. In terms of verification, CSP establishes a challenge list for each file to verify the user's ownership to prevent the unauthorized user's access.

A **MC** is a trusted third party, which is responsible for user authorization and role key management.

B. ADVERSARY MODEL

In our proposed system, we assume that: ① the MC is trusted by all entities involved in our system, and will not compromised by an adversary; ② the CSP is "honest-but-curious (HbC)", which performs our proposed protocol honestly, but is curious about the user's privacy information; ③ the communication channel between MC and the user is secure in our system; ④ we define a secure hash function resisted against the collision attack and exploit the standard symmetric encryption algorithm [4], [28].

According to the above assumptions, we define the following abilities of different types of adversaries [29], [30].

- $\mathcal{A}_1$ could eavesdrop the communications between CSP and the user to get the transmitted information, and plays a role of the user to interact with the CSP.
- $\mathcal{A}_2$ could eavesdrop the communications between CSP and the user to get the transmitted information, and cloud exchange S_{min} bytes information with the user.
- $\mathcal{A}_3$ may discard the user's data that have not been accessed or rarely accessed, and may tamper the user's data to maintain reputation.

C. DESIGN GOAL

We consider two aspects of system security and performance efficiency to construct our system, therefore, the design goal includes security and performance requirement, which is described as follows:

- Our proposed system is secure under the standard model. Specifically, ① the probability of an adversary running a successful secure protocol should be negligible under the secure parameter; ② the privacy information of users cannot be acknowledged by the CSP in the process of performing data deduplication; ③ the adversary can obtain a minimum amount S_{min} of the privacy information from the authorized user to run successful security protocol. Meanwhile, the system supports dynamic updating and revoking of privileges to achieve flexible access control.
- The bandwidth, the server memory space, the client storage space should be efficient and economical. In the process of performing authorized deduplication, the bytes of file exchanged between the user and CSP should be the smallest possible to reduce the communication overhead; the loaded information in the server memory should be the smallest possible and should be independent of the uploaded file size to reduce the memory overhead; the information about keys and ciphertext stored to the client should be the minimum possible, in addition, the number and length of the stored keys should be independent of the file size to reduce the storage overhead.

IV. CONSTRUCTION OF THE PROPOSED SYSTEM

In this section, we firstly give the notations and descriptions, as shown in Table 1, and we define that ":=" represents a define operator, "←" represents an assignment operator.

TABLE 1. Notations and descriptions.

Notations	Descriptions
λ	the system secure parameter
f	a file
n	the number of file blocks
b	a file block
$token$	an index of file
$role_{u_i}$	User u_i's role
id_{u_i}	User u_i's identifier
$rkey$	a role key
C	a ciphertext
k	a convergent key
C'	a re-encryption ciphertext
$\Im$	a four-tuple array to store data information
C''	an updating re-encryption ciphertext
h	a cryptographic hash function

The overview of the proposed SRRS system consists of the following three phases: authorized deduplication (Phase 1), proof of ownership (Phase 2), and role key update (Phase 3), as shown in Figure 2.

In phase 1, a file creator U_1 calls **`Divided`** algorithm to divide the file, then calls **`Encrypt`** algorithm to encrypt the file. MC calls **`rkeyGen`** algorithm to generate the role key, and calls **`Decomp`** algorithm to obtain the role key shares, then sends the role key and role key shares to the creator. After receiving the information, U_1 calls **`ReEncrypt`** algorithm, and sends to CSP. In phase 2, U_2 uploads a file to CSP, CSP then finds that the same file has been stored, it will perform proof of ownership for U_2. CSP generates and sends the challenge of the proof of ownership to U_2, who calls **`Divided`**, **`Encrypt`** and **`ReEncrypt`** algorithms to compute response, and sends the response to CSP, CSP will perform **`Verify`** algorithm. U_2 downloads a file from CSP, who calls **`Verify`** algorithm and sends ciphertext to U_2, U_2 then calls **`ReDecrypt`** and **`Decrypt`** algorithms to obtain the file. In phase 3, MC calls **`rkeyUp`** algorithm to update the role key, and CSP retrieves data information and updates date information.

A. ROLE AUTHORIZED TREE

We define a novel authorization structure named role authorized tree (RAT) based on a B+ tree, as shown in Figure 3. MC organizes a role group using a RAT to manage the role key and achieve the user's authorization.

For a RAT with order M, each node contains M elements at most, and $\lceil M/2 \rceil$ elements at least. A RAT has two type of nodes: leaf nodes and internal nodes, the root node belongs to the internal nodes. The leaf nodes store data information, and the internal nodes do not have these data.

We define a RAT with order 3, each node can store 3 elements at most. Each node V consists of three values, the definitions are described as follows:

- $contain(V)$. The $contain(V)$ stores the elements of this node based on the index from left to right, $contain(V) := \{v_1 \| v_2\}$.
- $children(V)$. The $children(V)$ stores the child nodes of this node, consists of left child, the middle child, the right child respectively, $children(V) := \{W_1 \| W_2 \| W_3\}$. For a leaf node, the child nodes will be null, $children(V) = \{null\}$.
- $rank(V)$. The $rank(V)$ stores the number of elements. For a RAT with order 3, each node can contain 3 elements at most, and 2 elements at least. Therefore, the $rank(V)$ can be described as:

$$rank(V) = \begin{cases} 2 & \text{the node has 2 elements.} \\ 3 & \text{the node has 3 elements.} \end{cases}$$

We define the elements of the leaf node corresponding to the user's roles, each role group consists of a two-tuple $R =< rolekey, filelist >$, where $R[i].rolekey$ is the role key and $R[i].filelist$ is the list of file indexes, and i is the index of the role group. Each role group holds the corresponding role key and manages the corresponding files. The user who owns the particular role has the privilege to manage the corresponding files.

In order to obtain the role group and the corresponding role key, MC must search the RAT. To search for the role group R_ι, MC needs to retrieve the element l_ι. MC takes the retrieved role group R_ι as input and returns the role key $rkey_\iota$, the procedures of search process are described as follows:

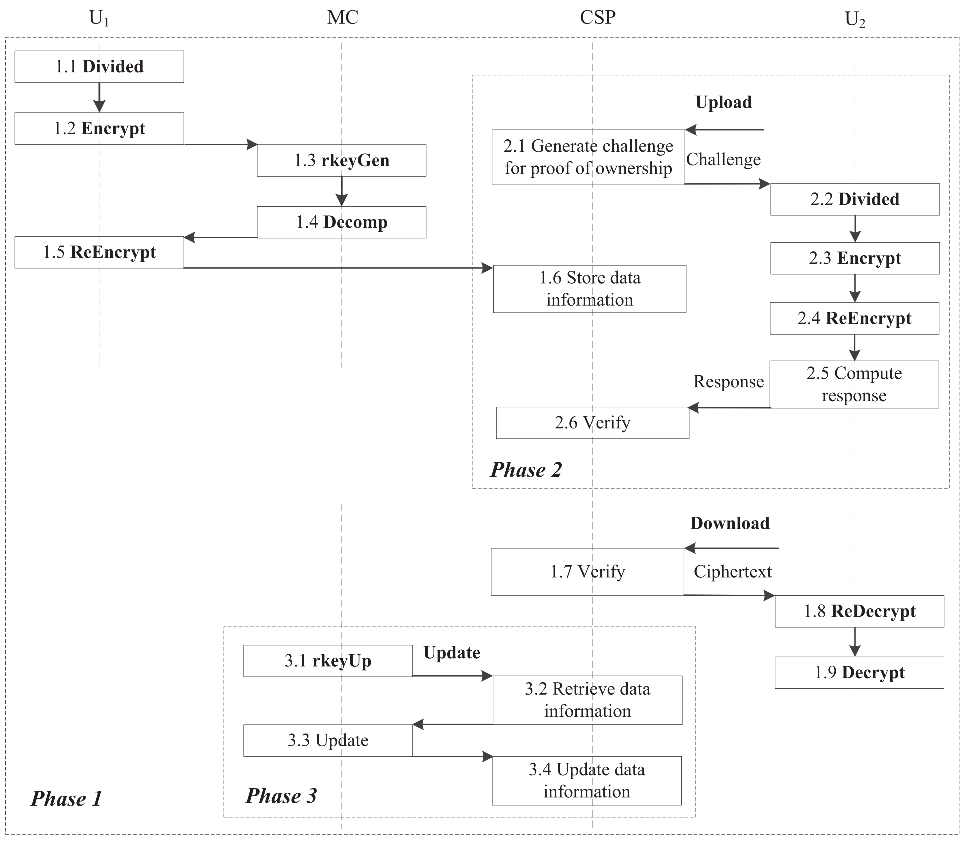

FIGURE 2. Overview of the SRRS system.

