IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 19 February 2023, accepted 26 February 2023, date of publication 2 March 2023, date of current version 24 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3251370

RESEARCH ARTICLE

Data Secure De-Duplication and Recovery Based on Public Key Encryption With Keyword Search

LE LI[1], DONG ZHENG[1,2], HAOYU ZHANG[1], AND BAODONG QIN[1]
[1]School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an 710121, China
[2]School of Computer, Qinghai Normal University, Xining 810008, China

Corresponding author: Baodong Qin (qinbaodong@xupt.edu.cn)

This work was supported in part by the Basic Research Program of Qinghai Province under Grant 2020-ZJ-701, and in part by the National Natural Science Foundation of China under Grant 62072207.

ABSTRACT In the current era of information explosion, users' demand for data storage is increasing, and data on the cloud has become the first choice of users and enterprises. Cloud storage facilitates users to backup and share data, effectively reducing users' storage expenses. As the duplicate data of different users are stored multiple times, leading to a sudden decrease in storage utilization of cloud servers. Data stored in plaintext form can directly remove duplicate data, while cloud servers are semi-trusted and usually need to store data after encryption to protect user privacy. In this paper, we focus on how to achieve secure de-duplication and recover data in ciphertext for different users, and determine whether the indexes of public key searchable encryption and the matching relationship of trapdoor are equal in ciphertext to achieve secure de-duplication. For the duplicate file, the data user's re-encryption key about the file is appended to the ciphertext chain table of the stored copy. The cloud server uses the re-encryption key to generate the specified transformed ciphertext, and the data user decrypts the transformed ciphertext by its private key to recover the file. The proposed scheme is secure and efficient through security analysis and experimental simulation analysis.

INDEX TERMS PEKS, secure de-duplication, proxy re-encryption, data recovery.

I. INTRODUCTION

As a major service provided by cloud computing technology, cloud storage enables users to backup and share data easily and quickly, which can efficiently reduce users' storage expenses and improve work efficiency. With the increasing maturity of cloud computing technology. There are many Cloud Service Providers (CSP) in the market, such as Baidu Cloud, Amazon Cloud, and other famous CSPs. Users will upload and store their confidential data to the data storage center of the cloud server, which is managed and maintained by the CSP, but with this comes the frequent occurrence of cloud computing security issues. For enterprises or individual users will be personal files, business contracts, user transaction records, environmental geographic data, and other susceptible data stored in the cloud server. However, user privacy leaks and sensitive data leaks have emerged, and even more, there are CSP through the sale of user data to achieve corporate profits. The issue of data security in cloud storage deserves widespread attention.

Big data and cloud computing are developing rapidly, with an explosion of data from users around the world, resulting in a dramatic increase in demand for cloud servers. An effective solution to the need for storage of massive amounts of data will be deduplicate data. For plaintext data, the equality test can be achieved by direct comparison, while user data involves user's personal privacy, and uploading or storing it in plaintext form to cloud servers can cause user privacy leakage. Encrypting data can protect user privacy effectively. In practical scenarios, different users use different keys for encrypting files, and there are random parameters in the encryption, then the ciphertext generated from the same file is different. By directly comparing ciphertexts, we cannot determine the duplicity of files, and cloud servers will store

The associate editor coordinating the review of this manuscript and approving it for publication was Tyson Brooks.

multiple encrypted copies of the same file, which will put huge storage pressure on cloud servers. Therefore, there is an urgent need to design a secure de-duplication scheme for encrypted data with different keys in multi-user scenarios.

Currently, convergent encryption [1] is widely used to construct secure data de-duplication systems, but convergent encryption also faces various dangers such as data leakage, faking attacks and chosen-plaintext attacks [2], [3], [4]. Since the encryption key used in convergent encryption is generated by the hash value of user's data file, multiple files of the same user will generate multiple different keys, thus causing a key management problem [5]. The operations of encryption and de-duplication of data affect each other. Encrypting data with the same key by different users will generate the same ciphertext. Secure data de-duplication is achieved by directly comparing ciphertexts with each other, but this will cause the problem of key management. If different users use different keys to encrypt data, the key management problem can be effectively reduced, but it is difficult to achieve equality test. Therefore, how different users can encrypt data with the same key without communicating with each other, thus producing the same ciphertext after encrypting the same data, and how users can recover their data are the main research directions of this paper.

In this paper, Public Key Encryption with Keyword Search (PEKS) is used to detect file duplicates by matching keywords with trapdoors, and Proxy Re-encryption (PRE) is used for data recovery [6]. The scheme is mainly divided into data de-duplication and data recovery. For the data de-duplication, the data owner needs to upload the file ciphertext, file tag, and re-encryption key to the cloud server, the file tag points to the file ciphertext, and the re-encryption key is stored in the corresponding ciphertext chain table. When the test result is True, it means that the file is already stored in the server. The data user does not need to upload the ciphertext again but only needs to upload the re-encryption key of the specified file to the corresponding ciphertext chain, which can effectively reduce the storage overhead of the cloud server. When the test result is False, it means that the file is not stored in the server, and the data owner needs to upload the ciphertext, file tag, and re-encryption key. Regarding data recovery, users only need to store the user key locally, not the file key for each file. The user key can be generated only locally without introducing a key generation center (KGC) to avoid key substitution attacks, malicious KGC attacks [7]. The user initiates a request to the cloud server to obtain the file, and the cloud server uses the re-encryption key of the user in the ciphertext chain table to re-encrypt and generates the transformed ciphertext. The transformed ciphertext is sent to the user, and the user can decrypt the file using his private key.

A. RELATED WORKS

The continuous development of cloud storage technology has brought new opportunities to many industries, and data on the cloud has become the immediate need for the digital transformation of traditional industries. How to effectively improve the space utilization of cloud storage is one of the problems that cloud storage needs to solve urgently. A large of duplicate data exists in the massive data being stored on the cloud server many times. The main ideas to solve the problem of deduplicated data storage are to improve the compression rate of stored files and to secure data de-duplication. This paper focuses on how to achieve secure data de-duplication in a multi-user environment.

Secure data de-duplication can be classified into file-level de-duplication and block-level de-duplication according to the granularity of de-duplication, client de-duplication and server de-duplication according to the de-duplication entity. In this paper, we focus on file-level secure de-duplication on the server. To protect the privacy of users, server de-duplication requires users to encrypt data files before uploading them to cloud servers. Doucear et al. [1] proposed convergent encryption, which can effectively balance data de-duplication and data encryption to achieve secure de-duplication in the ciphertext. It computes the hash value of files as the key of encrypted files.The same file will generate the same key, and encrypting the same file with the same key will generate the same ciphertext, thus realizing the direct comparison of the duplicity of files in the ciphertext state. Through this mechanism, we can see that the key generated using the file does not have randomness, and each file will generate a key, which will lead to the problem of key management. Bellare et al. [8] designed the variant algorithms HCE1, HCE2, and HCE3 of convergent encryption by analyzing the security of convergent encryption [1] to improve the security and efficiency of convergent encryption. Convergent encryption uses the file hash value as the encryption key and determines directly whether the file is duplicated by the ciphertext. Message Locking Encryption (MLE) is a further improvement to convergent encryption. MLE generates a tag for the file for de-duplication. MLE does not rely exclusively on the file hash to generate the file encryption key. The encryption key is generated after mapping the file by a deterministic function, which is not resistant to brute force attacks on predictable information. The encryption key and the tag are independent and they are not related in any way. To solve the problem, Keelveedh et al. [9] also proposed DupLESS server-assisted secure data de-duplication scheme, which effectively improves the randomness of deterministic ciphertexts. Abadi et al. [10] constructed MLE2 for lock-dependent messages based on MLE to improve the security of data de-duplication. Liu et al. [11] constructed a secure data de-duplication system based on a key exchange protocol without relying on an additional server, but the system has a large communication overhead and computational overhead and requires most users to be online at the same time. Puzio et al. [12] proposed PerfectDedup to perform secure de-duplication based on the popularity of data, combined with the property of perfect hashing to ensure the confidentiality of data. Li et al. [13] proposed a rekeying-aware encrypted de-duplication storage

