Scalable and popularity-based secure deduplication schemes with fully random tags

Guanxiong Ha, Chunfu Jia, Yixuan Huang, Hang Chen, Ruiqi Li, and Qiaowen Jia

Abstract—It is non-trivial to provide semantic security for user data while achieving deduplication in cloud storage. Some studies deploy a trusted party to store deterministic tags for recording data popularity, then provide different levels of security for data according to popularity. However, deterministic tags are vulnerable to offline brute-force attacks. In this paper, we first propose a popularity-based secure deduplication scheme with fully random tags, which avoids the storage of deterministic tags. Our scheme uses homomorphic encryption (HE) to generate comparable random tags to record data popularity and then uses the binary search in the AVL tree to accelerate the tag comparisons. Besides, we find the popularity tamper attacks in existing schemes and design a proof of ownership (PoW) protocol against it. To achieve scalability and updatability, we introduce the multi-key homomorphic proxy re-encryption (MKH-PRE) to design a multi-tenant scheme. Users in different tenants generate tags using different key pairs, and the cross-tenant tags can be compared for equality. Meanwhile, our multi-tenant scheme supports efficient key updates. We give comprehensive security analysis and conduct performance evaluations based on both synthetic and real-world datasets. The results show that our schemes achieve efficient data encryption and key update, and have high storage efficiency.

Index Terms—Cloud storage, data privacy, encrypted deduplication, data popularity, multi-tenant.

1 Introduction

With the rapid development of cloud computing, more and more individuals or enterprises choose to outsource data to the cloud server. The storage of large volumes of data brings huge overheads to the cloud service providers. Deduplication is an effective way to detect and eliminate redundant copies over clouds. The cloud server only stores the unique data after deduplication to reclaim a lot of storage space. Due to the insecure network environment, users tend to outsource encrypted data to prevent data privacy from being snooped on. However, conventional encryption algorithms aim to provide semantic security for user data. In other words, the encrypted data are indistinguishable from random bits. This property hinders data deduplication since the same messages will be encrypted into indistinguishable ciphertexts. Convergent encryption (CE) [1] is the first attempt to achieve encrypted deduplication. The encryption keys in CE are derived from the data content, so it is a deterministic encryption algorithm and can make sure that identical messages could be encrypted into identical ciphertexts. Nevertheless, CE provides confidentiality only for unpredictable data (the message space can not be exhausted) [2]. For predictable data (the message space can be exhausted), CE is vulnerable to offline brute-force attacks [2].

Since semantic security and data deduplication seem to be irreconcilable, some studies attempt to explore a tradeoff between them. For example, some schemes [3] [4] consider data popularity in encrypted deduplication systems. It is a reasonable way to balance data security and storage efficiency by setting different security levels for user data based on their popularity. Some popular songs or movies may be shared by a large number of users, which can be considered popular data. The medical records or scientific research results are generally shared by only a small number of users, which can be considered unpopular data. Obviously, unpopular data require more protection than popular data. In the popularity-based encrypted deduplication scheme, only when the data become popular will they be encrypted with CE and be deduplicated. The unpopular data are encrypted randomly to be provided with semantic security. Stanek *et al*. [3] use two real-world datasets to analyze the storage efficiency of the popularity-based deduplication scheme. The storage performance is good for datasets containing many files with very high popularity.

Existing popularity-based encrypted deduplication schemes [3] [4] need to deploy a trusted third party to store deterministic tags for recording data popularity. However, if the trusted party is compromised by adversaries, the deterministic tags will be vulnerable to offline brute-force attacks, which is a serious security vulnerability. Besides, we find the "*popularity tamper attack*" in existing popularity-based schemes [3] [4]. Since unpopular data are usually encrypted randomly in popularity-based schemes, it is difficult for the server to verify the data ownership for users. In other words, it is difficult to design proof-of-ownership (PoW) protocols [5] [6] for unpopular data in popularity-based schemes. However, in an encrypted deduplication scheme without the PoW protocol, the data popularity can be tampered with by adversaries with only a small part of the data. The cloud server may increase the number of owners by mistake under the attacks launched by adversaries. We call it the "*popularity tamper attack*", which will be described in detail in Section 2.3.

- *G. Ha, C. Jia, Y. Huang, and H. Chen are with the College of Cyber Science, Nankai University, Tianjin, China and Tianjin Key Laboratory of Network and Data Security Technology. E-mail: cfjia@nankai.edu.cn.*
- *R. Li is with the College of Safety Science and Engineering, Civil Aviation University of China, Tianjin, China.*
- *Q. Jia is with the Institute of Software, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing, China.*

1.1 Contribution

In this paper, we first present a popularity-based encrypted deduplication scheme with fully random tags. We use homomorphic encryption (HE) to generate random data tags and achieve a ciphertext-level comparison, which eliminates the requirement of storing deterministic tags to record data popularity. Furthermore, we use an AVL tree to store random tags based on their homomorphism and alleviate the inefficiency of the random tag comparison by binary search. Compared with other fully random tag comparison schemes [7] [8], our method does not need the involvement of clients and supports dynamic insertion and deletion of tags. Besides, we find the popularity tamper attack in the popularity-based deduplication and design a PoW protocol based on HE to resist it. Only the users with the whole data content could increase the count of data owners.

Since the multi-tenancy and scalability are crucial in cloud storage, we further expand our scheme with the multi-key homomorphic proxy re-encryption (MKH-PRE) to make it enjoy these properties. In our multi-tenant scheme, users can be divided into different tenants. Cross-tenant users generate their random tags with distinct HE key pairs, while the cloud server can record data popularity by comparing the equality of cross-tenant tags due to the property of MKH-PRE. Note that a tenant is the customer of the cloud storage service. For example, a tenant can be an enterprise or government department. As a side benefit of using MKH-PRE, our multi-tenant scheme supports efficient key updates for random tags based on the proxy re-encryption. Our contributions are summarized as follows:

- First, we propose a popularity-based encrypted deduplication scheme, which is the first attempt to record data popularity based on fully random tags instead of deterministic tags. Then, we use the binary search to reduce the time complexity of tag comparison to logarithmic time.
- Second, we point out the popularity tamper attack in existing popularity-based encrypted deduplication schemes [3] [4] and design a PoW protocol based on HE to resist it.
- Third, we introduce the MKH-PRE into our scheme to make it enjoy multi-tenancy and scalability. Besides, the efficient key update can be achieved in our multi-tenant scheme.
- Finally, we give the security analysis and conduct performance evaluations. The evaluation results show that the introduction of HE and MKH-PRE does not bring much extra computation overheads. Compared with the scheme proposed by Stanek *et al.* [3], our schemes have a slight improvement in encryption efficiency.

This is the full version of the paper that has been accepted by TrustCom 2021 [9]. Compared with the conference version, the main differences are as follows: First, we expand our original scheme to a multi-tenant scheme for scalability and updatability by introducing MKH-PRE. Second, we add integrity verification to the original scheme to provide data integrity. Third, we give a more elaborate security model and security analysis for our schemes compared with the conference version. Finally, we give a more comprehensive performance evaluation. We add the performance comparison with [7], add the evaluation of the communication overhead and storage overhead, and evaluate the performance of the key update.

2 Background and Motivation

2.1 Encrypted Deduplication

In conventional encryption algorithms, messages are encrypted into random bit strings and are hard to be deduplicated. Convergent encryption (CE) [1] is the first solution to achieve encrypted deduplication, which derives the encryption key from the data content. It utilizes the property of deterministic encryption to realize encrypted deduplication. Bellare *et al*. [10] formalize CE as the message-locked encryption (MLE) and analyze its security. They state that MLE is vulnerable to *offline brute-force attacks* and can only provide confidentiality for unpredictable messages. DupLESS [2] deploys a key server to introduce a system-level secret for MLE to offer confidentiality for both predictable and unpredictable messages. However, the key server may become a single point of failure and efficiency bottleneck. Some schemes [11] [12] [13] focus on designing encrypted deduplication schemes without additional independent servers. But these schemes all require a certain percentage of clients online, which makes it difficult to apply them in real scenarios. In sum, how to provide semantic security for outsourced data while achieving data deduplication is still an important issue.