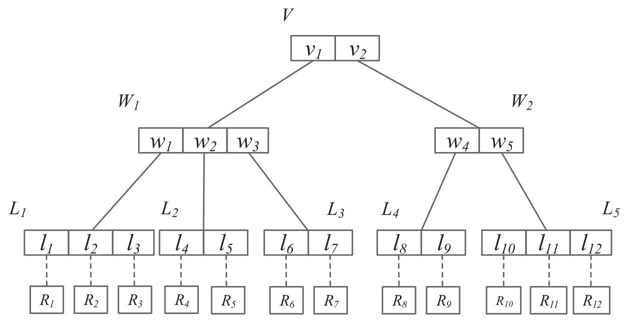
V
v1 v2
W1
W2
w1 w2 w3
w4 w5
L1 L2 L3 L4 L5
l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 l11 l12
R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12

FIGURE 3. Role authorized tree.

Step.1: MC gets $contain(\cdot)$ and $rank(\cdot)$ from current node X_i. If $rank(X_i) = 2$, MC needs to compare l_ι with two elements x_j, x_{j+1} of node X_i; If $rank(X_i) = 3$, MC needs to compare l_i with three elements x_j, x_{j+1}, x_{j+2} of node X_i.

Step.2: MC checks $children(X_i)$ to get the corresponding child node Y_j based on the result of *step.1*.

Step.3: MC invokes *Algorithm 1* to check node Y_j, and obtains $contain(Y_j)$ and $rank(Y_j)$.

MC repeats from *Step.1* to *Step.3* until $children(X_i) = \{null\}$, where the current node X_i is a leaf node, then, returns the target elements l_ι. Finally, MC obtains the corresponding role group R_ι and the role key $rkey_\iota$.

The *Algorithm 1* describes the procedure of **Check** Y_j, which takes the target node Y_j as input, and returns $contain(Y_j)$ and $rank(Y_j)$. Firstly, MC gets $rank(\cdot)$ from the current node Y_j, if $rank(Y_j) = 2$, MC gets two elements from node Y_j, $contain(Y_j) := \{y_t \| y_{t+1}\}$, then, returns $contain(Y_j)$ and $rank(Y_j)$; if $rank(Y_j) = 3$, MC gets three elements from node Y_j, $contain(Y_j) := \{y_t \| y_{t+1} \| y_{t+2}\}$, then, returns $contain(Y_j)$ and $rank(Y_j)$.

Algorithm 1 Check Y_j

```
input: the target node $Y_j$
output: $contain(Y_j)$ and $rank(Y_j)$
1: get $rank(Y_j)$
2: if ($rank(Y_j) = 2$) then
3:     get $contain(Y_j) := \{y_t \| y_{t+1}\}$;
       return $contain(Y_j)$ and $rank(Y_j)$;
4: end if
5: if ($rank(Y_j) = 3$) then
6:     get $contain(Y_j) := \{y_t \| y_{t+1} \| y_{t+2}\}$;
       return $contain(Y_j)$ and $rank(Y_j)$;
7: end if
```

B. ALGORITHMS

In this section, we elaborately describe all the algorithms in the SRRS system, as shown in Figure 2.

`Divided`. The user executes the algorithm to divided the file into n blocks. It takes a file f and a file block number n as inputs, and outputs a set of file blocks $\{b_i\}_{i \in [1,n]}$.

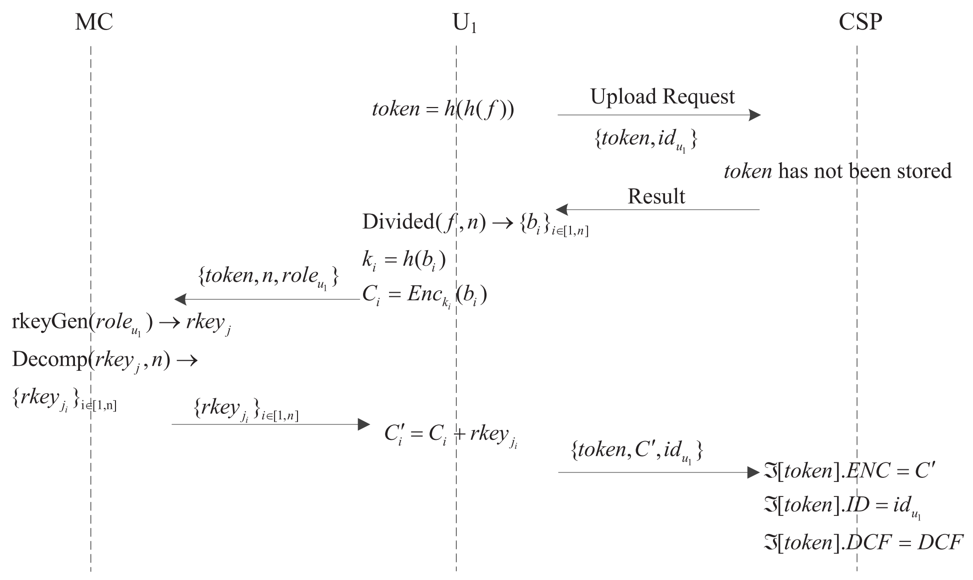

FIGURE 4. **Authorized deduplication: the process of uploading a file for the first time.**

Encrypt. The user executes the algorithm to encrypt the file blocks with the convergent key. It takes a set of file blocks $\{b_i\}_{i\in[1,n]}$ as input, and returns the ciphertext $\{C_i\}_{i\in[1,n]}$. The user runs the hash over file blocks $\{b_i\}_{i\in[1,n]}$ to obtain the convergent key $\{k_i\}_{i\in[1,n]}$, $k_i = h(b_i)$. Then, the user symmetrically encrypts the file blocks with the convergent key to obtain the ciphertext C_i, $C_i = Enc_{k_i}(b_i)$, where Enc is a symmetric encryption function.

rkeyGen. It takes a user's role $role_{u_i}$ as input, and returns a role key $rkey_j$, $rkey_j = role_{u_i}.rolekey$. The specific user u_i is associated with the role key $rkey_j$. Upon receiving an authorization request with the user's role, MC searches the node of RAT to obtain the corresponding two-tuple $R[u_i]$, and returns the role key $rkey_j$. Each role is associated with a unique role key and has the corresponding privilege.

Decomp. MC executes the algorithm to decompose the role key into n shares. It takes the role key $rkey$ and key share number n as inputs, and returns a set of key shares $\{rkey_i\}_{i\in[1,n]}$. The key share number is identical to the file block number.

ReEncrypt. It takes a set of ciphertext blocks and a set of role key shares as inputs, and returns the re-encryption ciphertext $\{C'_i\}_{i\in[1,n]}$. Upon receiving n shares of the role key $\{rkey_i\}_{i\in[1,n]}$, the user concatenates the ciphertext blocks with the role key shares to obtain the re-encryption ciphertext $\{C'_i\}_{i\in[1,n]}$, $C'_i = C_i + rkey_i$.

ReDecrypt. It takes a set of re-encryption ciphertext blocks and a set of role key shares as inputs, and returns the ciphertext $\{C_i\}_{i\in[1,n]}$. Upon receiving n shares of the role key $\{rkey_i\}_{i\in[1,n]}$, the user splits the re-encryption ciphertext blocks with the role key shares to obtain the ciphertext $\{C_i\}_{i\in[1,n]}$, $C_i = C'_i - rkey_i$.

Decrypt. The user executes the algorithm to decrypt the ciphertext with the convergent key. It takes a set of ciphertext blocks $\{C_i\}_{i\in[1,n]}$ as input, and returns the original file blocks $\{b_i\}_{i\in[1,n]}$. The user decrypts the ciphertext with the convergent key to obtain the original file blocks, $b_i = Dec_{k_i}(C_i)$, where Dec is a decryption function.

rkeyUp. It takes an original role key $rkey$ as input, and returns the updated role key $rkey'$. MC searches the RAT and updates the specific role key, $rkey \rightarrow rkey'$.

C. AUTHORIZED DEDUPLICATION

In our proposed system, we define user U_1 as the creator of a file f and U_2 as one of the users that owns the same file. Figure 4 and Figure 5 illustrate the procedure of authorized deduplication.

U_1 wants to store file f to the CSP, as shown in Figure 4. Firstly, U_1 computes a token of f, $token = h(h(f))$, and sends an upload request with $\{token, id_{u_1}\}$ to CSP. Upon receiving the token, CSP checks it, if it has not been stored, and requests U_1 to upload the file with related information. Secondly, U_1 calls **Divided** to obtain n file blocks, $\{b_i\}_{i\in[1,n]}$, and invokes **Encrypt** to get the ciphertext $C_i = Enc_{k_i}(b_i)$, where k_i is the convergent key of the ciphertext block, $k_i = h(b_i)$. Thirdly, U_1 sends the authorized request with $\{token, n, role_{u_1}\}$ to MC. Upon receiving the request, MC searches the node from RAT to obtain the corresponding two-tuple $R[u_1]$, and calls **rkeyGen** to generate the role key $rkey$. In order to obtain the role key shares, MC invokes **Decomp**, where inputs the role key $rkey$ and the key share number n and returns a set of key shares $\{rkey_i\}_{i\in[1,n]}$. Then, MC sends the result to U_1. U_1 calls **ReEncrypt** to obtain the re-encryption ciphertext C', $C'_i = C_i + rkey_i$, $i \in [1, n]$. Finally, U_1 uploads $\{token, C', id_{u_1}\}$ to CSP.