system, where the data owner only needs to re-encrypt part of the message using convergent all-or-nothing transform (CAONT) to achieve secure de-duplication and effectively reduce the computational overhead of the system. Li et al. [14] proposed CDStore, an enhanced secret sharing scheme based on convergent encryption which takes deterministic hash values as the input of secret sharing and supports de-duplication. Tang et al. [15] proposed a secure de-duplication scheme based on threshold re-encryption, which can resist side-channel attacks while effectively reducing computational overhead. Gao et al. [16] proposed a secure de-duplication scheme without trusted third parties, with hierarchical encryption based on prevalence and privacy. Kan et al. [17] proposed an identity-based proxy re-encryption scheme to achieve secure data de-duplication and recovery by combining data de-duplication and user access privileges to achieve a complete de-duplication. Yuan et al. [18] found that REED [14] has a stub-reserved attack problem and constructed a new secure de-duplication algorithm using CAONT and Bloom filter to resist stub-reserved attack.

PRE was first proposed by Blaze et al. [19], PRE enables data sharing without revealing the data owner's key. Using PRE, a proxy server can transform the ciphertext encrypted using the data owner's key to generate a transformed ciphertext that can decrypt by the data user's key, thus protecting the data owner's key while enabling data sharing. Lu and Li [20] proposed a pairing-free proxy re-encryption scheme that can meet the application requirements of devices with limited computing power. According to the practical application scenarios, the PRE scheme applicable to the IoT, cloud computing, and other [21], [22], [23] scenarios is proposed. The application of electronic medical records in medical institutions has problems such as information leakage, and Liu et al. [24]. used proxy re-encryption and sequential multi-signature to solve the problem.

PEKS was first proposed by Boneh et al. [6] to support the server to search the ciphertext without knowing the plaintext message. Fang et al. [25] constructed PEKS based on the standard security model to resist keyword guessing attacks. Lu et al. [26] that certificate-based searchable encryption not only resists keyword guessing attacks, but also supports implicit authentication, no secure channel. Guo and Yau [27] proposed PEKS that can satisfy the Indistinguishability of trapdoors. Qin et al. [28] introduced an improved CI model that enables public key authenticated encryption with keyword retrieval to resist fully chosen keyword to ciphertext-keyword attacks in a multi-user environment. Zhang et al. [29] first proposed public key encryption with bidirectional keyword search, which has practical applications in various scenarios such as email systems. Chen et al. [30], inspired by the Diffie-Hellman Exchange algorithm constructed a dual-server public-key authenticated encryption with keyword search scheme based on Chen et al. [31], where the system requires not only dual-server public keys for the generation of keyword ciphertexts and trapdoors but also the public keys between users to ensure that only authenticated users can search the ciphertext. Lu et al. [32] devise a lightweight public key authenticated encryption with keyword search scheme, which is suitable for the resource-constrained mobile devices.

B. OUR CONTRIBUTION

In this paper, we construct a secure data de-duplication and recovery scheme based on public key searchable encryption by combining public key searchable encryption and proxy re-encryption. The contributions of this paper are specified as follows:

1) A secure data de-duplication scheme based on public encryption with keyword search is constructed to realize secure data de-duplication in a multi-user environment, which can effectively save the storage space of cloud servers. The scheme in this paper is a server de-duplication, which can achieve secure de-duplication without users online, and its application scenario is more flexible.
2) This paper uses proxy re-encryption to achieve user data recovery. The server uses the re-encryption key stored in the ciphertext chain table for re-encryption, and the user can decrypt and recover the files using only his private key, so that there is no need to save each file key, which can effectively reduce the key management problem.
3) The only entities involved in the interaction are users and cloud servers, and no KGC is introduced, so that key substitution attacks and malicious KGC attacks can be effectively avoided. Meanwhile, for malicious servers that can obtain file tags and file ciphertexts of arbitrary files, it can achieve one-wayness under the chosen file attack.
4) Massive data are storaged in the cloud server, so the efficiency of equality test in the overall de-duplication process is critical, through experimental simulation analysis, for the storage of 5000 files in the database, the realization of safe de-duplication takes 42.86s, about one-third of the time consumed by the paper [17].

C. ORGANIZATION

The rest of the paper is organized as follows. Section II presents the algorithm and specific design. Section III analyzes the security of the scheme. Section IV simulates the scheme to analyze its performance. Finally, a conclusion is presented in Section V.

II. ALGORITHM AND SCHEME DESIGN

A. SYSTEM MODEL

As shown in Fig 1, the entities involved in this scheme include data user, CSP, and these entities are described below.

Data user: The data user encrypts the file and uploads it to the CSP to ensure the user's privacy. The user is distinguished

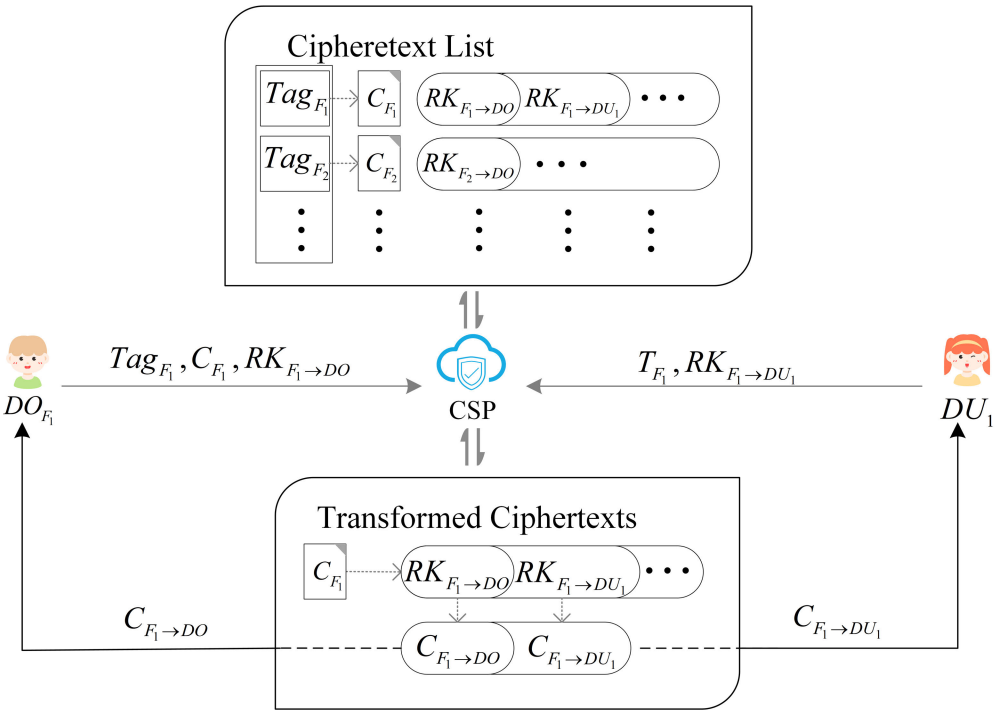

FIGURE 1. System model.

into data owner (DO) and data user (DU) according to the status at the time of upload, and the DO needs to upload the ciphertext, file tag, and re-encryption key, while the DU only needs to upload the re-encryption key about the file. When the data user recovers the file, the CSP uses the corresponding re-encryption key in the ciphertext chain table to generate the transformed ciphertext. The user can use his private key to decrypt the transformed ciphertext to recover the file.