2.2 Client-side Deduplication

Deduplication can be divided into server-side and client-side deduplication. The former needs clients to upload all data to the server, then the server performs the data deduplication. The latter needs clients first to upload data tags to the server. If the server finds that the data have already been stored, then it marks the client as a data owner. Therefore, clients only need to upload the data that are not stored in the server. Compared with server-side deduplication, client-side deduplication can save both bandwidth and storage overheads.

However, client-side deduplication suffers from *ownership cheating attacks* [5]. Concretely, small data tags calculated from the data are used to identify data ownership. The adversary with only data tags can convince the server that it has the whole data and can download data. Halevi *et al*. [5] find this vulnerability and present a proof-of-ownership (PoW) protocol based on the Merkle tree and specific encodings to let the client efficiently prove to the server that it does have the whole data content. The limitation is that the server in their scheme is trusted. Xu *et al*. [6] propose a PoW scheme against the honest-but-curious server. Their scheme is proved secure with regard to any distribution with sufficient min-entropy. Most existing PoW schemes [5] [6] [14] [15] [16] [17] focus on the data encrypted by deterministic encryption (such as MLE) rather than random encryption. How to design PoW for random ciphertexts is still an unsolved problem.

Besides, the duplicate faking attack [10] is also a threat to client-side deduplication. If a malicious uploader sends

a correct data tag and a fake ciphertext to the server, the server will only store this fake ciphertext and deduplicate the subsequent uploaded data. Then the subsequent data uploader may download and restore the fake data, and the data integrity is compromised.

2.3 Popularity-based Encrypted Deduplication

Recently, several encrypted deduplication schemes [3] [4] try to provide a fine-grained security-to-efficiency trade-off based on data popularity. They set different security levels for user data based on how popular they are (see Figure 1). If the number of data owners exceeds a popularity threshold, then the data can be considered popular and just need to be encrypted with CE. Otherwise, they should be considered unpopular and be encrypted with semantically-secure encryption algorithms. Stanek *et al*. [3] use a threshold cryptosystem to design a popularity-based encrypted deduplication scheme. When the data become popular, the storage server (i.e. the cloud server) decrypts the random ciphertexts into convergent ciphertexts using enough decryption shares. PerfectDedup [4] leverages the perfect hash function to enable clients to securely check data popularity without leaking data information to the storage server. However, existing schemes [3], [4] have the following limitations.

- **The storage of deterministic tags**. Existing schemes [3], [4] need to deploy a trusted third party to maintain the correspondence between deterministic tags and random tags for recording data popularity, as shown in Figure 2. The purpose of this is to let the honest-but-curious storage server only store random tags instead of deterministic tags (the hashes of data) to prevent data information leakage. But, once the deterministic tags are leaked to an adversary, the semantic security for unpopular data will be compromised. With the increasing amount of outsourced data, the number of deterministic tags that the trusted third party needs to store also increases rapidly. The storage of a large number of deterministic tags greatly increases the risk of data information leakage.
- **Popularity tamper attack**. Existing schemes [3], [4] are vulnerable to the popularity tamper attack since they do not perform PoW for unpopular data. Specifically, during the uploads for unpopular data, existing schemes cannot verify whether a client indeed has complete data content due to the lack of the PoW for random ciphertexts. So, a malicious client without complete data content can convince the storage server of its data ownership by uploading only a data tag. Assuming that an adversary has the data tag of a target file F_t and compromises multiple clients, it can have them upload data tags to the storage server to tamper with data popularity. Once F_t becomes popular data, its security protection will be degraded. For predictable data, the adversary can first launch the popularity tamper attack to make them become popular and degrade their security protection, then it can launch offline brute-force attacks to restore data information. In a word, the popularity tamper attack is a serious security concern for popularity-based deduplication schemes.

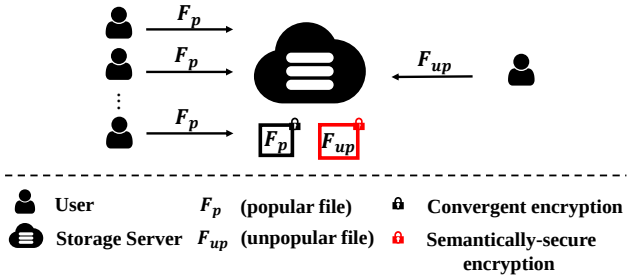


Fig. 1. The popular-based encrypted deduplication system.

Storage Server

Random Tag	Data
rT_1	c_1
rT_2	c_2
⋮	⋮
rT_n	c_n

Trusted Third Party

Deterministic Tag	Random Tag	Ctr
dT_1	$\{rT_{1_1}, rT_{1_2}, ..., rT_{1_{p_1}}\}$	p_1
dT_2	$\{rT_{2_1}, rT_{2_2}, ..., rT_{2_{p_2}}\}$	p_2
⋮	⋮	⋮
dT_m	$\{rT_{m_1}, rT_{m_2}, ..., rT_{m_{p_m}}\}$	p_m

Fig. 2. The records of the storage server and trusted third party.

2.4 Random Tag

Data tags in encrypted deduplication systems are used to detect data duplication. If two pieces of data are identical, then their tags should be identical. In existing encrypted deduplication systems, the hashes of user data are generally used as data tags. However, the hashes are deterministic and are easy to leak data information. Abadi *et al*. [8] propose R-MLE2, a fully randomized encrypted deduplication scheme to provide lock-dependent security. They use fully random data tags to detect data duplication. However, the time complexity of their tag comparison algorithm is linear and inefficient. μR-MLE2 [7] uses a decision tree to store random tags and reduces the time complexity of tag equality-testing to logarithmic time by having clients constantly interact with the server. But the insertion or deletion of an intermediate node in the decision tree requires the assistance of clients. Therefore, their scheme requires the assumption that many clients are always online, which makes their scheme less practical. So, the equality-testing of random tags is still a challenge.

3 PRELIMINARIES

3.1 Convergent Encryption

Convergent encryption (CE) uses the hash of message content as the encryption key to enable deduplication. It can be defined with the following algorithms.

- $\mathrm{CE.KeyGen}(M)$: On input a message M, output a convergent key K on M.
- $\mathrm{CE.TagGen}(M)$: On input a message M, output a data tag T on M.

- $\mathrm{CE.Enc}(K, M)$: It is a symmetric encryption algorithm. On input a convergent encryption key K and a message M, output a convergent ciphertext C.
- $\mathrm{CE.Dec}(K, C)$: It is a symmetric decryption algorithm. On input the convergent key K and convergent ciphertext C, output the message M.

3.2 AVL Tree

The AVL tree [18] is a self-balancing binary search tree with an equilibrium condition. In other words, the absolute value (equilibrium factor) of the difference between the heights of the left and right subtrees of each node in the tree is at most 1. When inserting or deleting a node, the AVL tree adjusts the related subtree structure according to the algebraic relation between nodes, and always maintains the equilibrium factor to less than or equal to 1. Hence, when the AVL tree contains n nodes, the height h of the tree is $\log(n)$. If a search, insert or delete operation is performed, in the worst case, it needs to traverse the height of the tree, and the time complexity is $\mathcal{O}(\log(n))$.