Upon receiving the data package, CSP establishes an associative array $\Im$ that maps a string with finite size to 3-tuples:

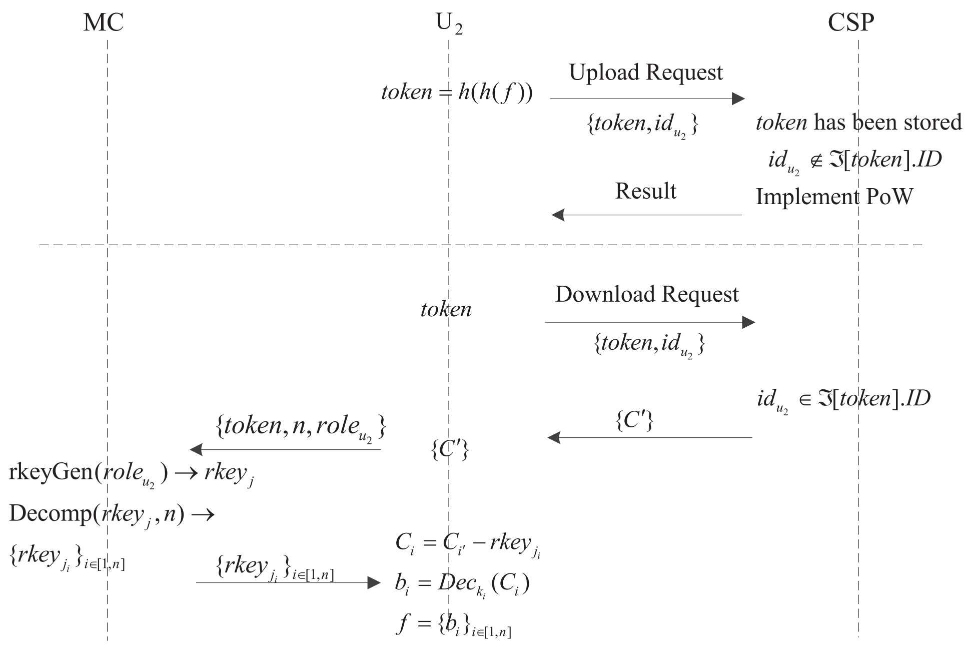
MC
U_2
CSP
$token = h(h(f))$
Upload Request
$\{token, id_{u_2}\}$
$token$ has been stored
$id_{u_2} \notin \Im[token].ID$
Implement PoW
Result
$token$
Download Request
$\{token, id_{u_2}\}$
$id_{u_2} \in \Im[token].ID$
$\{C'\}$
$\{token, n, role_{u_2}\}$
$\{C'\}$
$\text{rkeyGen}(role_{u_2}) \to rkey_j$
$\text{Decomp}(rkey_j, n) \to$
$\{rkey_{j_i}\}_{i\in[1,n]}$
$\{rkey_{j_i}\}_{i\in[1,n]}$
$C_i = C_{i'} - rkey_{j_i}$
$b_i = Dec_{k_i}(C_i)$
$f = \{b_i\}_{i\in[1,n]}$

FIGURE 5. **Authorized deduplication: the process of uploading and downloading a file.**

$\Im[token].ENC$, $\Im[token].ID$ and $\Im[token].DCF$, where token is the index of 3-tuples. Specifically, $\Im[token].ENC$ stores the re-encryption ciphertext C', $\Im[token].ID$ contains a list of identifiers for users, $\Im[token].DCF$ keeps a bits of dynamic count filters to check ownership.

The establishment of DCF is described as follows:

Step.1: CSP initializes DCF with the file block number n.

Step.2: CSP calculates the $index[i]$, that is the hash value of each block $C'[i]$, $index[i] = h(C'[i])$.

Step.3: CSP takes the $index[i]$ and the index i as inputs, runs PRF to get the inserted element em, $em \leftarrow PRF(index[i], i)$.

Step.4: An element em as the outputs of PRF is inserted into DCF, $InseDCF(em, \gamma)$.

CSP repeats from the *Step.2* to the *Step.4* until $i = n$, the encrypted blocks are inserted into DCF.

U_2 wants to store the same file at the CSP, as shown in Figure 5. U_2 computes a token of file, $token = h(h(f))$, and sends the upload request with $\{token, id_{u_2}\}$ to CSP. Upon receiving the token, CSP checks if the token has been stored, then checks if $id_{u_2} \in \Im[token].ID$ is true. If the result is true, CSP informs U_2 that the file has been stored. If not, CSP implements PoW protocol to check the ownership of U_2, and returns the corresponding result to U_2 (PoW protocol refers to Sect.4.4).

U_2 wants to download the file stored in the CSP, as shown in Figure 5. U_2 sends the download request with $\{token, id_{u_2}\}$ to CSP, CSP checks if $id_{u_2} \in \Im[token].ID$ is true. If the identifier of U_2 exists in the user list, CSP retrieves the associative array based on the index value of token and returns the re-encryption ciphertext C'. U_2 sends the role key request to MC, and calls **`ReDecrypt`** using the role key to decrypt the re-encryption ciphertext C'. Finally, U_2 calls **`Decrypt`** using the convergent key k to obtain the original file f.

D. PROOF OF OWNERSHIP

In order to solve the problem that an adversary can explore some sensitive data simply by supplying the hash values of a file, CSP implements PoW protocol when one of users uploads file that has been stored. We define U_3 is one of the users who owns the file f, and f has been stored in the CSP, meanwhile, the identifier of U_3 has not been stored in the list of owner identifier, as shown in Figure 6.

U_3 computes a token of f, $token = h(h(f))$, and sends the upload request with $\{token, id_{u_3}\}$ to CSP. Upon receiving the token, CSP checks whether the token has been stored or not, then checks whether $id_{u_3} \in \Im[token].ID$ is true or not. After verification, the token has been stored at CSP and id_{u_3} does not belong to the identifier list. Therefore, CSP generates an array pos of J randomly and independently chosen block indexes based on $\Im[token].ENC$. Finally, CSP sends pos to U_3, where pos is a challenge to verify a user who indeed owns f.

Upon receiving the challenge, U_3 calls **`Divided`** to get n file blocks, $\{b_i\}_{i\in[1,n]}$, and invokes **`Encrypt`** to get the ciphertext $C_i = Enc_{h(b_i)}(b_i)$, and interacts with MC to obtain re-encryption ciphertext C'. Then, U_3 calculates the corresponding responses res according to the pos and returns the result to CSP. Finally, CSP takes res and pos as inputs, calls PRF to get a retrieved element em, and checks whether the em belongs to $\Im[token].DCF$ or not, if $RetrBF(em, \Im[token].DCF)$ is true, CSP returns OK to U_3 and adds the id_{u_3} to $\Im[token].ID$. Otherwise, U_3 fails to PoW

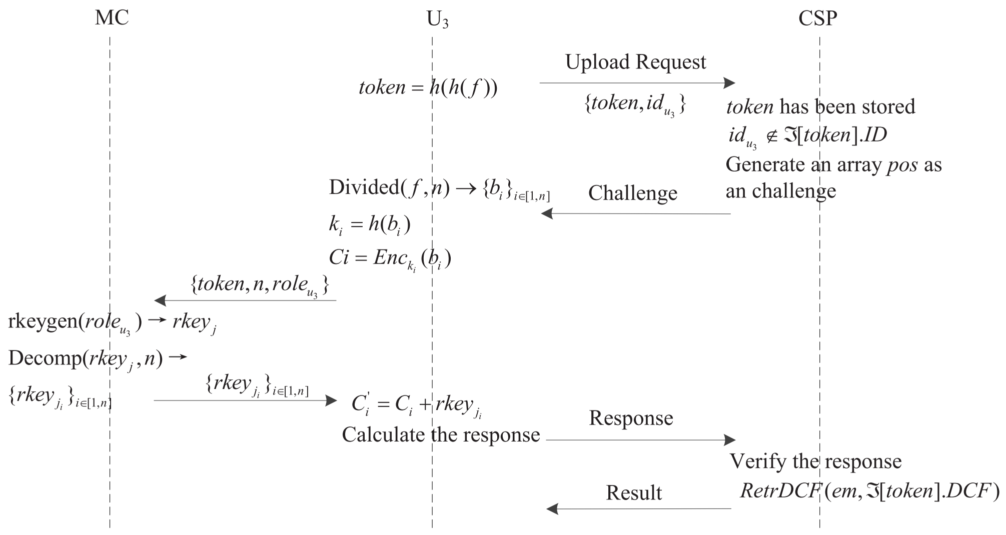
MC
U$_3$
CSP
$token = h(h(f))$
Upload Request
$\{token, id_{u_3}\}$
$token$ has been stored
$id_{u_3} \notin \Im[token].ID$
Generate an array pos as an challenge
Challenge
Divided$(f, n) \rightarrow \{b_i\}_{i\in[1,n]}$
$k_i = h(b_i)$
$Ci = Enc_{k_i}(b_i)$
$\{token, n, role_{u_3}\}$
rkeygen$(role_{u_3}) \rightarrow rkey_j$
Decomp$(rkey_j, n) \rightarrow$
$\{rkey_{j_i}\}_{i\in[1,n]}$
$\{rkey_{j_i}\}_{i\in[1,n]}$
$C_i' = C_i + rkey_{j_i}$
Calculate the response
Response
Verify the response
$RetrDCF(em, \Im[token].DCF)$
Result

FIGURE 6. Proof of ownership.