CSP: store the ciphertext and file tags uploaded by users, and establish file tag index. When the user uploads a test tag, a matching operation is performed to determine whether it is a duplicate file. For non-duplicate files, the user needs to store the ciphertext, file tags, and the user's re-encryption key about the file in the cloud server. For duplicate files, users only need to store their re-encryption keys for the files in the corresponding ciphertext chain table. When the user sends a file recovery request, the CSP uses the corresponding re-encryption key in the ciphertext chain table to generate a transformed ciphertext to send to the user.

B. ALGORITHM DESIGN

The scheme in this paper involves only the user and the cloud server. The overall workflow is divided into two phases of data de-duplication and data recovery, containing a total of 10 algorithms, which are described as follows.

Setup $(k) \to params$: Given a security parameter k, return a public parameter $params = \{q, g, e, Z, G_1, G_2, H_1, H_2, H_3\}$, G_1 and G_2 are two cyclic groups with prime order q and g is a generator G_1. It defines a bilinear map $e : G_1 \times G_1 \to G_2$ and let $Z = e(g, g)$. In addition, the following defines three hash functions $H_1 : \{0, 1\}^* \to Z_q^*$,$H_2 : \{0, 1\}^* \to G_1$,$H_3 : G_2 \to Z_q^*$.

FileKey $(params, F) \to (SK_F, PK_F)$: This algorithm is run by user. Input a file F,a user computes private key $SK_F = f = H_1(F)$ and public key $PK_F = Z^f = Z^{H_1(F)}$,then outputs a file key pair (PK_F, SK_F).

UserKey $(params) \to (SK_u, PK_u)$: This algorithm is run by user. A user choices random element a from the set Z_q^* as user private key SK_u and computes user public key $PK_u = g^a$,then outputs a user key pair (PK_u, SK_u).The user key pair is generated by itself without the help of KGC, and the key information is not involved in the process of interacting with the server, but only kept by the user, thus ensuring the privacy of the user key.

ReKey$(SK_F, PK_u) \to RK_{F\to u}$: This algorithm is run by user. It inputs the file private key SK_F and the user public key PK_u, then outputs file F about user's re-encryption key $RK_{F\to u} = g^{a\cdot f}$. The $RK_{F\to u}$ is sent to the cloud server and stored in the ciphertext chain table, which is used only for the specified user to recover the specified file owned by itself.

FileTag$(params, F, SK_F) \to Tag_F$: This algorithm is run by user. It inputs the file, then choices random element r_1 from the set Z_q^* and extract keyword $w = H_2(F)$ by file F. It outputs file tag $Tag_F = (T_1, T_2)$ about file F, where $T_1 = g^{r_1}$, $T_2 = H_3\left(e\left(w, g^{r_1\cdot f}\right)\right)$. The user sends Tag_F to the cloud server, and the tag serves as a unique identifier for the file. When the tag is stored in the server, it indicates that the file already exists and does not need to be uploaded repeatedly.

$Enc(F, PK_F) \rightarrow C$: This algorithm is run by user. It inputs the file F and user public key, then returns ciphertext C. This algorithm runs as below:

1) Select random element r_2 from the set Z_q^* and compute $C_1 = g^{r_2}$.
2) Compute $C_2 = F \cdot e(g, g)^{f \cdot r_2} = F \cdot Z^{f \cdot r_2}$.

This algorithm outputs ciphertext $C_2 = (C_1, C_2) = \left(g^{r_2}, F \cdot Z^{f \cdot r_2}\right)$. For non-duplicate files, the user needs to encrypt them using this algorithm and then upload to the cloud server for storage.

$TestTag(F) \rightarrow T_F$: his algorithm is run by user. It inputs the file F, then computes test tag $T_F = H_2(F)^{H_1(F)}$. The test tag is used to verify that the corresponding file already exists in the cloud server.

$Test(T_F, Tag_F) \rightarrow \perp$: This algorithm is run by cloud server. It inputs test tag T_F and file tag Tag_F, the cloud server verifies whether $H_3(e(T_F, T_1)) = T_2$ is valid. When the equation holds, it indicates that the file is duplicated, so the user doesn't need to upload the ciphertext, but only needs to generate the re-encryption key $RK_{F \rightarrow u}$ of the duplicate file about itself by the algorithm, and store the key in the ciphertext chain table of the corresponding ciphertext.

$ReEnc(C, RK_{F \rightarrow u}) \rightarrow C'$: This algorithm is run by cloud server. It inputs original ciphertext C and re-encryption key $RK_{F \rightarrow u}$, then computes transformed ciphertext $C' = \left(C_1', C_2'\right)$. This algorithm runs as below:

$C_1' = e(C_1, RK_{F \rightarrow u}) = Z^{r_2 \cdot a \cdot f}$, $C_2' = C_2$. When the user needs to obtain the duplicate file, he only needs to send a request to the server, which will find the corresponding re-encryption key in the corresponding ciphertext chain table, use the re-encryption key to generate the transformed ciphertext and send it to the user.

$ReDec\left(C', SK_u\right) \rightarrow F$: This algorithm is run by user. When the user receives the transformed ciphertext C' from the server, the user can simply use his private key SK_u to compute $F = C_2' / \left(C_1'\right)^{1/a}$.

C. CORRECTNESS ANALYSIS

1) CORRECTNESS OF DE-DUPLICATION

The user needs to perform de-duplication before uploading files to the server, and when the files are duplicated, there is no need to upload the files. The user computes the test tag $T_{F'}$, and uses the test tag $T_{F'}$ and the file tag $Tag_{F'}$ as input to the algorithm *Test* to determine whether the file F' is duplicated. The specific de-duplication process is as follows:

Using the bilinear pairing properties it follows that:

$$\begin{aligned} H_3\left(e\left(T_{F'}, T_1\right)\right) &= H_3\left(e\left(H_2\left(F'\right), g\right)^{f \cdot r_1}\right) \\ T_2 &= H_3\left(e\left(H_2\left(F\right), g^{f r_1}\right)\right) \\ &= H_3\left(e(H_2\left(F\right), g)^{f r_1}\right) \end{aligned}$$

The above equation shows that if the test tag corresponds to the same file as the file tag, then $H_3(e(H_2(F'), g)^{f \cdot r_1}) = H_3(e(H_2(F), g)^{f \cdot r_1})$ show that the equation $H_3(e(T_{F'}, T_1)) = T_2$ holds. Since the hash function is collision-resistant, when $H_2(F') \neq H_2(F)$ then it means $F' \neq F$. Therefore, the output of the algorithm *Test* is true for the same file and false for a different file, thus determining whether the file is a duplicate.