3.3 Homomorphic Encryption

Homomorphic encryption (HE) [19] [20] [21] is a probabilistic encryption algorithm that allows users to perform arithmetic operations on encrypted data without decryption. The HE scheme consists of the following algorithms:

- $\mathrm{HE.KeyGen}(\lambda)$: On input a security parameter λ, output the public key pk and secret key sk.
- $\mathrm{HE.Enc}(pk, m)$: On input a message m and the public key pk, output the ciphertext ct of m.
- $\mathrm{HE.Dec}(sk, ct)$: On input the ciphertext ct of m and the secret key sk, output the message m.
- $\mathrm{HE.Add}(ct_1, ct_2)$: On input ciphertexts ct_1 and ct_2 of m_1 and m_2, output the ciphertext ct_{add} of (m_1+m_2).
- $\mathrm{HE.Sub}(ct_1, ct_2)$: On input ciphertexts ct_1 and ct_2 of m_1 and m_2, output the ciphertext ct_{sub} of (m_1-m_2).

3.4 MKH-PRE

Multi-key homomorphic encryption (MKHE) [22] [23] [24] [25] is a generalization of HE in the multi-user setting, which supports the homomorphic computation on ciphertexts encrypted with different key pairs. Yasuda *et al*. [26] propose multi-key homomorphic proxy re-encryption (MKH-PRE), which adds the function of the proxy re-encryption on the basis of MKHE. In MKH-PRE, each key pair has a corresponding id. Each ciphertext is accompanied by an id set T, which is used to record all key pairs involved in the encryption process. The set T_0 of a freshly generated ciphertext only consists of one element, while the set T_m of the ciphertext after multi-key homomorphism computation may become $T_m = \{id_0, id_1, ..., id_n\}$. An MKH-PRE scheme consists of the following algorithms:

- $\mathrm{MP.KeyGen}(\lambda)$: On input a security parameter λ, output the public key pk and secret key sk.
- $\mathrm{MP.Enc}(pk, m)$: On input a message m and the public key pk, output the ciphertext ct of m.
- $\mathrm{MP.Add}(ct_1, ct_2, \{pk_i\}_{i\in T_A})$: On input ciphertexts (ct_1, ct_2) of (m_1, m_2) and the corresponding public keys $\{pk_i\}_{i\in T_A}$, output the ciphertext ct_{add} of $(m_1 + m_2)$. The id set of ct_{add} is T_A, which is equal to $T_1 \cup T_2$. Note that T_1 and T_2 respectively denote the id set of c_1 and c_2.
- $\mathrm{MP.Sub}(ct_1, ct_2, \{pk_i\}_{i\in T_A})$: On input ciphertexts (ct_1, ct_2) of (m_1, m_2) and the corresponding public keys $\{pk_i\}_{i\in T_A}$, output the ciphertext ct_{sub} of $(m_1 - m_2)$. The id set of ct_{sub} is T_A, which is equal to $T_1 \cup T_2$. Note that T_1 and T_2 respectively denote the id set of c_1 and c_2.
- $\mathrm{MP.Dec}(\{sk_i\}_{i\in T}, ct)$: On input a ciphertext ct and the corresponding secret keys $\{sk_i\}$ in its id set T, output the message m.

The decryption of the multi-key ciphertext needs to input all secret keys in the id set. However, it is unreasonable to allow one party to own all secret keys. Therefore, the MKH-PRE provides the *distributed decryption*, which consists of the PartDec and FinDec. Specifically, one party divides the ciphertext ct into several ciphertext shares ct_i according to its id set, and then sends them to each party related to the decryption. Each related party invokes the PartDec to decrypt ct_i with its secret key sk_i, and outputs a partial decryption share ρ_i. Finally, a party collects all ρ_i to invoke the FinDec to restore the message m. The definitions of the PartDec and FinDec are described as follows:

- $\mathrm{PartDec}(ct_i, sk_i)$: On input a ciphertext share ct_i and a secret key sk_i, output a partial decryption share ρ_i.
- $\mathrm{FinDec}(\{\rho_i\})$: On input all partial decryption shares $\{\rho_i\}$, output the message m.

Besides, the MKH-PRE also supports proxy re-encryption. The algorithms of the re-encryption key generation and re-encryption are described as follows:

- $\mathrm{MP.RKGen}(sk_A, sk_B)$: On input the secret keys sk_A and sk_B, output a re-encryption key $rk_{A\to B}$.
- $\mathrm{MP.ReEnc}(rk_{A\to B}, ct_A)$: On input a re-encryption key $rk_{A\to B}$ and a ciphertext ct_A, output a re-encrypted ciphertext ct_B. Note that ct_A and ct_B are respectively protected by (pk_A, sk_A) and (pk_B, sk_B).

Remarks. The homomorphic encryption algorithms (HE and MKH-PRE) in our schemes are implemented based on the Brakerski-Fan-Vercauteren (BFV) [27] and NTRU [28]. The ciphertext polynomials in BFV and NTRU are both defined over a ring. The plaintexts need to be encoded to plaintext polynomials in binary. We set the plaintext modulus as 3, so a homomorphic ciphertext can be decrypted into a plaintext polynomial with coefficients in $\{-1, 0, 1\}$. After we decrypt the output ciphertext of $\mathrm{HE.Sub}(ct_1, ct_2)$ and then obtain $(m_1 - m_2)$, we could get the algebraic relation between m_1 and m_2 by extracting the highest order coefficient Co of $(m_1 - m_2)$. If $Co = 1$, then $m_1 > m_2$. If $Co = 0$, then $m_1 = m_2$. Otherwise, $m_1 < m_2$.

4 SECURE POPULARITY-BASED DEDUPLICATION SCHEME

4.1 Main Idea

There are several challenges that need to be addressed in popularity-based encrypted deduplication.

The first challenge is how to record data popularity without compromising the semantic security of unpopular

data. Since data tags are used to perform the duplication check, it is necessary to perform the equality-testing on them to record the count of data owners to reflect the data popularity. It is trivial to perform equality-testing on deterministic tags. But they are vulnerable to offline brute-force attacks. Therefore, we want to design a scheme with fully random tags. However, it is non-trivial to directly perform equality-testing on random tags. We use the homomorphic ciphertexts of deterministic tags as random tags to resolve this issue. HE supports equivalence comparison on ciphertexts without decryption. So, we can perform equality-testing on random tags. Moreover, HE can also be used to resist offline brute-force attacks, in that it is probabilistic rather than deterministic. As shown in Figure 3, our scheme deploys a crypto-service provider (CSP) to manage the key pair of HE and perform the homomorphic decryption. When the storage server (SS) performs the tag equality-testing, it sends the homomorphism subtractions between random tags to CSP. The latter decrypts them and returns the comparison results to SS.

The second challenge is how to improve the efficiency of the equality-testing of random tags. The linear time complexity of the tag comparison is unacceptable when there are a large number of random tags. Our solution is to use binary search. Due to the property of HE, SS can get the algebraic relationship between any two random tags by interacting with CSP. Therefore, random tags could be sorted and then stored in an AVL tree. The time complexity of tag comparison can be reduced to logarithmic time by the binary search in the AVL tree. Another advantage of our scheme is that the equality-testing only requires the interaction between SS and CSP, while clients do not need to be involved. As a consequence, our scheme circumvents the limitation in μR-MLE2 [7].

The third challenge is how to resist the popularity tamper attack. As described in Section 2.3, in existing popularity-based deduplication schemes [3] [4], an adversary with only data tags can tamper with data popularity due to the lack of PoW. In the worst case, the adversary can launch the popularity tamper attack to make the target data become popular and then launch offline brute-force attacks to restore data information. Most existing PoW schemes [5] [14] [15] [16] [17] aim at convergent ciphertexts. It is impossible to apply them to resist the popularity tamper attack since the unpopular data are encrypted randomly. Besides, most existing PoW schemes leak data hashes, which will compromise the semantic security of unpopular data. To this end, we design a PoW protocol based on HE to resist the popularity tamper attack without leaking any data information. The idea of our PoW is to let clients compute the hashes of some randomly sampled challenge blocks and then encrypt these hashes with HE. These encrypted hashes are used as the proofs for data ownership. SS can verify whether the proofs are valid through the interaction with CSP. Since the proofs are randomly encrypted, they do not reveal any data information.