Algorithm 2 Proof of Ownership

```
input: $token, id$
output: $the\ result\ of\ verification$
 1: (@User): $token = h(h(f))$;
 2: uploads $\{token, id\}$ to CSP;
 3: (@CSP): checks $id\ ?\in \Im[token].ID$;
 4: if $id \in \Im[token].ID$ then
        return $\top$;
 5: else
 6:     generates the challenge;
 7:     sends $pos$ and $J$ to the user;
 8:     (@User): calculates the response;
 9:     Divided$(f, n) \rightarrow \{b_i\}_{i\in[1,n]}$;
10:     Encrypt$(\{b_i\}_{i\in[1,n]}) \rightarrow \{C_i\}_{i\in[1,n]}$;
11:     ReDecrypt
12:     $(\{C_i'\}_{i\in[1,n]}, \{rkey_i\}_{i\in[1,n]}) \rightarrow \{C_i\}_{i\in[1,n]}$;
13:     for $j \leftarrow 1$ to $J$ do
14:         $res[j] \leftarrow h(C'[pos[j]])$;
15:     end for
16:     sends $res$ to CSP;
17:     (@CSP): verifies the response;
18:     for $j \leftarrow 1$ to $J$ do
19:         $em \leftarrow PRF(res[j], pos[j])$;
20:         if $\neg RetrDCF(em, \Im[token].DCF)$ then
        return $\perp$;
21:         end if
22:     end for
23:     $\Im[token].ID \leftarrow \Im[token].ID \cup id$;
        return $\top$;
24: end if
```

protocol. The description of the above procedure is shown in *Algorithm 2*.

E. ROLE KEY UPDATE

MC manages RAT to update the role key. Through role re-encryption key updating and revoking, our system is able to achieve the dynamic updating of the authorized user's privilege.

The update operations contain the modification of role keys and the updating of RAT. For a RAT, the elements of leaf node are corresponding to the role information. Based on the properties of B+ tree, the modification of role keys only affects the nodes along the path from a wanted element of a leaf node to the root node on the RAT. Therefore, the modification of role keys is equal to search for a wanted element of the leaf node. The update operations include Modification, Deletion and Insertion, which mostly conform to the procedures of a standard B+ tree [25], [31].

MC wants to update the role key $rkey$, it firstly calls **rkeyUp** to obtain the updated role key $rkey'$, and sends the update request with a set of $\{token_i\}_{i\in[1,l]}$ to CSP. CSP retrieves the associative array based on the set of token, and returns the corresponding re-encryption ciphertext $\{C'_{token_i}\}_{i\in[1,l]}$ to MC. Upon receiving this information, MC calls **ReDecrypt** used $rkey$ to decrypt C', $C_{token_i} = C'_{token_i} - rkey$, then, calls **ReEncrypt** used $rkey'$ to obtain the updated re-encryption ciphertext C'', $C''_{token_i} = C_{token_i} + rkey'$. Finally, MC uploads a set of $\{C''_{token_i}\}_{i\in[1,l]}$ to CSP. Upon receiving the updated ciphertext, CSP retrieves the corresponding $\Im[token].DCF$, runs $DeleDCF$ and $InseDCF$ to update the DCF. All the processes are shown in *Algorithm 3*.

V. SECURITY ANALYSIS

In this section, we analyze the security of the proposed SRRS under the adversary model (defined in Section III.B) based on the standard model [29].

A. SECURITY OF AUTHORIZED DEDUPLICATION

In the standard model, we propose a security formulation for the authorized deduplication scheme.

Theorem 1: Suppose the expression: $\varepsilon_1 > \varepsilon_2 = 2\lambda + \Omega(\lambda)$ is satisfied. We define that h represents a collision-resistant

Algorithm 3 Role Key Update

```
input: $role_{u_i}$, $rkey'$
output: $\{\Im'[token_i].ENC\}_{i\in[1,l]}$, $\{\Im'[token_i].DCF\}_{i\in[1,l]}$
1: (@MC): searches the RAT to obtain $R[u_i].rolekey$ and $R[i].filelist$;
2: updates role key $rkey$ to $rkey'$;
3: sends the file list index $\{token_i\}_{i\in[1,l]}$ to CSP;
4: (@CSP): retrieves $\Im$;
5: sends the $\{\Im[token_i].ENC\}_{i\in[1,l]}$ to MC;
6: (@MC): updates the ciphertext;
7: for $k \leftarrow 1$ to $l$ do
8:     ReDecrypt $(C'_k, rkey) \rightarrow C_k$;
9:     ReEncrypt $(C_k, rkey') \rightarrow C''_k$;
10:    $\Im'[token_k].ENC \leftarrow C''$;
11: end for
        return $\{\Im'[token_i].ENC\}_{i\in[1,l]}$;
12: sends the $\{\Im'[token_i].ENC\}_{i\in[1,l]}$ to CSP;
13: (@CSP): updates the DCF;
14: for $k \leftarrow 1$ to $l$ do
15:     for $v \leftarrow 1$ to $n$ do
16:         $em \leftarrow PRF(h(C'_v), v)$;
17:         $DeleDCF : CBF \leftarrow \{em_{h_t}\}_{t\in[1,\gamma]}$;
18:         $em' \leftarrow PRF(h(C''_v), v)$;
19:         $InseDCF : CBF \leftarrow \{em'_{h_t}\}_{t\in[1,\gamma]}$;
20:     end for
21:     $\Im'[token_k].DCF \leftarrow DCF'$;
22: end for
        return $\{\Im'[token_i].DCF\}_{i\in[1,l]}$;
```

hash function, and E is an encryption scheme with semantic secure. The proposed authorized deduplication scheme is $(\varepsilon_1, \varepsilon_2)$-secure.

Proof: Any PPT adversary who attempts to attack our proposed scheme is defined as $\mathcal{A}_{SRRS}$, we define that $\mathcal{A}_1$ is a PPT adversary, who attempts to attack the encryption scheme E. $\mathcal{A}_1$ has obtained a re-encryption ciphertext $C' = E.ReEncrypt(rkey, C)$, $C = E.Encrypt(h(f), f)$, without any information about the role key $rkey$ and the input message f. We define $\varepsilon_1 > \varepsilon_2 \geq \lambda$, and the min-entropy of the message C is at least ε_1 bits, meanwhile, the adversary $\mathcal{A}_1$ is allowed to learn at most $(\varepsilon_1 - \varepsilon_2)$ bits information of message f from the challenger. Though querying the oracle $\mathcal{O}^f$, the adversary $\mathcal{A}_1$ can learn any output of $Func(f)$, where the PPT function $Func(f)$ is selected by $\mathcal{A}_1$.

Based on the above definitions, the adversary $\mathcal{A}_1$ simulates a security game G^{Sim}, which is described as follows:

Setup. $\mathcal{A}_1$ obtains the index value $token = h(h(f))$ and sends it to $\mathcal{A}_{SRRS}$.

Learning-I. $\mathcal{A}_1$ queries to the oracle $\mathcal{O}^f$ with $Func(f)$, and obtains the corresponding response $q = Func(f)$, where the bit-length of output q is required to be small than $(\varepsilon_1 - \varepsilon_2)$. Upon receiving q, $\mathcal{A}_1$ forwards the response to $\mathcal{A}_{SRRS}$.

Commit. Denote a subset of $\{x_i\}_{i\in[1,y]}$, $y \in [1, |f|]$ and $y + |q| \leq \varepsilon_1 - \varepsilon_2$. Though querying the oracle $\mathcal{O}^f$, $\mathcal{A}_1$ learns the value of the challenged subsequence $\beta = f[x_1]\|\cdots\|f[x_y]$, and sets $\beta_0 \leftarrow \beta$, $\beta_1 \leftarrow \{0,1\}^y$. Then, the challenger $\mathcal{A}_1$ integrates the above result and sends (β_0, β_1) to the adversary $\mathcal{A}_{SRRS}$.

Guess-I. Define $b^{Sim}_{\mathcal{A}^*_{SRRS}} \in \{0,1\}$ as the output of the extractor $\mathcal{A}^*_{SRRS}$.

Learning-II. In order to answer the queries made by $\mathcal{A}_{SRRS}$, $\mathcal{A}_1$ selects $\hat{rkey} \leftarrow rkeyGen(1^\lambda)$ randomly and independently, sets $\hat{C}' = ReEncrypt(C, \hat{rkey})$ and sends $\hat{C}'$ to $\mathcal{A}_{SRRS}$.

Guess-II. $\mathcal{A}_{SRRS}$ outputs a guess $b^{Sim}_{\mathcal{A}^*_{SRRS}} \in \{0,1\}$ of b.