2) CORRECTNESS OF RE-DECRYPTION

For the user who owns the file, the ciphertext chain table corresponding to the ciphertext stored in the cloud server should hold the re-encryption key of the user. This re-encryption key is generated from the file encryption key and the user's public key, forming a one-to-one correspondence between the file and the user's identity. Therefore, the user can decrypt the corresponding transformed ciphertext with his private key. The specific decryption calculation process is as follows:

$$\begin{aligned} C_2' / \left(C_1'\right)^{1/a} &= F \cdot e(g, g)^{f r_2} / e(C_1, RK_{F \rightarrow u})^{1/a} \\ &= F \cdot e(g, g)^{f r_2} / e\left(g^{r_2}, g^{a \cdot f}\right)^{1/a} \\ &= F \cdot e(g, g)^{f r_2} / e(g, g)^{f r_2} \\ &= F \end{aligned}$$

The above equation shows that if and only if the user has the file, he can use his private key to recover the file F correctly, while for users who don't have access to the file, the generated transformed ciphertext cannot be decrypted.

D. WORK PROCESS

This program can realize secure data de-duplication and data recovery. This subsection mainly describes the workflow of this program, which is mainly divided into two phases: data de-duplication and data recovery.

Phase 1. data de-duplication

The workflow of the data de-duplication phase is depicted in Fig 2. The data user generates file tags Tag_F based on the files F and uploads them to the cloud server to form the file tag index. When the data user uploads a file, a test tag T_F is generated and sent to the CSP, which uses the matching relationship between Tag_F and T_F to determine whether there is a file tag that matches the test tag uploaded by the user, and sends the test result $Test\ (T_F, Tag_F)$ to the data user. If there exist a matching file tag, return true, it means that there is already a duplicate file in the CSP, and the user does not need to encrypt the file, but only needs to upload the file re-encryption key about the user to the ciphertext chain table of the corresponding file in the cloud server. If there is no matching file tag, return false, it means that there is no duplicate file in CSP, then the user as the data owner needs to generate file ciphertext, file tag, re-encryption key about the user and upload it to the cloud server.

Phase 2. Data recovery

The workflow of the data recovery phase is depicted in Fig 3. The data user sends a request to get the file, and the CSP queries whether the re-encryption key $RK_{F \rightarrow u}$ of the user exists in the ciphertext chain table of the file F. When the user's re-encryption key exists in the ciphertext chain table of the file F, the cloud server re-encrypts the original ciphertext C of the file, generates the transformed ciphertext

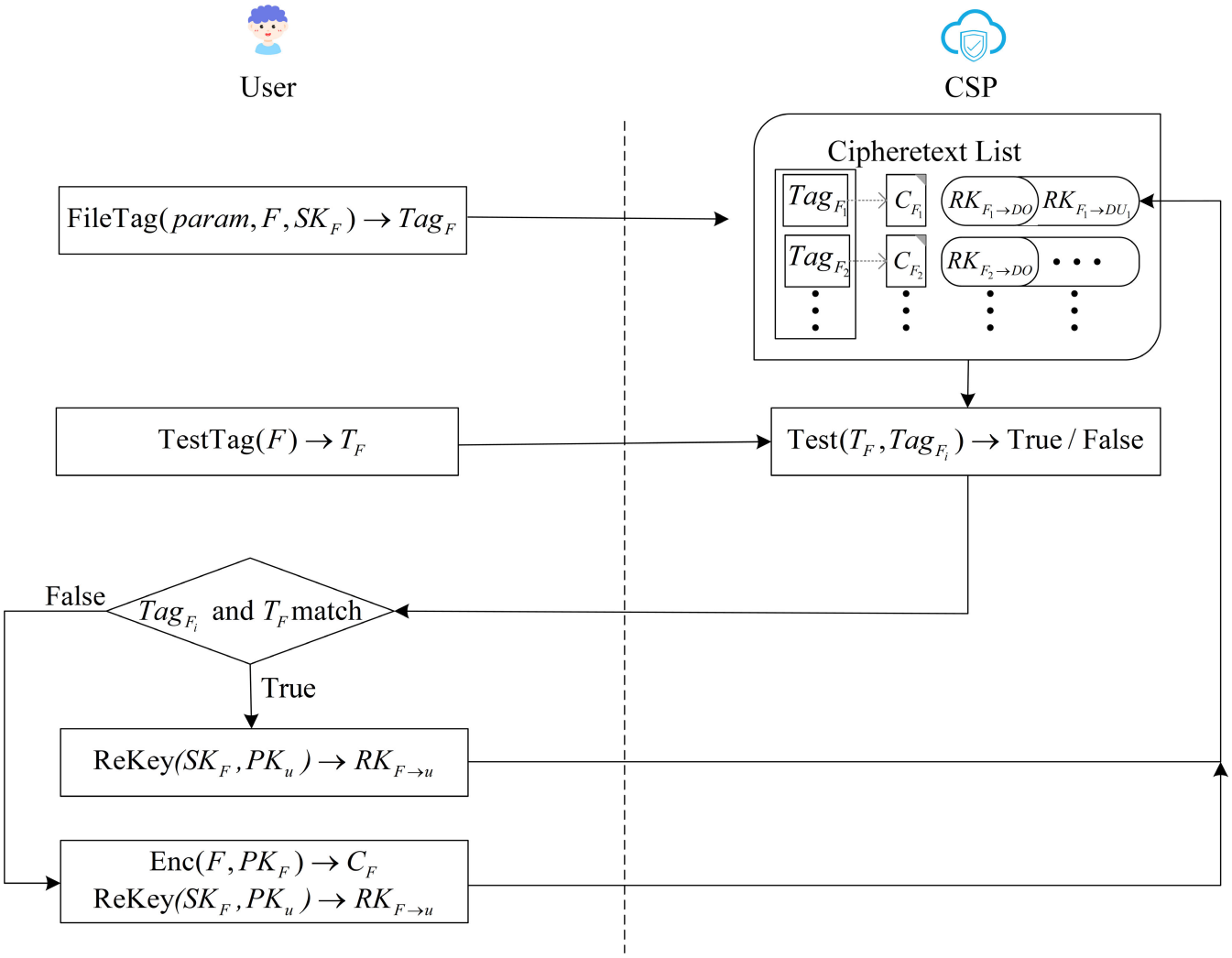

FIGURE 2. File de-duplication process.

C', and sends it to the user. After receiving the transformed ciphertext, the user can use private key SK_u to decrypt the file F. When the user's re-encryption key doesn't exist in the ciphertext chain table of the file F, it means that the user cannot access the file F.

III. SECURITY PROOF

A. SECURITY MODEL

In this paper, we consider an insider attacker malicious CSP server. This type of attacker will comply with the execution of the protocol, but can obtain the file tag and ciphertext of any file. In addition, the attacker can obtain the test tag and re-encryption key of the file. For the target file, the attacker is not allowed to obtain the test tag of the file. If the attacker knows the private key of user, the attacker is also not allowed to obtain the re-encryption key of the target file to that user. For this type of attacker, the formal definition of the security model of the scheme is given below.

Setup. The challenger generates system parameters $params = \{q, g, e, Z, G_1, G_2, H_1, H_2, H_3\}$ from *Setup* and generates the key pair of the challenger user from *UserKey*. The challenger sends *params* and PK_u to the attacker $\mathcal{A}$.