4.2 Architecture

As shown in Figure 3, our scheme consists of three entities: clients, a storage server (SS), and a crypto-service provider (CSP).

- The client outsources user data to SS to save local storage overhead. To maintain privacy, the client outsources encrypted data to SS.
- The storage server (SS) provides data storage services for multiple users and performs cross-user deduplication to save storage space.
- The crypto-service provider (CSP) is independent of clients and the storage server. It is responsible for managing the HE key pair. It authenticates clients and distributes the HE public key to SS and all authenticated clients.

Note that CSP can be implemented by the third-party external cryptographic service [29], where a cryptographic server performs some cryptographic operations [30] for applications. Many cryptography-based schemes [2], [30], [31], [32], [33] and well-resourced enterprises (e.g. Facebook [34]) deploy external cryptographic services.

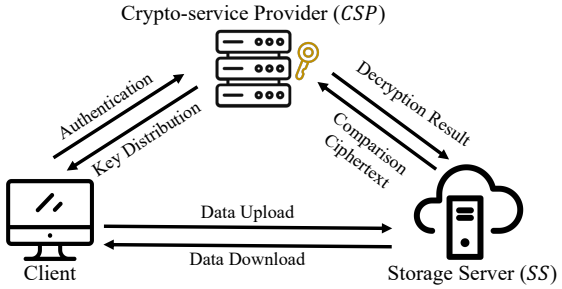


Fig. 3. The architecture of our scheme.

4.3 Threat Model

We assume that CSP is a semi-trusted third party, which is similar to the assumption for the key server in DupLESS [2]. In our scheme, CSP cannot learn any data information, but it needs to store the HE secret key sk securely. If an adversary compromises CSP and learns sk, then the security for all user data degrades to convergent security. Besides, we consider the following two kinds of adversaries.

An honest-but-curious (HBC) storage server SS honestly follows our proposed protocol, but attempts to compromise data confidentiality. It can access all outsourced data and attempts to restore data information.

A malicious client C holds a random tag and a portion of the data content of a target file F_t. It aims to launch the popularity tamper attack to convince SS of its data ownership and tamper with data popularity. Besides, it tries to launch the duplicate faking attack for F_t to compromise the data integrity of other users.

4.4 Security Goal

The security goals of our scheme are as follows.

Data confidentiality. Our scheme provides semantic security and convergent security for unpopular data and popular data, respectively.

Data integrity. Our scheme provides integrity for outsourced data. Any adversaries cannot tamper with user data stored in SS without being detected.

Tag consistency. Our scheme provides tag consistency including *validity* and *security*.

1) *Validity.* Only if two pieces of data are identical can SS interact with CSP to identify that two tags are identical.
2) *Security.* Tags should be indistinguishable from random bit strings of equal length.

Attacks resistance. Our scheme resists the typical attacks in encrypted deduplication, such as brute-force attacks [2], popularity tamper attacks [9], ownership cheating attacks [5], and duplicate faking attacks [10]. Any adversaries cannot launch these attacks to compromise data confidentiality or integrity.

4.5 Definition

Let λ and $\mathcal{M}$ be a security parameter and the message space respectively. We denote by $\mathcal{F}(\{\mathcal{A}_i(\{x_i\})\}) \rightarrow \{a_i\}$ the event that multiple parties $\{\mathcal{A}_i\}$ jointly engage in the protocol $\mathcal{F}$ with the inputs $\{x_i\}$ and outputs $\{a_i\}$. The empty string is denoted by ε.

Our scheme consists of the algorithms and protocols (KG, RTG, PopDet, PoW, Enc, Dec, CV, IV), which we define as follows:

KG$(\lambda) \rightarrow (pk, sk)$. CSP uses a security parameter λ to invoke the key generation algorithm to generate a key pair (pk, sk). After that, we assume that all parties take λ as input in all algorithms and protocols.

RTG$(pk, m) \rightarrow rT$. The random tag generation algorithm takes the public key pk and a message $m \in \mathcal{M}$ as input, and outputs a random tag rT.

PopDet$(SS(rT, T_A), CSP(sk)) \rightarrow (ctr, \varepsilon)$. During the popularity detection protocol, SS inputs a random tag rT and an AVL tree T_A, while CSP inputs the secret key sk. When the protocol concludes, SS outputs the data popularity ctr, while CSP outputs nothing, denoted by the empty string ε.

PoW$(C(pk, F_c), SS(c_a, c_b, P_{set}), CSP(sk)) \rightarrow (\varepsilon, P'_{set}, \varepsilon)$. During the PoW protocol, client C inputs pk and a convergent ciphertext F_c, SS inputs two random seeds (c_a, c_b) and a proof set P_{set} (described in Section 4.6), while CSP inputs sk. When the protocol concludes, C and CSP output nothing, while SS outputs an updated P'_{set}.

Enc$(k_E, m) \rightarrow C_m$. The encryption algorithm takes a key k_E and a message m as input, and outputs a ciphertext C_m.

Dec$(k_E, C_m) \rightarrow m$. The decryption algorithm takes a key k_E and a ciphertext C_m as input, and outputs the message m.

CV$(SS(pk, rT, F_c), CSP(sk)) \rightarrow (f, \varepsilon)$. During the ciphertext verification protocol, SS inputs pk, a random tag rT, and a convergent ciphertext F_c, while CSP inputs sk. When the protocol concludes, SS outputs a flag $f \in \{0, 1\}$ to indicate whether the verification is successful, while CSP outputs nothing.

IV$(k_c, m) \rightarrow f$. The integrity verification algorithm takes a convergent key k_c and a message m as input, and outputs a flag f to indicate whether the verification is successful.

4.6 Construction

Here, we present the constructions of the algorithms and protocols in our scheme.

Key generation (KG). Our scheme follows the key generation algorithm in HE. During the system setup, CSP runs $\mathrm{HE.KeyGen}(\lambda)$ to generate HE key pair (pk, sk).

Random tag generation (RTG). To record data popularity, we generate random tags based on HE. Specifically, to upload an outsourced file F, client C first uses CE to generate a convergent ciphertext $F_c = \mathrm{CE.Enc}(k_c, F)$, where k_c is the convergent key ($k_c = \mathrm{CE.KeyGen}(F)$). Then, C generates a deterministic tag $dT = \mathrm{H}(F_c)$, where $\mathrm{H}(\cdot)$ is a cryptographic hash function. The purpose of using the hash of the convergent ciphertext as the deterministic tag is to prevent duplicate faking attacks [10], which is explained in detail in Section 6.3. After that, C encrypts dT with HE to obtain a random tag $rT = \mathrm{HE.Enc}(pk, dT)$, which can be used for popularity detection and data retrieval.

Popularity detection (PopDet). After the generation of rT, C sends it to SS for popularity detection. SS can determine the algebraic relation between any two random tags by the interactions with CSP. In consequence, all random tags could form an ordered set and be stored in an AVL tree. After receiving rT, SS first finds the root node rT_1 of the AVL tree T_A (see Figure 4). Then it computes a tag equality-testing ciphertext $etc_t = \mathrm{HE.Sub}(rT, rT_1)$ and sends it to CSP. CSP decrypts etc_t with sk, performs a function $\mathcal{I}(\cdot)$ on the decryption result to obtain the tag comparison result $rs_t \in \{-1, 0, 1\}$ (see Equation 1), and returns rs_t to SS. Note that $\mathcal{I}(\cdot)$ is a function that outputs the highest order coefficient of the plaintext polynomial. As described in Section 3.4, we set the plaintext modulus as 3. So, we can get the algebraic relation between data tags by computing the highest order coefficient of the subtraction of them.

$$rs_t = \mathcal{I}(\mathrm{HE.Dec}(sk, etc_t)) = \begin{cases} -1 & rT < rT_1 \\ 0 & rT = rT_1 \\ 1 & rT > rT_1 \end{cases} \quad (1)$$

As shown in Figure 4, SS can use multiple tag equality-testing ciphertexts $\{etc_t\}$ to interact with CSP and then eventually find the node corresponding to rT. Then, SS can detect the popularity ctr (number of owners) for the data corresponding to the random tag of this node. Note that SS will create a new node for rT if it can not match any existing nodes in T_A. The time complexity of the equality-testing for random tags is logarithmic time, and the tag storage structure supports efficient node insertion and deletion due to the property of the AVL tree. Besides, neither the process of the tag comparison nor the node update requires client involvement, so we do not need to assume that the clients are online.