Finally, the PPT adversary $\mathcal{A}_1$ outputs $\beta_{b^{Sim}_{\mathcal{A}^*_{SRRS}}} \in \{\beta_0, \beta_1\}$ and wins the game G^{Sim} if $\beta_{b^{Sim}_{\mathcal{A}^*_{SRRS}}} = \beta = f[x_1]\|\cdots\|f[x_y]$.

Though querying and learning from the oracle $\mathcal{O}^f$, the adversary $\mathcal{A}_1$ can obtain at most $(\lambda + \varepsilon_1 - \varepsilon_2)$ bits message about the unknown f. Therefore, the unknown f has at least $(\varepsilon_2 - \lambda) = \lambda + \Omega(\lambda)$ bits min-entropy.

Theorem 1 is proved, our proposed SRRS is $(\varepsilon_1, \varepsilon_2)$-secure.

B. SECURITY OF PROOF OF OWNERSHIP

Theorem 2: Let S_{min} is the least byte exchanging with a colliding user, a PPT adversary succeeds in a PoW for file f with negligible probability, our proposed proof of ownership is (S_{min})-secure.

Proof: We construct a PPT adversary $\mathcal{A}_2$ against the PoW protocol, who does not possess the entire file. p is the probability of $\mathcal{A}_2$ knowing a byte of the file f at a randomly chosen position, and g is the probability of $\mathcal{A}_2$ guessing correct when he doesn't possess the given byte.

We define that the event bl_i is $\mathcal{A}_2$ passing the protocol when he is given a *token*, the result happens in either of the following two cases: ① $\mathcal{A}_2$ gets a correct *token*; ② When DCF checks the element, a false positive occurs. We define the false positive of DCF is p_f. Through the above analysis, the probability of the event bl_i can be described as:

$$\begin{aligned} P(bl_i) &= P(bl_i \cap (token_i \cup \overline{token_i})) \\ &= P(bl_i \mid token_i)P(token_i) + P(bl_i \mid \overline{token_i})P(\overline{token_i}) \\ &= P(token_i) + p_f P(\overline{token_i}). \end{aligned} \tag{1}$$

The cloud server uses the *token* as a seed to generate the index with *PRF*. Therefore, we define that an event $token_i$ is $\mathcal{A}_2$ generating the bits of the i-th *token*, the event φ_i is $\mathcal{A}_2$ knowing the i-th *block*. According to the mathematics knowledge, the probability of $\mathcal{A}_2$ obtaining the correct *block* is g^B, and guessing the l-bit output of H_5 is 0.5^l. Obviously, the size of *token* is shorter than *block*, guessing the *token* is easier for $\mathcal{A}_2$. That is, $g^B \ll 0.5^l$. Therefore, the probability of the event $token_i$ can be described as:

$$\begin{aligned} P(token_i) &= P(token_i \cap (\varphi_i \cup \overline{\varphi_i})) \\ &= P(token_i \mid \varphi_i)P(\varphi_i) + P(token_i \mid \overline{\varphi_i})P(\overline{\varphi_i}) \\ &= p + (1-p)0.5^l. \end{aligned} \tag{2}$$

TABLE 2. Comparison with other schemes for efficiency.

Proposals	Communication overhead			Storage overhead	
	Upload message size	Download message size	Rekeying message size	Key size	Token size
Xu [29]	$S_C + S_K + S_T + S_{ID}$	$S_C + S_K + S_T$	-	S_K	S_T
Hur [32]	$S_C + S_K + S_T + S_{ID}$	$S_C + S_K + S_T$	$(x-y)log\frac{x}{x-y}S_k$	$(logx+1)S_K$	S_T
SRRS	$S_C + S_T + S_{ID}$	$S_C + S_{PoW} + S_T$	$S_K + y * S_C$	S_K	S_T

S_C:Size of a encryption file; S_K:Size of a key; S_T:Size of a $token$; S_{ID}:Size of an identifer; S_{PoW}:Size of exchanged data in PoW; x:Number of users; y:Number of ownership users owned the same file.

TABLE 3. Comparison with other schemes for computational cost.

Algorithms	Xu [29]	Ding [33]	Hur [32]	SRRS
Setup	-	$1 * Exp$	-	$-$
KeyGen	λ	$1 * BP$	$y * log_2 x$	$log_n m$
Enc	$SE + hash + XOR$	$2 * Exp + 2 * ModExp$	$2 * hash + SE + XOR$	$2 * hash + CE$
ReEnc	-	$1 * BP$	$2 * SE$	Con
KeyUp	-	$1 * Exp + 1 * BP$	$(x-y)log\frac{x}{x-y}$	$log_n m + y * Con$

Exp:Exponent operation on $\mathbb{G}_1$; BP:Bilinear pair; SE:Symmetric encryption; $ModExp$:Modular exponent operation; CE:Convergent encryption; Con:Concatenate operation; λ:Security parameter; n:Number of roles; m:Number of ownership users in a role group.

Then, we can obtain:

$$\begin{aligned} P(bl_i) &= p + (1-p)0.5^l + p_f(1-(p+(1-p)0.5^l)) \\ &= p + (1-p)(0.5^l + p_f(1-0.5^l)). \end{aligned} \quad (3)$$

The challenge contains J independent block positions, the probability of $\mathcal{A}_2$ passing the challenge is described as:

$$P(success) = P(bl_i)^J. \quad (4)$$

We set up a security parameter κ to derive a lower bound for J, that is $P(success) \leq 2^\kappa$, as:

$$J \geq \kappa ln2/(1-p)(1-(0.5^l + p_f(1-0.5^l))). \quad (5)$$

Therefore, the probability of $\mathcal{A}_2$ running a successful PoW protocol should be negligible under the security parameter κ, and the privacy information of users cannot be acknowledged by the CSP in the process of authorized deduplication.

We ensure that the adversary $\mathcal{A}_2$ obtains at least S_{min} bytes of the privacy information from the legitimate user, which can run a successful RSE-PoW protocol. According to the definitions of *token* and *block*, we know that *token* is shorter than *block*, therefore, we set the *token* length as:

$$l \geq S_{min}\frac{B}{F}. \quad (6)$$

For the performance goals, the communication bandwidth of our proposed scheme is related to the security parameters, we use the small exchanged file between user and CSP to achieve the security goal of authorized deduplication. Furthermore, the server memory overhead is also related to the security parameters and is independent of the uploaded file size, which achieves the goal of reducing server memory overhead. The role key is assigned by MC, the key size stored in client is independent of the file size, which achieves the goal of reducing the client storage overhead.

VI. PERFORMANCE EVALUATION

In this section, we give the performance analysis and evaluation of our SRRS.

A. COMPLEXITY ANALYSIS

We compare our proposed SRRS system with the related schemes in terms of the efficiency and computational complexity, and the result is summarized in Table 2 and Table 3.

For the comparison of the efficiency, we give the communication overhead and storage overhead of the related works. Upload message size refers to the communication overhead of the file upload phase between the users and CSP, download message size refers to the communication overhead of the file download phase and proof of ownership phase between the users and CSP, rekeying message size refers to the communication overhead of the key updating phase and ciphertext updating phase between the user (or MC) and CSP, key size and token size refer to the storage overhead that the user required to store respectively.

In the upload phase, both Xu's scheme [29] and Hur's scheme [32] exploit the randomized convergent encryption to obtain the ciphertext. In order to realize the decryption, the randomized key needs to be uploaded. In our SRRS system, the key is related to the role that managed by the MC, and it does not need to upload. In the download phase, our system performs proof of ownership to realize the authorized access, S_{PoW} is the size of exchanged information in our proposed PoW protocol. In the key updating and ciphertext updating phase, Hur's and our SRRS system support the rekeying, the communication overheads of the rekeying are $(x-y)log\frac{x}{x-y}S_k$ and $S_K + y * S_C$ respectively. In terms of the storage overhead, the user is required to store *logn* additional KEKs in Hur's scheme. In Xu's and our SRRS system, the user only needs to store the S_K. For the token size, the above schemes have the same storage overhead.