Phase 1. $\mathcal{A}$ answers queries as follows.

- File key queries: Input file F, then returns the key pair (PK_F, SK_F) of the file F.
- File key queries: Input file F, then returns the file tag Tag_F of the file F.
- ciphertext queries: Input file F, then returns the ciphertext C_F of the file F.
- Re-encryption key queries: Input file F and user public key PK_u, then returns the re-encryption key $RK_{F\to u}$.
- Test tag queries: Input file F, then returns the test tag T_F of the file F.

In the above queries, in addition to the challenge file F^*, the attacker can complete various queries for other files based on the relevant algorithms.

Challenge. The challenger randomly selects a challenge file F^* from the file space G_2 computes the file tag Tag_{F^*} and ciphertext C_{F^*}, and returns them to $\mathcal{A}$.

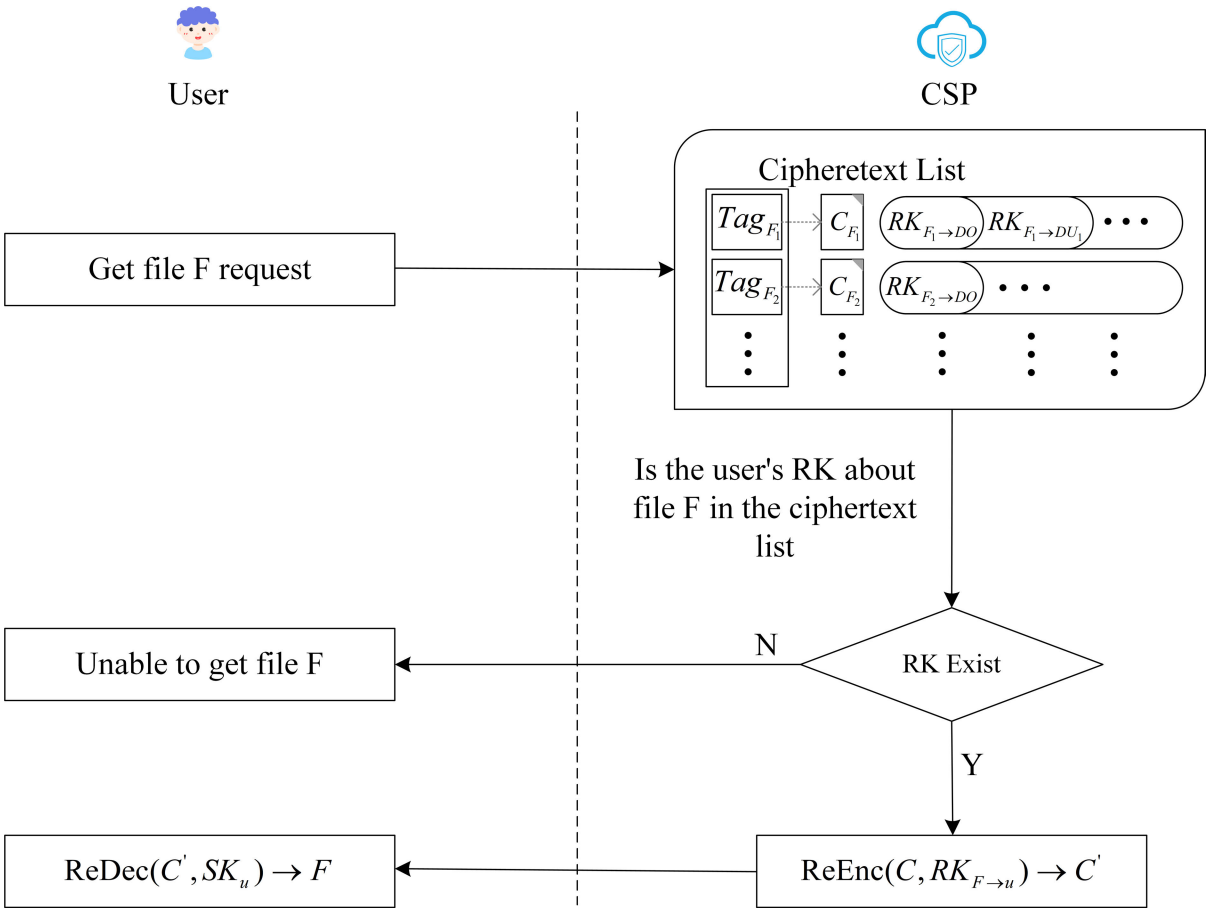

FIGURE 3. File recovery processs.

Phase 2. $\mathcal{A}$ can ask for queries related to any file in the same way as in *phase*1, but cannot ask for queries related to a challenge file F^* as follows:

- File private key of challenge file F^*.
- $\mathcal{A}$ can obtain the re-encryption key $RK_{F^*\to u}$ from the challenge file F^* to the challenge user, but cannot ask for the re-encryption key from the challenge file to another user whose private key is known.
- File test tag of challenge file F^*.

Guess. $\mathcal{A}$ returns a file $F \in G_2$. If $F = F^*$, then the attacker succeeds, otherwise the attacker fails.

For any Probabilistic Polynomial-Time (PPT) attacker, a scheme is said to satisfy one-wayness under chosen file attack if the probability of the attacker succeeding in the above game is negligible.

B. SECURITY ANALYSIS

SBDH problem. Given a security parameter k, a group G with the order q, a generator g of G, and randomly choose $a, b, c \in Z_q^*$. On input a tuple $\left(g, g^a, g^b, g^c, g^{1/a}, g^{bc/a}\right) \in {G_1}^6$ to compute $e(g,g)^{abc}$. There is an PPT adversary $\mathcal{A}$, the advantage of

$$\begin{aligned}&\mathrm{Adv}_{\mathcal{A}}^{\mathrm{SBDH}}(k)\\&= \Pr\left[\mathcal{A}\left(g, g^a, g^b, g^c, g^{1/a}, g^{bc/a}\right) \to e(g,g)^{abc}\right],\\&\le \varepsilon\end{aligned}$$

if ε is negligible, the SBDH problem is hard to solve by $\mathcal{A}$.

Theorem 1: If the SBDH problem is hard to solve, under the random oracle model, the scheme satisfies one-wayness security under the chosen file attack.

Specifically, If there exists an algorithm A that attacks the one-wayness of the scheme with probability ε, then another algorithm B can be constructed to solve one solution of the SBDH problem instance with probability at least

$$\left(\frac{1}{2} + \frac{1}{2q_3}\right)\varepsilon - \frac{1}{2q_3 q},$$

where q_3 denotes the number of times the attacker asks the hash function H_3.

Proof Assume that $\mathcal{A}$ is a one-wayness PPT attacker of any attack scheme. Given an instance $\left(g, g^a, g^b, g^c\right) \in {G_1}^4$ of the BDH problem and a bilinear pairing $e : G_1 \times G_1 \to G_2$, if the attacker can successfully attack the one-wayness of the scheme with probability ε, then another simulator algorithm B can be constructed to successfully solve the above BDH problem with at least xxx probability. Algorithm B simulates the execution of the game as follows.