Proof of Ownership (PoW). The random tag cannot be considered as the proof of data ownership, since adversaries may learn deterministic tags of user data (especially for predictable data) and the HE public key pk is publicly known. So, we design a PoW protocol to resist both the ownership cheating attack and popularity tamper attack.

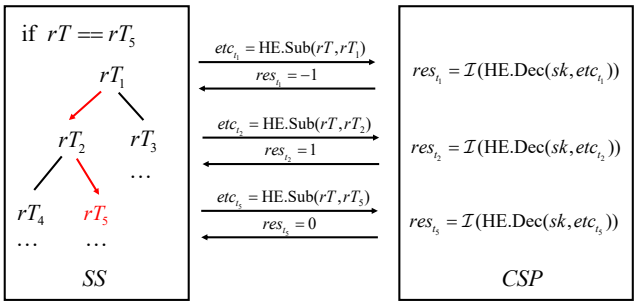


Fig. 4. Tag comparison.

Note that the PoW protocol needs to be run for both the uploads of popular data and unpopular data.

We assume that client C uploads file F, its random tag and convergent ciphertext are rT and F_c respectively. We also assume that this is the first upload of F. After the popularity detection, SS runs PoW with C. Specifically, SS generates two random seeds (c_0, c_1) and sends them to C. Note that the current proof set P_{set} for rT is empty since this is the first upload. C takes (c_0, c_1) as seeds to generate two random sequences of block index $\{c_{01}, c_{02}, ..., c_{0n}\}$ and $\{c_{11}, c_{12}, ..., c_{1n}\}$ (the length of each index sequence is set to be n). Then C divides F_c into fix-sized blocks $\{F_c[i]\}_{i=1}^{l_F}$, where l_F denotes the length of the block set of F_c ($l_F \geq n$). C generates the proofs $p_0 = \text{HE.Enc}(pk, \text{H}(\text{H}(F_c[c_{01}])\|...\|\text{H}(F_c[c_{0n}])))$ and $p_1 = \text{HE.Enc}(pk, \text{H}(\text{H}(F_c[c_{11}])\|...\|\text{H}(F_c[c_{1n}])))$, and then sends (p_0, p_1) to SS. The latter inserts $\{(c_0, p_0), (c_1, p_1)\}$ into the proof set P_{set} for subsequent uploads and then add C into the list of owners.

If another client C' uploads file F' and its random tag is equal to rT, SS will assume that C' may be the second uploader of F and runs PoW with C'. Specifically, SS generates a random coin $b \in \{0, 1\}$ and a new random seed c_2. Then it sends the challenge seeds (c_b, c_2) to C'. C' generates proofs (p'_b, p_2) based on (c_b, c_2) and then sends them to SS. The generation of (p'_b, p_2) is the same as that of (p_0, p_1). SS computes a proof equality-testing ciphertext $etc_p = \text{HE.Sub}(p_b, p'_b)$ and sends it to CSP, where p_b is the proof stored in P_{set} before. CSP returns the comparison result $rs_p = \mathcal{I}(\text{HE.Dec}(sk, etc_p))$ to SS. If rs_p is equal to 0, then C and C' upload the same file and they both pass PoW, then SS increases the owner number of F by one, inserts (c_2, p_2) into P_{set}, and then add C' into the list of data owners. Otherwise, either C or C' uploads a fake proof, SS will treat F and F' as two different files and set their popularity separately. If C_m is the $m-th$ uploader for F, SS randomly selects $c_i \in \{c_0, c_1, ..., c_{m-1}\}$ from P_{set} and a new random seed c_m to run PoW with C_m. The process is the same as that of the second uploader.

In our PoW, SS sends two seeds (c_u, c'_u) to client C, where c_u is selected from the proof set P_{set} and c'_u is freshly generated. C generates two proofs (p_u, p'_u) based on (c_u, c'_u). Note that these seeds and proofs have different roles. The role of (c_u, p_u) is to allow SS to verify that the file of C is consistent with previously stored files and that C has complete data content. Whereas (c'_u, p'_u) is used to increase the size of P_{set}. If only a fixed seed and proof are used in each PoW, the compromise of them will allow the adversary without complete content to pass PoW. If SS randomly generates a seed each time, the verification will be infeasible because SS does not learn convergent ciphertexts of unpopular data and cannot generate valid proofs for verification. Therefore, SS constructs the proof set P_{set} and selects a seed from it for verification during each PoW. Increasing the size of P_{set} decreases the probability that the adversary passes PoW after a certain seed and proof are compromised. The exception is the first uploader C_1. SS uses two freshly generated seeds to run PoW with it since P_{set} is empty at this point. If C_1 uploads fake proofs, SS can detect that during the PoW for subsequent uploaders.

Enc and Dec. After the popularity detection and PoW, SS may require the client to upload encrypted data. Our scheme employs the symmetric encryption algorithm (e.g. AES). If the outsourced data are unpopular, the client performs random encryption to provide semantic security. Otherwise, the client performs CE to encrypt popular data to provide convergent security. Also, the client performs symmetric decryption to restore data.

Ciphertext verification (CV). The ciphertext verification protocol is designed to prevent a malicious client from uploading correct data tags and forged convergent ciphertexts to launch duplicate faking attacks. After receiving random tag rT and convergent ciphertext F_c from the client, SS computes a random tag $rT' = \text{HE.Enc}(pk, \text{H}(F_c))$ and a tag consistency ciphertext $etc_h = \text{HE.Sub}(rT, rT')$, and then sends etc_h to CSP. The latter returns a comparison result $rs_h = \mathcal{I}(\text{HE.Dec}(sk, etc_h))$. If rs_h is equal to 0, then F_c is consistent with rT, and SS outputs a flag $f = 1$ to indicate the success of ciphertext verification. Otherwise, SS outputs $f = 0$ to indicate failure.

Integrity verification (IV). After the client downloads and restores its outsourced file F, it runs the integrity verification algorithm to check data integrity. It computes a convergent key $k'_c = \text{CE.KeyGen}(F)$ and then checks whether k'_c is equal to the original convergent key k_c. If so, it outputs a flag $f = 1$. Otherwise, it outputs $f = 0$.

4.7 The Workflow of the Scheme

The workflow consists of the system setup, data upload, and data download. The specific processes are as follows.

System Setup. CSP runs $\text{KG}(\lambda)$ to generate a system-level HE key pair (pk, sk), sends the public key pk to SS, and keeps the secret key sk secretly. Once a client joins the system, it can obtain pk after authenticating with CSP. SS sets a popularity threshold t for all user data. It is feasible to set different thresholds for different data. Here, we use a uniform threshold t for simplicity. We further discuss this issue in Section 5.6.