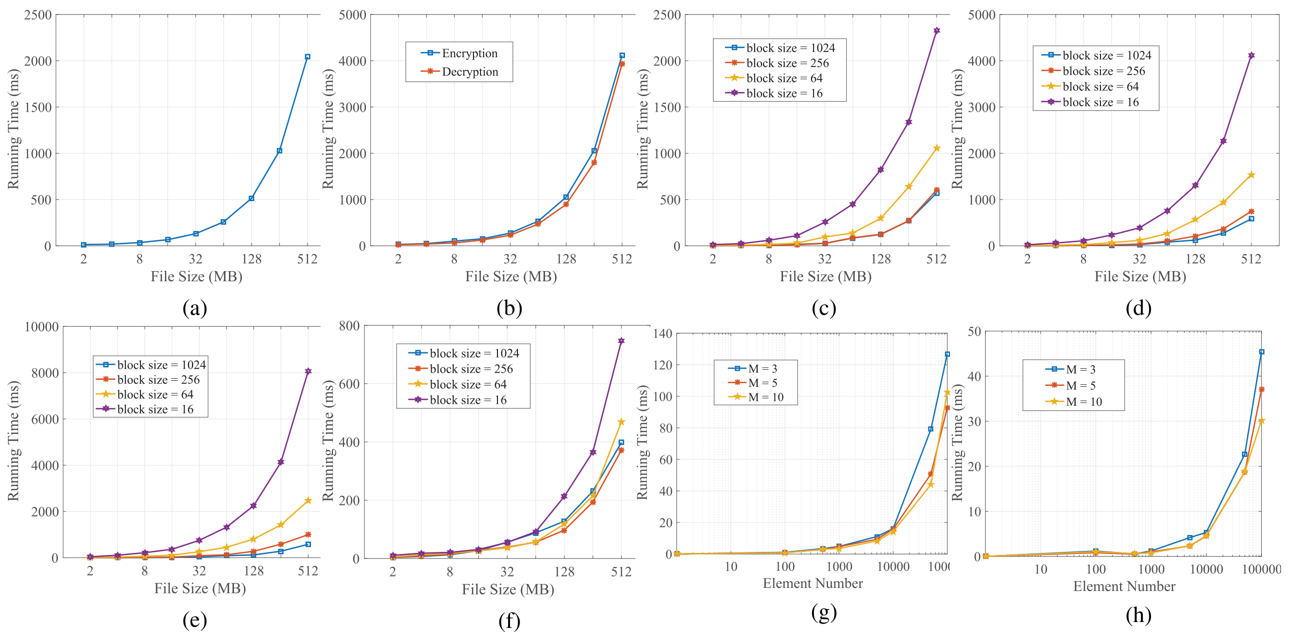

FIGURE 7. Performance analysis of our system. (a) The computational cost of generating file token. (b) The computational cost of encryption and decryption. (c) The computational cost of role re-encryption ($|rkey| = 256bit$). (d) The computational cost of role re-encryption ($|rkey| = 512bit$). (e) The computational cost of role re-encryption ($|rkey| = 1024bit$). (f) The computational cost of re-decryption. (g) The computational cost of RAT initialization. (h) The computational cost of RAT retrieval.

For the comparison of the computational complexity, we give the computational cost of different algorithms with the related works. Both Xu's scheme [29] and Hur's scheme [32] exploit the randomized convergent encryption, the key is selected by the file creator randomly, therefore, the computational cost of *KeyGen* are λ and $y * log_2 x$, and the computational cost of *Enc* are $SE + hash + XOR$ and $2 * hash + SE + XOR$ respectively. In Ding's scheme [33], the key is generated and managed by the third party, therefore, the computational cost of *Setup* is $1 * Exp$, and *KeyGen* is $1 * BP$. Our SRRS system exploits the convergent encryption and role re-encryption to obtain the re-encryption ciphertext, therefore, the computation cost of *KeyGen* is $log_n m$, and *Enc* is $2 * hash + CE$. With regard to the computational cost of *ReEnc* and *KeyUp*, Ding's scheme [33] uses the asymmetric encryption to generate the re-encryption ciphertext, Hur's scheme [32] uses symmetric encryption to generate the re-encryption ciphertext and key, our SRRS system uses the concatenation operation to generate the re-encryption ciphertext. Therefore, the computation costs of *ReEnc* in three schemes are $1 * BP$, $2 * SE$ and *Con*, the computation cost of *KeyUp* are $1 * Exp + 1 * BP$, $(x - y)log\frac{x}{x-y}$, $log_n m + y * Con$, respectively.

B. PERFORMANCE MEASUREMENTS

In this section, we conduct a serial of simulation experiments to evaluate the performance of the proposed scheme, the experiments are implemented on a PC with the following configurations: CPU: Intel(R) Core(TM) i7-6700 @ 3.40GHz 3.41GHz; RAM: 16.0GB; System type: 64-bit operating system, X64-based processor; OS: Ubuntu 12.04.4 LTS. We invoke OpenSSL library for cryptographic operations, and use Java language with Eclipse application program on Linux. Specifically, we employ AES-256 and SHA-256 for the symmetric encryption and all hash algorithms, respectively.

We measure the computational cost of the basic algorithms and protocols of our proposed scheme. We run 200 times for each algorithm and protocol to obtain the average value, and use nine files with different sizes from 2MB to 512MB.

The token is the unique identifier of a file, and the computational cost of the number of token is related to the file size. We test the performance of token generation with different file sizes from 2MB to 512MB, as shown in Figure 7 (a). We use the AES-256 to test the performance **Encrypt** and **Decrypt** algorithms with different file sizes, as shown in Figure 7 (b).

With the file size increasing, the computational cost of generating file token, symmetric encryption and decryption has significantly increased. When the file size is 512MB, the computational cost of generating token requires about 2.04s, **Encrypt** algorithm requires 4.11s to encrypt the original file into a ciphertext, and **Decrypt** algorithm requires 3.93s to decrypt the ciphertext to the original file.

We set the role key *rkey* as {256, 512, 1024} bits, set the block size as {1024, 256, 64, 16} bits and test the computational cost of **ReEncrypt** and **ReDecrypt** algorithms with different file sizes from 2MB to 512MB, as shown in Figure 7 (c) to Figure 7 (e).

The main computational cost of **ReEncrypt** algorithm is the runtime of concatenating the ciphertext blocks with the role key shares. When *rkey* is 1024 bits and the block size is

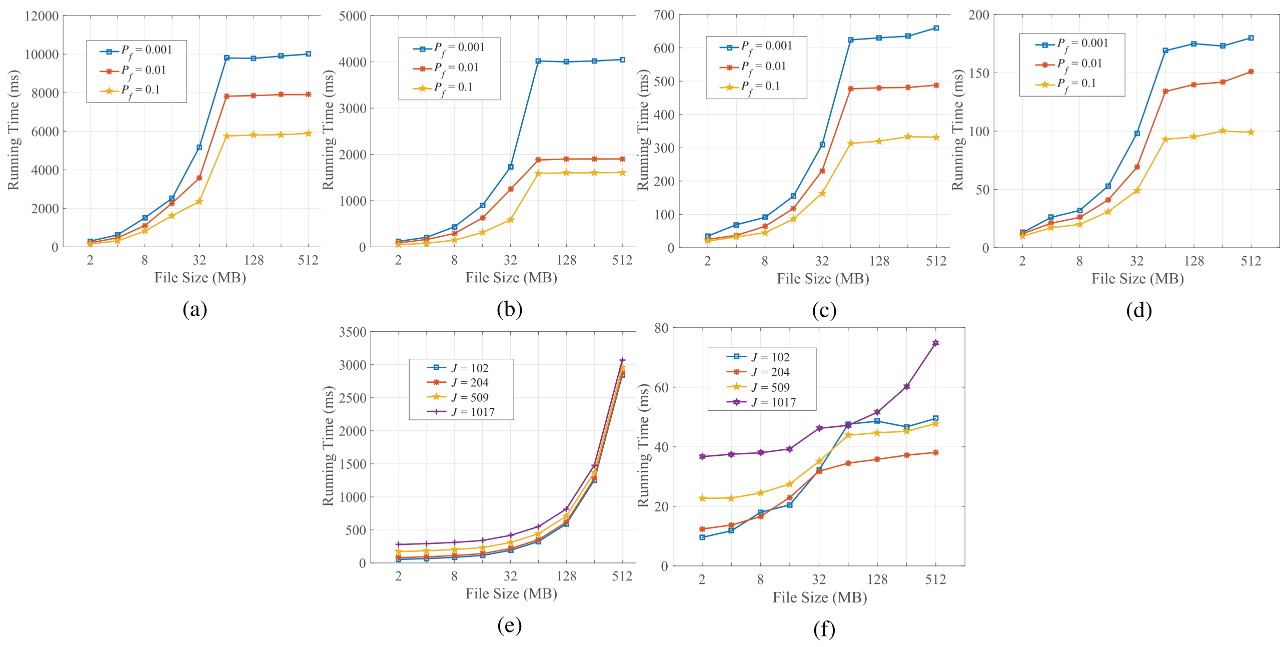

FIGURE 8. Performance analysis of our system. (a) The computational cost of DCF initialization ($|token| = 16bit$). (b) The computational cost of DCF initialization ($|token| = 64bit$). (c) The computational cost of DCF initialization ($|token| = 256bit$). (d) The computational cost of DCF initialization ($|token| = 1024bit$). (e) The computational cost of PoW challenge generation. (f) The computational cost of PoW verification.

1024 bits, the computational cost reaches maximum. When *rkey* is 16 bits and the block size is 256 bits, the computational cost uncertain reaches minimum, which is also affected by the number of blocks. When *rkey* is 1024 bits, the block size is 1024 bits, with file size of 512 MB, **`ReEncrypt`** algorithm requires 8.05s to re-encrypt the ciphertext. When *rkey* is 16 bits, the block size is 256 bits, with file size of 512 MB, **`ReEncrypt`** algorithm requires 5.71s to re-encrypt the ciphertext.