Setup. Simulator B generates system parameter $params = \{q, g, e, Z, G_1, G_2, H_1, H_2, H_3\}$ based on the SBDH problem instance, where $Z = e(g,g)$. $H_1 : \{0,1\}^* \to Z_q^*$, $H_2 : \{0,1\}^* \to G_1$ and $H_3 : G_2 \to Z_q^*$ are the three hash functions

selected by the simulator. The simulator sets the public key of the challenge user u^* as $PK_{u^*} = g^{1/a}$ according to the BDH problem instance and implicitly defines its private key as $SK_{u^*} = 1/a$. In addition, the simulator randomly selects $R \in G_2$ and $k_1 \in Z_q^*$ and implicitly defines $F^* = R \cdot e(g, g)^{-k_1 abc}$ as the challenge file. Finally, the simulator sends *params* and PK_{u^*} to the attacker.

Phase 1. When the attacker makes the following query, the simulator responds as follows.

- Hash queries. The simulator builds three hash lists $L_1 = \{(*, *)\}$, $L_2 = \{(*, *)\}$ and $L_3 = \{(*, *)\}$ with empty initial elements. When the attacker asks the hash function H_1 about the hash value of element $x_{1,i}$, the simulator first checks whether the binary $(x_{1,i}, h_{1,i})$ exists in list L_1 and returns $h_{1,i}$ if it does, otherwise, it randomly selects $h_{1,i} \in Z_q^*$ and returns it to the attacker.
- File key queries. When the attacker asks for the key of file F, the simulator returns the $(PK_F, SK_F) = (Z^{H_1(F)}, H_1(F))$ according to the *FileKey*. In particular, when the attacker asks for the public key of the challenge file F^*, the simulator computes $PK_{F^*} = e(g^b, g^c)$ and returns it to the attacker.
- File tag queries. When the attacker asks for the tag of file F, the simulator computes the file tag Tag_F of F according to *FileTag* and returns it to the attacker.
- File ciphertext queries. When the attacker asks for the ciphertext of file F, the simulator computes the ciphertext C_F according to *Enc* and returns it to the attacker.
- Re-encryption key queries. When the attacker $u \neq u^*$, the simulator computes the re-encryption key $RK_{F \to u}$ according to *ReKey* and returns it to the attacker. If the attacker asks for the re-encryption key from the challenge file to the challenge user, the simulator makes $RK_{F^* \to u^*} = g^{bc/a}$ and returns it to the attacker.
- Test tag queries. When the attacker asks for the test tag of file F, the simulator computes the test tag T_F according to *TestTag* and returns it to the attacker.

Phase 2. The simulator answers the attacker's queries as in *Phase* 1, but the attacker cannot make the following queries.

- $\mathcal{A}$ asks for the private key SK_{F^*} of challenge file F^*.
- $\mathcal{A}$ asks for the re-encryption key of Challenge file F^* to other users.
- $\mathcal{A}$ asks for the test tag of challenge file F^*.

Guess. The attacker returns a guess file $F \in G_2$. The simulator randomly selects a bit $b \in \{0, 1\}$. If $b = 0$ the simulator computes

$$T^* = \left(\frac{R}{F}\right)^{1/k_1}.$$

Otherwise, the simulator randomly selects an element $(x_{3,i}, h_{3,i})$ from the hash list L_3, computes $T^* = \left(x_{3,i}\right)^{3/r_2}$, and uses T^* as the solution to the challenge SBDH problem.

Success probability analysis of the Simulator. If the attacker selects file $F \neq F^*$ in the query phase, then the simulator's hash queries, file key queries, file tag queries, file ciphertext queries, re-encryption key queries, and test tag queries to the attacker are answered in the same way as in the original security model. Since challenge file F^* is randomly and independently selected by the simulator, the probability that the attacker selects a file F equal to the challenge file in each queries phase does not exceed $1/q$. The correctness of the challenge tag and challenge ciphertext is analyzed below.

Because of $C_1^* = (g^a)^{k_1}$ and $F^* = R \cdot e(g, g)^{-k_1 abc}$, it follows that $C_2^* = F^* \cdot e(g, g)^{f \cdot r_2} = R \cdot e(g, g)^{-k_1 abc} \cdot e(g, g)^{k_1 abc} = R$. The challenge file ciphertext in the simulated game is consistent with the challenge ciphertext distribution in the original model.

Because of $T_1^* = g^{r_1}$ and $H_2(F^*) = (g^a)^{k_2}$, it follows that $e(H_2(F), g^{r_1 bc}) = e(g, g)^{abcr_2}$. If the attacker has never asked H_2 about the hash of $e(g, g)^{abcr_2}$, then the simulator randomly chooses $T_2^* \in Z_q^*$ to be consistent with the challenge tag distribution in the original model. If the attacker has asked H_2 about the hash of $e(g, g)^{abcr_2}$, the simulator randomly chooses $x_{3,i}$, from the query list of H_2. Then $\left(x_{3,i}\right)^{1/r_2}$ is a solution to the SBDH problem with the probability of at least $1/q_3$.

Let E denote the event "Attacker asks H_2 about the hash of $e(g, g)^{abcr_2}$, and prove below that the probability of E occurring is at least $\varepsilon - 1/q$. When E does not occur, $\Pr[F = F^* | \neg E] = 1/q$ is completely independent of the challenge file because the challenge tag and challenge ciphertext are completely independent of the challenge file. Because

$$\begin{aligned}
\Pr[F = F^*] &= \Pr[F = F^* | E] \cdot \Pr[E] \\
&= + \Pr[F = F^* | \neg E] \cdot \Pr[\neg E] \\
&\leq \Pr[E] + \frac{1}{q} \Pr[\neg E] \\
\Pr[E] &\geq \frac{\Pr[F = F^*] - 1/q}{1 - 1/q} \geq \varepsilon - 1/q
\end{aligned}$$

In summary, the analysis shows that if the attacker breaches the security of the scheme with probability ε, the simulator obtains at least one solution to the SBDH problem with probability

$$\left(\frac{1}{2} + \frac{1}{2q_3}\right)\varepsilon - \frac{1}{2q_3 q}.$$

IV. PERFORMANCE EVALUATION

A. THEORETICAL ANALYSIS

The proposed scheme in this paper involves entities such as data users and cloud servers, and data users divide into data owners and data users based on the order of uploading files. For the entities involved in the paper [17] are data owner, data user, cloud server, and KGC. The key generation involved in this scheme requires the participation of KGC, while the key generation in the scheme of this paper are generated by the user without the need of a third party trusted institution. Although key generation increases the computational overhead of the client, it is able to avoid KGC attacks [7]. In Table 1, by comparing with paper [17], the algorithms in paper [17] and this paper are grouped into

TABLE 1. Computational cost comparison.

Scheme	UserKey	FileKey	FileTag	TestTag	Test	Enc	Dec	ReKey	ReEnc	ReDec
[17]	1E+1H	-	1P+1E+2H	2E+1H	2P+1E+1H	1P+3E+1H	1E	1P+1H	1P	2P+1H
Ours	1E	1E+1H	1P+2E+2H	1E+2H	1P+1H	2E	-	1E	1P	1E

the ten algorithms involved in Table 1 according to their functions for the sake of comparative analysis. The correspondence between the algorithms of the scheme in this paper and the algorithms in the paper [17] is, where *UserKey* is equivalent to the *Key generation*, *FileTag* algorithm is equivalent to the *Data tag generation*, *TestTag* is equivalent to the *Ownership challenge and dedupliaction* in the Test tag *s* generation algorithm, *Test* is equivalent to the process of equality test by CSP in *ownership challenge and dedupliaction*, and the rest of the algorithms not mentioned can be directly corresponded to the algorithms in Table 1. E denotes a exponentiation operation, H denotes a hash operation, and P denotes a pairing operation, and '-' denotes the algorithm does not exist. Since the paper [17] uses user keys for file encryption, it doesn't need to execute the *FileKey* algorithm, while the scheme in this paper uses proxy re-decryption for both data owner and data user decryption, so it doesn't need to execute *Dec*.