Data Upload. As shown in Figure 5, to upload a file F, client C first runs $\text{RTG}(pk, F)$ to get a random tag rT, and then sends rT to SS. SS performs the popularity detection after receiving rT. Specifically, SS inputs rT and AVL tree T_A to run PopDet with CSP. After the popularity detection protocol, SS can find the corresponding node of rT in T_A after multiple equality tests. The storage structure of SS is shown in Figure 6. Each node corresponds to a representative random tag $rT_i (i \in [1, n] \cap \mathbb{Z})$ and a set of

random tags that are equal to rT_i after the HE decryption. The representative tag is used as an index, and it could be any element of the set. SS looks over the current owner number for the node corresponding to rT to detect data popularity. For example, if rT is equal to rT_2 after the HE decryption, then its popularity is 101.

After the popularity detection, SS runs PoW with C. SS increases the owner number by one and adds C into the list of owners only if C passes PoW. If the number of owners for F has not reached the threshold t, it means that F is still unpopular, then SS will set an uploading response urs to up. If the number of owners just reaches t, urs is set to pc. Otherwise, urs is set to p. SS returns the uploading response urs to C.

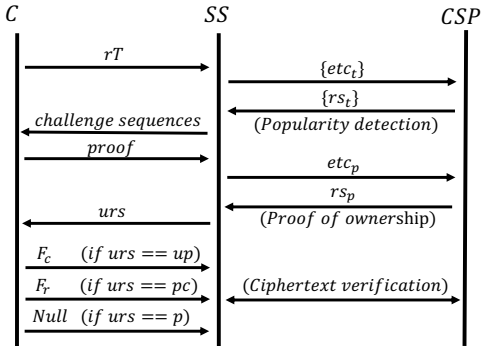


Fig. 5. The workflow for uploading data.

Index	Number of owners	Tag	Ciphertext
rT_1	17	$\{rT_{1_1}, \cdots, rT_{1_{p1}}\}$	$\{F_{r_{1_1}}, \cdots, F_{r_{1_{p1}}}\}$
rT_2	101	$\{rT_{2_1}, \cdots, rT_{2_{p2}}\}$	F_{c_2}
$\cdots$	$\cdots$	$\cdots$	$\cdots$
rT_n	76	$\{rT_{n_1}, \cdots, rT_{n_{pn}}\}$	$\{F_{r_{n_1}}, \cdots, F_{r_{n_{pn}}}\}$

Fig. 6. The storage structure of SS (the threshold t is set to 100).

If urs is equal to up, which means that F is unpopular, C needs to upload a random ciphertext of F. C generates a random key $k_r \leftarrow \{0,1\}^\lambda$ and computes a random ciphertext $F_r = \mathrm{SE.Enc}(k_r, F)$, where $\mathrm{SE.Enc}(\cdot)$ is a symmetric encryption (SE) algorithm. C uploads F_r and stores (rT, k_c, k_r) locally for downloading and restoring F. SS stores rT and F_r for C. SS only deduplicates popular data, while unpopular data are randomly encrypted and cannot be deduplicated. As shown in Figure 6, if the file corresponding to a random tag is unpopular (e.g. rT_1 and rT_n), SS will store all random tags that are equivalent to that random tag along with their corresponding random ciphertexts. If the file is popular, SS only stores one copy of the convergent ciphertext (e.g. F_{c_2}).

If urs is equal to pc, which means that the popularity conversion needs to be performed. C needs to upload a convergent ciphertext F_c. After receiving F_c, SS inputs (pk, rT, F_c) to run the ciphertext verification protocol with CSP to resist duplicate faking attacks. When the protocol concludes, if SS outputs $f = 1$, then the protocol is successful, and all random ciphertexts corresponding to rT will be removed to save storage space. Otherwise, C may upload a fake ciphertext and launch the duplicate faking attack, then SS aborts the popularity conversion process.

If urs is equal to p, which means that F is popular, C does not need to upload a complete encrypted file, while SS just needs to add C to the owner list.

Data Download. To download file F, client C sends random tag rT to SS. If F is unpopular, SS sends random ciphertext F_r to C. The latter uses random key k_r to restore file $F = \mathrm{SE.Dec}(k_r, F_r)$, where $\mathrm{SE.Dec}(\cdot)$ is a symmetric decryption algorithm. Otherwise, if F is popular, SS sends convergent ciphertext F_c to C. C uses convergent key k_c to restore file $F = \mathrm{CE.Dec}(k_c, F_c)$. After that, C runs $\mathrm{IV}(k_c, F)$ to perform the integrity verification. If the output is $f = 1$, then the integrity verification passes. Otherwise, the outsourced data may be tampered with adversaries, and the decryption fails.

5 The Multi-tenant Popularity-based Secure Deduplication Scheme

In this section, we first point out the limitations of the scheme in Section 4. Then, we propose a multi-tenant popularity-based secure deduplication scheme to address these limitations. Hereinafter, we call the scheme in Section 4 the *single-tenant scheme* and call the scheme in this section the *multi-tenant scheme*.

5.1 Limitations of the Single-tenant Scheme

The single-tenant scheme has the following limitations. The first limitation is that the centralized CSP may become an efficiency bottleneck. As the number of users increases, the computation overhead of CSP increases significantly, which makes the system hard to scale up. A straightforward solution is to deploy multiple CSPs with an identical system-wide shared HE key pair. Nevertheless, this key pair will become a single point of failure. If an adversary breaks any one of CSPs and gets the key pair, the semantic security of all unpopular data will be compromised, which is a serious vulnerability. The second limitation is that the single-tenant scheme cannot provide the key rotation. Once the HE key pair is leaked, all tags need to be regenerated, which is inconvenient and computationally expensive.

5.2 Overview of the Multi-tenant Scheme

We introduce MKH-PRE into our single-tenant scheme to design a multi-tenant scheme, which achieves both scalability and updatability. The architecture of the multi-tenant scheme is shown in Figure 7. Users are divided into different tenants. Each tenant deploys a CSP for managing the HE key pair. To avoid the single point of failure, different CSPs hold distinct HE key pairs. Based on the property of MKH-PRE, SS can perform the equality-testing on cross-tenant random tags by interacting with multiple CSPs. Furthermore, the multi-tenant scheme also supports key rotation since MKH-PRE has the proxy re-encryption algorithm.

The threat model and security goal of the multi-tenant scheme are similar to the single-tenant scheme (see Section 4.3 and 4.4). The only difference is that the multi-tenant scheme adds a security goal of forward security. That is, even if an adversary learns original system-level secret keys

and the updated outsourced data, it cannot restore any data information.

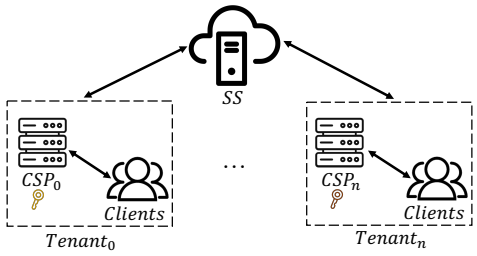


Fig. 7. The architecture of the multi-tenant scheme.

5.3 Definition of the Multi-tenant Scheme

The multi-tenant scheme consists of (KG, MRTG, CPopDet, PoW, Enc, Dec, CV, IV, RKGen, Update). We define MRTG, CPopDet, RKGen, and Update as follows, while the others are the same as the single-tenant scheme.

MRTG$(pk, m) \rightarrow (rT, ID_S)$. The multi-tenant random tag generation algorithm takes the public key pk and a message $m \in \mathcal{M}$ as input, and outputs a random tag rT and an id set ID_S of rT.

CPopDet$(SS(rT, ID_S, T_A), \{CSP_i(sk_i)\}) \rightarrow (ctr, \{\varepsilon\})$. The cross-tenant popularity detection protocol is run by SS and multiple CSPs. SS inputs a random tag rT, its id set ID_S, and an AVL tree T_A, while each CSP_i inputs its secret key sk_i. When the protocol concludes, SS outputs the data popularity ctr, while CSPs output nothing.