The main computational cost of **`ReDecrypt`** algorithm is the runtime of splitting the ciphertext blocks with the role key shares, where the size of role key has little effect on the runtime. Therefore, we test the performance of re-decryption with different block sizes and file sizes, as shown in Figure 7 (f). When *rkey* is 1024 bits, with file size of 512 MB, **`ReDecrypt`** algorithm requires 7.47s to re-decrypt the re-encryption ciphertext.

We set the order of RAT with the set of $\{3, 5, 10\}$, we test the computational cost of RAT initialization and search with different number of elements, as shown in Figure 7 (g) and Figure 7 (h). With the number of elements increasing, the different orders of RAT have an increasing impact on the runtime of RAT initialization and search. When the order of RAT is 3 and the number of elements is 10^5, the computational cost of RAT initialization and search is about 0.126s and 0.045s.

We set the security parameter κ as 66, S_{min} as $64MB$, the size of token l as in the set of $\{16, 64, 256, 1024\}$ bits, the probability p as in the set of $\{0.5, 0.75, 0.9, 0.95\}$ and the false positive rate of the DCF p_f as $\{0.001, 0.01, 0.1\}$. According to Equation (9), the PoW challenges J is set to $\{102, 204, 509, 1017\}$ based on the above values. And the size of blocks b is obtained, according to the values of l, S_{min}, the size of input files and Equation (10).

The main computational cost of DCF initialization is the runtime of *InitDCF* and *InseDCF*, which is related to the size of token and the false positive rate. When the file size reaches 64 MB, the inserted elements to the DCF reaches a fixed value, resulting in a decrease in curve growth. We test the computational cost of DCF initialization with different sizes of token and the false positive rates, as shown in Figure 8 (a) to Figure 8 (d). When the false positive rate p_f is 0.001 and token size is 16 bits, for file size of 512 MB, DCF initialization requires 9.89s. When the false positive rate p_f is 0.1 and token size is 1024 bits, for file size of 512 MB, DCF initialization requires 0.09s.

The generation of PoW challenge is affected by the value of J, therefore, we test the computational cost of challenge generation with different value J based on the above analysis, as shown in Figure 8 (e). When value of J is 1017 and file size is 512 MB, the generation of PoW challenge requires 3.07s. After receiving the response, CSP verifies the correct of the responses *res* with different J challenges using the DCF retrieval. Finally, we test the computational cost of PoW verification with the different value of J, as shown in Figure 8 (f). when J is 1017, the PoW verification is about 0.074s, which is highly efficient.

VII. RELATED WORKS

In order to tackle the problem that the unauthorized users can access the user information only by supplying the hash value of the file. Halevi *et al.* [34] proposed the proof of ownership (PoW), which is an interaction protocol between client side

TABLE 4. Comparison with other schemes.

Proposals	Main technology	Third parties	Authorized deduplication	PoW	Key update
Halevi [34]	MHT + Erasure code	-	*N*	*Y*	*N*
González [35]	CE + RanSam	-	*N*	*Y*	*N*
Li [18]	CE + Identification protocol	Private Cloud	*Y*	*Y*	*N*
Xu [29]	SE	Cloud Storage Server	*Y*	*N*	*Y*
Tang [36]	CP-ABE	The authority	*Y*	*N*	*N*
González [37]	SE + RanSam	ACS	*Y*	*Y*	*N*
Xiong [4]	SE + RanSam	RCS	*Y*	*Y*	*Y*
Ding [33]	HomEnc + Proxy Re-enc	AP	*Y*	*Y*	*Y*
Hur [32]	RCE + Group key distribution	-	*Y*	*Y*	*Y*
Our proposal	CE + Role Re-enc	MC	*Y*	*Y*	*Y*

MHT: Merkle Hash trees; SE: Symmetric encryption; CE: Convergent Encryption; RCE: Randomized Convergent Encryption; HomEnc: Homomorphic encryption; RanSam: Random sampling; Proxy Re-enc: Proxy Re-encryption; Role Re-enc: Role Re-encryption.

and server side to verify the ownership of that client. In [34], the client and server create a Merkle Hash Tree (MHT) based on the source file, and use a challenge-response model to verify the correct of MHT path provided by the client. Ng *et al.* [38] proposed a private data deduplication in data storage, where a client held a private data proves to a server stored a summary string of the data that he/she is the owner of that data without revealing further information to the server. Xu *et al.* [29] proposed a cryptographic primitive to enhance the security of client-side deduplication in the bounded leakage setting where a certain amount of efficiently-extractable information about file F is leaked. In order to reduce the computational cost and improve the efficiency, Pietro and Sorniotti [39] proposed a s-PoW scheme, which requests some particular random bits of the file from the verified client. Blasco *et al.* [40] proposed a bf-PoW scheme based on the bloom filter, which requires the certain tokens from the verified client to achieve the proof of ownership efficiently. Through the security analysis, a wide range of benchmark tests and comparison of the existing schemes, the proposed scheme greatly reduces the cost of both the client and the cloud server. Gonzalez-Manzano and Orfila [35] proposed a ce-PoW scheme based on the convergent encryption (CE) [13], which requires neither trusted third party nor the complex key management. The cloud server stores a four-tuples containing the encrypted blocks, challenge, response, and a list of client identifiers. The experimental evaluation shows the efficiency and feasibility of the proposed scheme.

In order to solve the unauthorized access in data deduplication, Puzio *et al.* [41] proposed a secure and efficient storage system called ClouDedup, which is based on the CE algorithm and access control mechanism, and achieves block-level data deduplication. Li *et al.* [42] proposed a CDStore scheme, which is based on the CE algorithm and convergent dispersal mechanism to achieve secure data deduplication, and saves nearly 70% of the storage overhead. Tang *et al.* [36] proposed a secure data deduplication scheme based on CP-ABE algorithm [43] in cloud. With a recursive algorithm and an additional randomness adding method, it can achieve the duplication check and secure deduplication. Li *et al.* [18] proposed a novel data deduplication scheme and gave a concept of hybrid cloud environment, where the generation of encryption keys is related to the corresponding privilege, the private cloud server is responsible for management and storage of the user's keys, and the public cloud server stores the ciphertext and performs the data deduplication. Xiong *et al.* [4] proposed a role symmetric encryption (RSE) algorithm, and constructed a role symmetric encryption PoW (RSE-PoW) scheme, which establishes a hierarchical role tree and utilizes the role access control mechanism to achieve authorized deduplication. Gonzalez-Manzano *et al.* [37] proposed an ase-PoW scheme in hierarchical environment, which uses a lightweight access control procedure to resist against the content guessing attack, where the encryption key is linked to the owner's attributes and calculated by the symmetric recursive encryption.

However, the above data deduplication schemes do not take into account the key updating and user revocation. Kwon *et al.* [44] proposed a new deduplication scheme with multimedia data, which is based on randomized convergent encryption and privilege-based encryption to achieve authorized deduplication and user revocation. Li *et al.* [45] and Qin *et al.* [46] proposed a rekeying-aware encrypted deduplication (REED) system to achieve data deduplication and dynamic access control. Hur *et al.* [32] proposed a novel data deduplication scheme for the server-side, which uses the randomized convergent encryption algorithm and ownership group key distribution technique to achieve the authorized access and support security deduplication with ownership changes dynamically. Ding *et al.* [33] proposed a secure encrypted data deduplication scheme, which exploits the homomorphic encryption algorithm to achieve security data deduplication [47], and supports ownership check and user revocation. However, the above schemes exploit the homomorphic encryption and proxy re-encryption with high computation cost.

To present the novelties of our system, a comparison of related works is depicted in Table 4.

VIII. CONCLUSION

To process the massive cloud data efficiently, the cloud service providers widely perform data deduplication to reduce the occupation of storage space and the bandwidth consumption. In order to prevent privacy data leakage, achieve authorized deduplication and satisfy dynamic privilege updating and revoking, we proposed a novel secure role re-encryption system with authorized deduplication in cloud environment. In our proposed system, we firstly exploited the convergent encryption algorithm to prevent privacy data leakage and used the role re-encryption algorithm to achieve authorized deduplication efficiently. Specifically, we created the role authorized tree to manage the user's roles and the corresponding role keys, and introduced the management center to reduce the computation cost and management overhead of the client, and implement the dynamic updating of the authorized user's privilege. Furthermore, We exploit the dynamic count filters (DCF) to achieve the data updating and improve the retrieval of ownership verifying efficiency. Finally, the security analysis demonstrates the security of our proposed scheme, and the performance evaluation makes it clear that the proposed scheme is effective and efficient.

REFERENCES

[1] X. Liu, R. H. Deng, K.-K. R. Choo, and J. Weng, "An efficient privacy-preserving outsourced calculation toolkit with multiple keys," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 11, pp. 2401–2414, Nov. 2016.

[2] H. Xiong, H. Zhang, and J. Sun, "Attribute-based privacy-preserving data sharing for dynamic groups in cloud computing," *IEEE Syst. J.*, to be published.

[3] D. Wu, H. Shi, H. Wang, R. Wang, and H. Fang, "A feature-based learning system for Internet of Things applications," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1928–1937, Apr. 2019.