The computational overheads incurred in *UserKey* and *ReEnc* are basically the same for both schemes. The computational overhead of this paper's scheme in generating test tags increases by one exponential operation compared to the paper [17], which will lead to an increase in computational overhead. For each execution of *Test* algorithm, our scheme needs to perform one bilinear pairing operation. In contrast, the paper [17] requires two bilinear pairing operations, so there is a significant difference in efficiency when the equality test is performed on a large number of data. Moreover, the computational withholding overheads of the encryption algorithm and the re-decryption algorithm in this paper are smaller than those in the paper [17]. Therefore, through theoretical analysis, the computational overhead of the scheme in this paper is relatively smaller and can save some computational resources in practical application scenarios.

B. SIMULATION ANALYSIS

In order to analyze the efficiency, our scheme and the comparison scheme are implemented by JPBC cryptographic library. Regarding JPBC cryptographic library, uses type A pairing parameters for simulation. Type A pairing is constructed based on elliptic curve $y^2 = x^3 + x \bmod p$ where $p \equiv 3mod4$ the group G_1, G_2, G_T generated using type A bilinear pairing in JPBC are 512bit symmetric group, and the set Z_q is 160bit. The testing environment is the desktop with AMD R7 5700U@1.80 GHz and 8 GB RAM. The results of each experiment are averaged over 100 runtimes because of the uneven time consumption of the algorithm initialization.

Analyze the computational overhead of each algorithm in the scheme. According to the function of the secure de-duplication system, it is divided into ten algorithms in Fig 4. As shown in Fig 4, the computational overhead of each algorithm is derived from simulations in which the file size used is 1MB. It can be seen that the computational overhead generated by our scheme and the paper [17] in generating user keys and re-encryption is basically the same. Since the equality test in the de-duplication system requires Test algorithm for each file in the cloud storage, in order to improve the efficiency of equality test, our scheme increases the computational overhead in *FileTag* compared to *Test* to reduce the computational overhead of *Test*, and the computational overhead in *Test* is about $1/3$ of that of the paper [17]. For *TestTag*, *Enc*, *ReKey*, *ReDec* algorithm, the simulation results show that the scheme of our scheme is better than the paper [17]. In terms of the overall execution efficiency of the system, the time required in a single process is about 82.2 ms in ours, while the paper [17] is about 137.65 ms. It can be seen that the computational overhead of the scheme in this paper is lower than that in the paper [17].

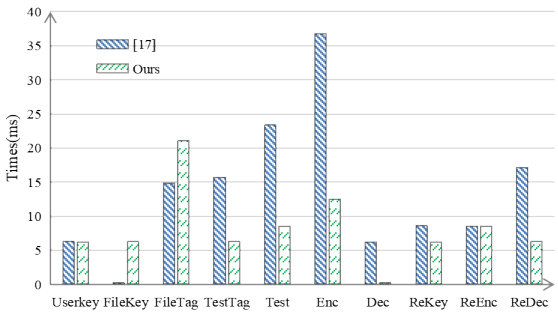

FIGURE 4. Average operation time of each phase of the scheme.

Test the impact of different file sizes on the generated test tags. In the process of equality test of encrypted data, the cloud server determines whether the plaintexts corresponding to the two ciphertexts are the same based on the test tags uploaded by the user and the file tags stored in the database using *Test*, to achieve equality test of the files. The user performs subsequent operations based on the results of equality test. In order to further analyze the performance of test tag generation, we compare this paper with the selected scheme by testing the test tag generation time for different file sizes of 5MB, 10MB, 15MB, 20MB, and 25MB, and the results are shown in Fig 5. With the increase of the upload file size, the time consumed to generate the test tag also increases, and the change of computational overhead of this scheme is basically the same as that of the paper [17]. The computational performance of this scheme in generating the test tag is better than that of the paper [17] because this

scheme reduces one exponential operation when generating the test tag compared with the paper [17].

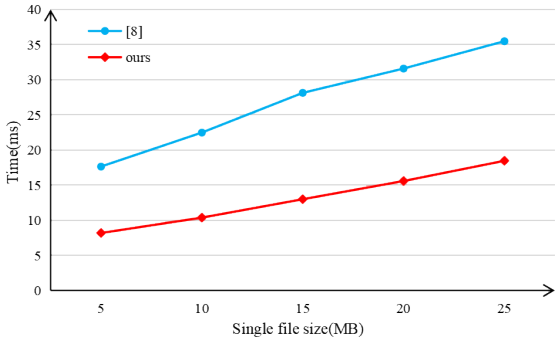

FIGURE 5. Computational overhead of generating file tags for different file size.

Test the impact of different numbers of files on the equality test of the system. In the process of test, the cloud server compares the uploaded test tags with all the file tags in the database to determine whether there are duplicate files in the database, and based on the judgment result, the user performs the next operation. In order to analyze the performance advantages and disadvantages of this paper's scheme and the comparison scheme in the process of equality test, the implementation of storing 1000, 2000, 3000, 4000, 5000 file tags in the database, size of each file is 1MB. Through simulation analysis, the results obtained are shown in Fig 6. It can be seen through Fig 6 that the time consumed for equality test increases linearly with the increase of the number of files in the database. Due to the large number of files in the database, compared to the paper [17] ours scheme has obvious advantages in practical applications.

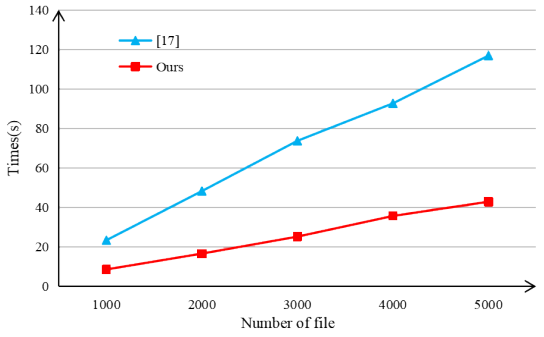

FIGURE 6. Computational overhead of duplication test for different file number.

V. CONCLUSION

Secure data de-duplication is of great value in cloud storage, and it can effectively improve the space utilization of cloud storage systems. In this paper, a Secure data de-duplication and recovery scheme based on PEKS is constructed by using the matching relationship between keyword and trapdoor of searchable encryption to achieve file equality test in ciphertext state and using proxy re-encryption to achieve data recovery. Since the de-duplication process of a single file requires the execution of multiple equality test algorithms depending on the size of the database, this scheme is designed to avoid the computational overhead of this algorithm as much as possible. Through experimental simulation, the results show that the scheme in this paper has good performance in a cloud storage system. At present, scholars have made some achievements in the study of Secure data de-duplication and have applied it to practical scenarios. This paper conducts in depth research based on the previous work, but there are still some shortcomings, such as the current scheme of this paper only supports equality test at the file level. In the future, the main consideration is the de-duplication rate. When the data user has two files with only minor differences, this paper will determine them as different files, which will reduce the de-duplication rate.