RKGen$(sk, sk') \rightarrow rk$. The re-encryption key generation algorithm takes two secret keys (sk, sk') of MKH-PRE as input, and outputs a re-encryption key rk.

Update$(rk_i, \{rT_i\}) \rightarrow \{rT_i'\}$. The update algorithm takes a re-encryption key rk_i and a tag set $\{rT_i\}$ as input, and outputs an updated tag set $\{rT_i'\}$.

5.4 Construction of the Multi-tenant Scheme

Here, we present the constructions of the algorithms and protocols in the multi-tenant scheme.

Key generation (KG). All CSPs run $\mathrm{MP.KeyGen}(\lambda)$ to generate key pairs.

Multi-tenant random tag generation (MRTG). We take client C_i in tenant T_i as an example for illustration. C_i first computes a deterministic tag dT, which is the same as the single-tenant scheme. Then, it uses the public key pk_i of T_i and dT to generate a random tag $rT = \mathrm{MP.Enc}(pk_i, dT)$, whose id set is $\{i\}$.

Cross-tenant popularity detection (CPopDet). After receiving rT and $\{i\}$, SS first finds the root node rT_1 of the AVL tree T_A (see Figure 8). Different from the single-tenant scheme, the id set $\{j\}$ of rT_1 may be not equal to that of rT. If two id sets are the same, then the process of popularity detection is the same as the single-tenant scheme. Otherwise, SS computes a tag equality-testing ciphertext $etc_t = \mathrm{MP.Sub}(rT_1, rT, \{pk_i, pk_j\})$, extracts (etc_{T_i}, etc_{T_j}) from etc_t, and then sends them to CSP_i and CSP_j respectively. Note that the id set of etc_t is $\{i, j\}$. CSP_i and CSP_j respectively run $\mathrm{PartDec}(etc_{T_i}, sk_i)$ and $\mathrm{PartDec}(etc_{T_j}, sk_j)$ to get decryption shares ρ_i and ρ_j, and return them back to SS. The latter invokes $\mathcal{I}(\mathrm{FinDec}(\rho_i, \rho_j))$ to restore the final decryption result $rs_t \in \{-1, 0, 1\}$. Then SS could determine the algebraic relation between rT_1 and rT based on rs_t.

As shown in Figure 8, SS can repeat the above steps to find the corresponding node of rT in T_A, and then obtain the cross-tenant data popularity ctr. If rT cannot match any existing node, SS will create a new node for it. In this case, ctr will be set to 1, if the client passes PoW later. The cross-tenant popularity detection is efficient due to the distributed decryption, which can be demonstrated in Section 7.3.

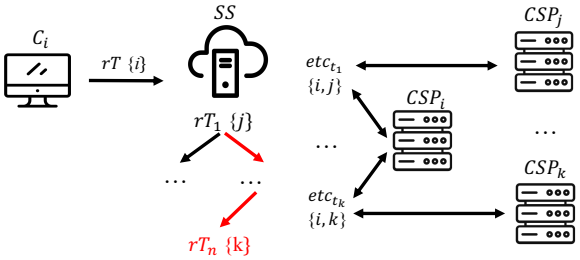


Fig. 8. The equality-testing for tags in the multi-tenant scheme.

Re-encryption key generation (RKGen). CSP_i of tenant T_i inputs an original secret key sk_i and a new secret key sk_i' to run $\mathrm{MP.RKGen}(sk_i, sk_i')$, and outputs a re-encryption key rk_i.

Update. After receiving a re-encryption key rk_i from CSP_i, SS first finds all random tags whose id set is $\{i\}$ to form a tag set $\{rT_i\}$. Then, SS runs $\mathrm{MP.ReEnc}(rk_i, rT_i)$ on all elements in $\{rT_i\}$ and then obtains all updated tags to form an updated tag set $\{rT_i'\}$.

The constructions of other algorithms and protocols are the same as the single-tenant scheme.

5.5 The Workflow of the Multi-tenant Scheme

The workflow consists of the system setup, data upload/download, and key update. The specific processes are as follows.

System Setup. All CSPs run $\mathrm{KG}(\lambda)$ to generate key pairs and send their public keys to SS. The key pair of CSP_i is (pk_i, sk_i). When a client C_i of tenant T_i joins the system, it can obtain the public key pk_i after authenticating with CSP_i. SS sets a uniform popularity threshold t for all user data from different tenants. The setting of t will be further discussed in Section 5.6.

Data Upload/Download. When a client C_i of tenant T_i outsources its file F to SS, it runs $\mathrm{MRTG}(pk_i, F)$ to generate a random tag rT and its id set $\{i\}$, and then sends $(rT, \{i\})$ to SS. The latter runs CPopDet with multiple CSPs to get the popularity ctr_i of rT and then runs PoW (the same as the single-tenant scheme) with C_i. If C_i passes PoW, SS increases the data popularity by one and adds C_i into the list of owners. The following steps are the same as the single-tenant scheme. Specifically, SS returns an uploading response $urs \in \{up, pc, p\}$ according to the current data popularity, while C_i uploads data in different ways according to urs (see Section 4.7). The steps for data download are the same as the single-tenant scheme.

Key Update. If HE secret keys are compromised by adversaries, then the security of unpopular data will be reduced to convergent security. Hence, CSPs rotate HE key pairs as a regular routine to protect against key compromise. In the multi-tenant scheme, CSPs of different tenants can update their keys independently. If CSP_i performs the key update, it invokes $\mathrm{KG}(\lambda)$ to generate a new key pair (pk_i', sk_i'). Then, it uses the original secret key sk_i and the new secret key sk_i' to invoke $\mathrm{RKGen}(sk_i, sk_i')$ to get a re-encryption key rk_i, and then sends rk_i to SS. The latter runs Update to get an updated tag set $\{rT_i'\}$. After the key update, the updated tags are protected by the new key pair (pk_i', sk_i'), while the original key pair (pk_i, sk_i) is invalid.

5.6 Discussion

Here, we discuss some issues of our schemes.

Security protection. Compared to the state-of-the-art popularity-based encrypted deduplication schemes [3], [4], our schemes need a weaker security assumption and provide stronger security protection. First, our schemes deploy a semi-trusted third party (CSP) rather than a fully trusted party. Second, our schemes perform PoW to resist the popularity tamper attack, while existing schemes are vulnerable to it. Besides, we shrink the size of the data that needs to be securely stored. Recall that existing schemes need to maintain confidentiality for all deterministic tags, while our schemes just need to protect HE secret keys. This feature reduces the size of sensitive data that needs to be protected, while reducing the risk of sensitive data leakage.

Popularity threshold. We use a uniform threshold t to classify data as popular or unpopular for simplicity. Actually, SS can set various thresholds based on real scenarios. Note that the number of thresholds needs to be controlled to prevent uncontrollable boom [3]. After SS publishes candidate thresholds, users can choose a preferred threshold for their outsourced data. During the data upload, the client needs to state the threshold it chooses for outsourced data. The popularity of the same file with distinct thresholds needs to be recorded separately [3]. Otherwise, security will be easily compromised.

User revocation. For cloud storage systems, user revocation is an important issue. In our schemes, to revoke an owner U_r of an unpopular file F_r, SS removes U_r from the owner list, decreases the owner number of F_r by one, and then deletes the random tags and random ciphertexts uploaded by U_r. Since unpopular data are encrypted by random keys, a revoked user cannot restore any data information even if it intercepts the unpopular data uploaded by other users. To efficiently revoke an owner of a popular file, we can tweak our schemes by using the encryption schemes in REED [33] to encrypt popular data. Specifically, when uploading popular data, clients use *convergent all-or-nothing transform* (CAONT [35]) to transform data into trimmed packages and stubs. Then the trimmed packages are encrypted with deterministic encryption algorithms and are used to be deduplicated to save storage space, while the stubs are encrypted with random keys and are used to perform the efficient key update and user revocation. As a result, our scheme can revoke users' access rights by re-encrypting the stubs of popular data. The user revocation is efficient since the size of the stub is small (0.78% of the original data [33]), and the security can be reduced to the property of CAONT.