[4] J. Xiong, Y. Zhang, X. Li, M. Lin, Z. Yao, and G. Liu, "RSE-PoW: A role symmetric encryption pow scheme with authorized deduplication for multimedia data," *Mobile Netw. Appl.*, vol. 23, no. 3, pp. 650–663, 2018.

[5] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, Sep. 2016.

[6] J. Li, C. Qin, P. P. C. Lee, and X. Zhang, "Information leakage in encrypted deduplication via frequency analysis," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2017, pp. 1–12.

[7] J. Li, X. Chen, S. S. M. Chow, Q. Huang, D. S. Wong, and Z. Liu, "Multi-authority fine-grained access control with accountability and its application in cloud," *J. Netw. Comput. Appl.*, vol. 112, pp. 89–96, Jun. 2018.

[8] D. Wu, Q. Liu, H. Wang, Q. Yang, and R. Wang, "Cache less for more: Exploiting cooperative video caching and delivery in D2D communications," *IEEE Trans. Multimedia*, to be published.

[9] H. Xiong, Q. Mei, and Y. Zhao, "Efficient and provably secure certificateless parallel key-insulated signature without pairing for IIoT environments," *IEEE Syst. J.*, to be published.

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

[11] J. Paulo and J. Pereira, "A survey and classification of storage deduplication systems," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1–30, 2014.

[12] J. Xiong, Y. Zhang, L. Lin, J. Shen, X. Li, and M. Lin, "ms-PoSW: A multi-server aided proof of shared ownership scheme for secure deduplication in cloud," *Concurrency Comput., Pract. Exper.*, Aug. 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4252. doi: 10.1002/cpe.4252.

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

[14] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2013, pp. 296–312.

[15] 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.

[16] 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.*, Oct. 2015, pp. 874–885.

[17] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Trans. Storage*, vol. 7, no. 4, 2012, Art. no. 14.

[18] J. Li, Y. K. Li, X. Chen, P. P. C. 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.

[19] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. S. Shen, "Providing task allocation and secure deduplication for mobile crowdsensing via fog computing," *IEEE Trans. Dependable Secure Comput.*, to be published.

[20] Y. Zhang, X. Chen, J. Li, D. S. Wong, H. Li, and I. You, "Ensuring attribute privacy protection and fast decryption for outsourced data security in mobile cloud computing," *Inf. Sci.*, vol. 379, pp. 42–61, Feb. 2017.

[21] J. Xiong, J. Ren, L. Chen, Z. Yao, M. Lin, D. Wu, and B. Niu, "Enhancing privacy and availability for data clustering in intelligent electrical service of IoT," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1530–1540, Apr. 2019.

[22] K.-H. Yeh, "A secure transaction scheme with certificateless cryptographic primitives for IoT-based mobile payments," *IEEE Syst. J.*, vol. 12, no. 2, pp. 2027–2038, Jun. 2018.

[23] C.-M. Chen, B. Xiang, Y. Liu, and K.-H. Wang, "A secure authentication protocol for Internet of vehicles," *IEEE Access*, vol. 7, pp. 12047–12057, 2019.

[24] Y. Zhang, R. H. Deng, G. Han, and D. Zheng, "Secure smart health with privacy-aware aggregate authentication and access control in Internet of Things," *J. Netw. Comput. Appl.*, vol. 123, no. 12, pp. 89–100, Dec. 2018.

[25] Z. Mo, Y. Zhou, and S. Chen, "A dynamic Proof of Retrievability (PoR) scheme with O(logn) complexity," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 912–916.

[26] 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.*, Jul. 2002, pp. 617–624.

[27] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero, and J.-L. Larriba-Pey, "Dynamic count filters," *ACM SIGMOD Rec.*, vol. 35, no. 1, pp. 26–32, 2006.

[28] J. Li, X. Chen, M. Li, J. Li, P. P. C. 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.

[29] 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.

[30] J. Xu and J. Zhou, "Leakage resilient proofs of ownership in cloud storage, revisited," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Cham, Switzerland: Springer, 2014, pp. 97–115.

[31] C. Guan, K. Ren, F. Zhang, F. Kerschbaum, and J. Yu, "Symmetric-key based proofs of retrievability supporting public verification," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2015, pp. 203–223.

[32] J. Hur, D. Koo, Y. Shin, and K. Kang, "Secure data deduplication with dynamic ownership management in cloud storage," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 3113–3125, Nov. 2016.

[33] W. Ding, Z. Yan, and R. H. Deng, "Secure encrypted data deduplication with ownership proof and user revocation," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.* Cham, Switzerland: Springer, 2017, pp. 297–312.

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

[35] L. González-Manzano and A. Orfila, "An efficient confidentiality-preserving proof of ownership for deduplication," *J. Netw. Comput. Appl.*, vol. 50, pp. 49–59, Apr. 2015.

[36] H. Tang, Y. Cui, C. Guan, J. Wu, J. Weng, and K. Ren, "Enabling ciphertext deduplication for secure cloud storage and access control," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 59–70.

[37] L. González-Manzano, J. M. de Fuentes, and K. K. R. Choo, "ase-PoW: A proof of ownership mechanism for cloud deduplication in hierarchical environments," in *Proc. 12th EAI Int. Conf. Secur. Privacy Commun. Netw.*, 2016, pp. 412–428.

[38] W. K. Ng, Y. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage," in *Proc. 27th Annu. ACM Symp. Appl. Comput.*, 2012, pp. 441–446.

[39] 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.
[40] 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. (CNS)*, Oct. 2014, pp. 481–489.
[41] P. Puzio, R. Molva, M. Öen, and S. Loureiro, "ClouDedup: Secure deduplication with encrypted data for cloud storage," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, vol. 1, Dec. 2013, pp. 363–370.
[42] M. Li, C. Qin, and P. P. C. Lee, "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *Proc. USENIX Tech. Conf.*, 2015, pp. 111–124.
[43] Q. Li, J. Ma, R. Li, X. Liu, J. Xiong, and D. Chen, "Secure, efficient and revocable multi-authority access control system in cloud storage," *Comput. Secur.*, vol. 59, pp. 45–59, Jun. 2016.
[44] H. Kwon, C. Hahn, D. Kim, and J. Hur, "Secure deduplication for multimedia data with user revocation in cloud storage," *Multimedia Tools Appl.*, vol. 76, no. 4, pp. 5889–5903, 2017.
[45] J. Li, C. Qin, P. P. C. Lee, and J. Li, "Rekeying for encrypted deduplication storage," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun./Jul. 2016, pp. 618–629.
[46] C. Qin, J. Li, and P. P. C. Lee, "The design and implementation of a rekeying-aware encrypted deduplication storage system," *ACM Trans. Storage*, vol. 13, no. 1, 2016, Art. no. 9.
[47] X. Liu, R. H. Deng, W. Ding, R. Lu, and B. Qin, "Privacy-preserving outsourced calculation on floating point numbers," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 11, pp. 2513–2527, Nov. 2016.

JINBO XIONG (M'13) received the M.S. degree in communication and information systems from the Chongqing University of Posts and Telecommunications, China, in 2006, and the Ph.D. degree in computer system architecture from Xidian University, China, in 2013. He is currently a Visiting Scholar with the University of North Texas, USA, and an Associate Professor with the Fujian Provincial Key Laboratory of Network Security and Cryptology and the College of Mathematics and Informatics, Fujian Normal University. He has published more than 40 publications and one monograph. He holds eight patents in these fields. His research interests include cloud data security, privacy protection, and the mobile Internet security.

YUANYUAN ZHANG received the B.S. and M.S. degrees in software engineering from Fujian Normal University, Fujian, China, in 2015 and 2018, respectively. She is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. She was a Research Assistant with the School of Information System, Singapore Management University, Singapore, from 2017 to 2018. Her current research interests include mobile crowd sensing, and privacy preserving in cloud computing and blockchain technology.

SHAOHUA TANG (M'99) received the B.Sc. and M.Sc. degrees in applied mathematics and the Ph.D. degree in communication and information system from the South China University of Technology, China, in 1991, 1994, and 1998, respectively. He was a Visiting Scholar with North Carolina State University, USA, and a Visiting Professor with the University of Cincinnati, USA. He has been a Full Professor with the School of Computer Science and Engineering, South China University of Technology, since 2004. He has authored or coauthored over 100 technical papers in journals and conference proceedings. His current research interests include information security, data security, and privacy preserving in cloud computing and big data.

XIMENG LIU (S'13–M'16) received the B.Sc. degree in electronic engineering and the Ph.D. degree in cryptography from Xidian University, Xi'an, China, in 2010 and 2015, respectively. He is currently a Research Fellow with the School of Information System, Singapore Management University, Singapore, and a Qishan Scholar with the College of Mathematics and Computer Science, Fuzhou University. His research interests include cloud security, applied cryptography, and big data security.

ZHIQIANG YAO received the Ph.D. degree in computer system architecture from Xidian University, China, in 2014. He is currently a Professor with Fujian Normal University. He has published more than 80 research papers. He holds ten patents. His research interests include security in cloud computing and multimedia security.