REFERENCES

[1] 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.*, Vienna, Austria, 2002, pp. 617–624.

[2] A. Agarwala, P. Singh, and P. K. Atrey, "DICE: A dual integrity convergent encryption protocol for client side secure data deduplication," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Banff, AB, Canada, Oct. 2017, pp. 2176–2181.

[3] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted deduplication," in *Proc. 24th Large Installation Syst. Admin. Conf.*, San jose, CA, USA, 2010, pp. 1–12.

[4] 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.

[5] 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.

[6] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, C. Cachin and J. Camenisch, Eds. Interlaken, Switzerland, 2004, pp. 506–522.

[7] M. H. Au, J. Chen, J. K. Liu, Y. Mu, D. S. Wong, and G. Yang, "Malicious KGC attacks in certificateless cryptography," in *Proc. 2nd ACM Symp. Inf., Comput. Commun. Secur.*, New York, NY, USA, 2007, pp. 302–311.

[8] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. 32nd Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, T. Johansson and P. Q. Nguyen, Eds. Athens, Greece, 2013, pp. 296–312.

[9] S. Keelveedhi, M. Bellare, and T. Ristenpart, "DupLESS: Server-aided encryption for deduplicated storage," in *Proc. 22th USENIX Secur. Symp.*, S. T. King, Ed. Washington, DC, USA, 2013, pp. 179–194.

[10] M. Abadi, D. Boneh, I. R. A. Mironov, and G. Segev, "Message-locked encryption for lock-dependent messages," in *Proc. 33rd Annu. Cryptol. Conf.*, Santa barbara, CA, USA, 2013, pp. 374–391.

[11] 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.*, Denver, CO, USA, Oct. 2015, pp. 874–885.

[12] P. Puzio, R. Molva, M. Önen, and S. Loureiro, "PerfectDedup: Secure data deduplication," in *Proc. 10th Int. Workshop, 4th Int. Workshop*, Vienna, Austria, 2015, pp. 150–166.

[13] J. Li, C. Qin, P. P. C. Lee, and J. Li, "Rekeying for encrypted deduplication storage," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Toulouse, France, Jun. 2016, pp. 618–629.

[14] M. Li, C. Qin, J. Li, and P. P. C. Lee, "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," *IEEE Internet Comput.*, vol. 20, no. 3, pp. 45–53, May 2016.

[15] X. Tang, L. Zhou, W. Shan, and D. Liu, "Threshold re-encryption based secure deduplication method for cloud data with resistance against side channel attack," *J. Commun.*, vol. 41, no. 6, p. 14, 2020.
[16] W. Gao, H. Xian, and R. Cheng, "A Cloud data deduplication method based on double-layered encryption and key sharing," *Chin. J. Comput.*, vol. 44, no. 11, pp. 2203–2215, 2021.
[17] G. Kan, C. Jin, H. Zhu, Y. Xu, and N. Liu, "An identity-based proxy re-encryption for data deduplication in cloud," *J. Syst. Archit.*, vol. 121, Dec. 2021, Art. no. 102332.
[18] H. Yuan, X. Chen, J. Li, T. Jiang, J. Wang, and R. H. Deng, "Secure cloud data deduplication with efficient re-encryption," *IEEE Trans. Services Comput.*, vol. 15, no. 1, pp. 442–456, Jan. 2022.
[19] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, Espoo, Finland, 1998, pp. 127–144.
[20] Y. Lu and J. Li, "A pairing-free certificate-based proxy re-encryption scheme for secure data sharing in public clouds," *Future Gener. Comput. Syst.*, vol. 62, pp. 140–147, Sep. 2016.
[21] K. O. Agyekum, Q. Xia, E. Sifah, J. Gao, H. Xia, X. Du, and M. Guizani, "A secured proxy-based data sharing module in IoT environments using blockchain," *Sensors*, vol. 19, no. 5, p. 1235, Mar. 2019.
[22] Q. Wang, W. Li, and Z. Qin, "Proxy re-encryption in access control framework of information-centric networks," *IEEE Access*, vol. 7, pp. 48417–48429, 2019.
[23] H. Hong and Z. Sun, "Sharing your privileges securely: A key-insulated attribute based proxy re-encryption scheme for IoT," *World Wide Web*, vol. 21, no. 3, pp. 595–607, 2018.
[24] X. Liu, J. Yan, S. Shan, and R. Wu, "A blockchain-assisted electronic medical records by using proxy reencryption and multisignature," *Secur. Commun. Netw.*, vol. 2022, Feb. 2022, Art. no. 6737942.
[25] L. Fang, W. Susilo, C. Ge, and J. Wang, "Public key encryption with keyword search secure against keyword guessing attacks without random Oracle," *Inf. Sci.*, vol. 238, pp. 221–241, Jul. 2013.
[26] Y. Lu, J. Li, and Y. Zhang, "Secure channel free certificate-based searchable encryption withstanding outside and inside keyword guessing attacks," *IEEE Trans. Services Comput.*, vol. 14, no. 6, pp. 2041–2054, Nov. 2021.
[27] L. F. Guo and W. C. Yau, "Efficient secure-channel free public key encryption with keyword search for EMRs in cloud storage," *J. Med. Syst.*, vol. 39, p. 11, Feb. 2015.
[28] B. Qin, H. Cui, X. Zheng, and D. Zheng, "Improved security model for public-key authenticated encryption with keyword search," in *Proc. 15th Int. Conf.*, Q. Huang and Y. Yu, Eds. Guangzhou, China, 2021, pp. 19–38.
[29] W. Zhang, B. Qin, X. Dong, and A. Tian, "Public-key encryption with bidirectional keyword search and its application to encrypted emails," *Comput. Standards Interfaces*, vol. 78, Oct. 2021, Art. no. 103542.
[30] B. Chen, L. Wu, S. Zeadally, and D. He, "Dual-server public-key authenticated encryption with keyword search," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 322–333, Jan. 2022.
[31] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "A new general framework for secure public key encryption with keyword search," in *Proc. 20th Australas. Conf.*, E. Foo and D. Stebila, Eds. Brisbane, QLD, Australia, 2015, pp. 59–76.
[32] Y. Lu and J. Li, "Lightweight public key authenticated encryption with keyword search against adaptively-chosen-targets adversaries for mobile devices," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4397–4409, Dec. 2022.

LE LI received the B.S. degree from the Xi'an University of Posts and Telecommunications, Xi'an, China, in 2020, where he is currently pursuing the M.S. degree. His research interest includes cloud storage security.

DONG ZHENG received the M.S. degree in mathematics from Shaanxi Normal University, Xi'an, China, in 1988, and the Ph.D. degree in communication engineering from Xidian University, Xi'an, in 1999. He was a Professor with the School of Information Security Engineering, Shanghai Jiao Tong University, Shanghai, China. He is currently a Professor with the Xi'an University of Posts and Telecommunications and is also connected with the National Engineering Laboratory for Wireless Security, Xi'an. His research interests include provable security and new cryptographic technology.

HAOYU ZHANG is currently pursuing the M.S. degree with the Xi'an University of Posts and Telecommunications. Her research interests include public key searchable encryption and the SM9 algorithm.

BAODONG QIN is currently a Professor with the Xi'an University of Post and Telecommunications. His research interests include cryptography and cloud computing security.