6 SECURITY ANALYSIS

In this section, we analyze the security of our schemes under our threat model (see Section 4.3).

6.1 Data Confidentiality

Both single-tenant and multi-tenant schemes encrypt popular data with CE, so they are provided with convergent security, as analyzed in [10]. As a result, our security analysis focuses on the semantic security of unpopular data. We take the single-tenant scheme as an example to analyze the security for unpopular data, while the analysis for the multi-tenant scheme is similar. The specific security model is defined by a security game $\mathbb{G}$ played between adversary $\mathcal{A}$ and challenger $\mathcal{C}$ below, which respectively model an honest-but-curious SS and an honest client.

Setup. $\mathcal{C}$ runs $\mathrm{KG}(\lambda)$ to generate a HE key pair (pk, sk) and sends pk to $\mathcal{A}$.

Query. $\mathcal{A}$ adaptively issues queries on selected messages. For a queried message m_i, $\mathcal{C}$ generates a random symmetric key k_i, invokes $\mathrm{SE.Enc}(k_i, m_i)$ to generate a symmetric ciphertext C_{m_i}, and then returns C_{m_i} to $\mathcal{A}$. Note that $\mathcal{C}$ generates a fresh symmetric key for each query.

Challenge. $\mathcal{A}$ outputs two messages m_0 and m_1 (the sizes of them are equal). $\mathcal{C}$ picks a bit $b \in \{0, 1\}$ and generates a random symmetric key k_c. Then, $\mathcal{C}$ computes a random tag $rT_b = \mathrm{RTG}(pk, m_b)$ and a random ciphertext $C_b = \mathrm{SE.Enc}(k_c, m_b)$. Besides, $\mathcal{C}$ generates two random seeds (c_0, c_1), a convergent ciphertext C_b' of m_b, and then uses (c_0, c_1, C_b', pk) to generate a proof set P_b for m_b by locally emulating PoW. Finally, $\mathcal{C}$ returns (rT_b, C_b, P_b) to $\mathcal{A}$.

Guess. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ of which message is chosen by $\mathcal{C}$.

We define the advantage $\mathrm{Adv}_{\mathcal{A}}$ of $\mathcal{A}$ in the above game $\mathbb{G}$ as

$$\mathrm{Adv}_{\mathcal{A}} = |\Pr[b = b'] - 1/2|, \tag{2}$$

where the probability is over the random bits used by the challenger and the adversary.

Definition 1. *Our scheme provides semantic security for unpopular data only if for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists a negligible function $\mathrm{negl}(\lambda)$ such that*

$$\mathrm{Adv}_{\mathcal{A}} \leq \mathrm{negl}(\lambda).$$

Theorem 1. *If SE and HE are semantically secure, then our scheme provides semantic security for unpopular data.*

Proof. We prove by defining a sequence of indistinguishable security games, each differs slightly from the previous.

$\mathbb{G}_0$: is identical to $\mathbb{G}$.

$\mathbb{G}_1$: The challenger $\mathcal{C}$ replaces the random ciphertext C_b by a random bit string C_r, whose length is equal to C_b. The indistinguishability of $\mathbb{G}_1$ to $\mathbb{G}_0$ follows from the semantic security of SE.

$\mathbb{G}_2$: The challenger $\mathcal{C}$ replaces the random tag rT_b by a random bit string rT_r of equal length. Then $\mathcal{C}$ replaces all

elements in the proof set P_b by random bit strings of equal length to output a set P_r, whose elements are all uniformly random. The indistinguishability of $\mathbb{G}_2$ to $\mathbb{G}_1$ follows from the semantic security of HE.

In $\mathbb{G}_2$, the information available to $\mathcal{A}$ are (rT_r, C_r, P_r), which are all uniformly random values. Therefore, in $\mathbb{G}_2$, $\mathcal{A}$ could only outputs b' randomly. So, its probability advantage $\mathrm{Adv}_{\mathcal{A}}$ is negligible. Since $\mathbb{G}_2$ and $\mathbb{G}_0$ are indistinguishable, the probability advantage for $\mathcal{A}$ in $\mathbb{G}_0$ is negligible. □

The security of the multi-tenant scheme. The security analysis for the multi-tenant scheme is similar to the single-tenant scheme. The only difference is that the semantic security of unpopular data is reduced to MKH-PRE. If there is a PPT adversary $\mathcal{A}$ compromises the semantic security for unpopular data, then we can construct another adversary $\mathcal{A}'$ to break the semantic security of MKH-PRE or SE.

Remarks. Note that $\mathcal{A}$ could learn the owner number of a file corresponding to a random tag (see Figure 6). But this information does not affect the semantic/convergent security of unpopular/popular data, since all outsourced data are semantically/convergently encrypted.

6.2 Attack Resistance

We analyze the resistance to popularity tamper attacks (PTA), duplicate faking attacks (DFA), and brute-force attacks (BFA) for our schemes. Note that the resistance to PTA is the same as the resistance to ownership cheating attacks. The analysis for these attacks in the single-tenant and multi-tenant schemes is the same, so we do not differentiate them.

Definition 2. *Let ϵ be a security parameter, if the probability P_{pass} that a malicious client passes PoW satisfies that $P_{pass} \leq 2^{-\epsilon}$, then our schemes achieve the resistance to PTA.*

Theorem 2. *Assuming that there are no collisions in the hash function, and the malicious client owns a certain portion p of a target file F, then our schemes could achieve the resistance to PTA if the number n of challenge blocks in PoW satisfies that $n \geq -\epsilon \cdot \frac{\ln 2}{\ln p}$.*

Proof. Assuming that the malicious client $\mathcal{A}_e$ knows p of F. In other words, given any block B of F to $\mathcal{A}_e$, the probability that it owns B is p. For any challenge block, the probability that $\mathcal{A}_e$ owns it is p. If $\mathcal{A}_e$ has the challenge block, it can use its hash to compute the proof of ownership. Otherwise, it can only guess the hash of the challenge block. Suppose that the length of the hash is K bits, then the probability that $\mathcal{A}_e$ guesses the correct hash of a challenge block is 2^{-K}. So, we can deduce that the probability that $\mathcal{A}_e$ passes PoW is $P_{pass} = \left(p + (1-p) \cdot 2^{-K}\right)^n$. Generally, K is about 128 or 256. So, $(1-p) \cdot 2^{-K}$ can be considered a negligible value. Therefore, we have $P_{pass} \approx p^n$. To resist PTA, we need to ensure that $P_{pass} \leq 2^{-\epsilon}$. Through simple arithmetic, we can deduce that the minimum bound of n is $-\epsilon \cdot \frac{\ln 2}{\ln p}$. □

Theorem 3. *Assuming that there are no collisions in the hash function, then our schemes could achieve resistance to DFA.*

Proof. If a malicious client $\mathcal{A}_e$ successfully launches DFA, it needs to let SS output $f = 1$ during the ciphertext verification. In other words, its forged convergent ciphertext $F_c'(F_c' \neq F_c)$ needs to satisfy $\mathrm{H}(F_c') = \mathrm{H}(F_c)$, where F_c is an honestly generated ciphertext. In this case, $\mathcal{A}_e$ finds collisions in the hash function, which breaks the assumption. □

Theorem 4. *If both SE and HE are semantically secure, then our schemes could resist BFA for unpopular data and unpredictable popular data.*

Proof. All tags, proofs, and communication transcripts between SS and CSP are encrypted by HE. Based on the semantic security of HE, any adversary can not launch BFA on them to restore data information. Unpopular data are encrypted with SE, so the BFA on them can not work if SE is semantically secure. Popular data are encrypted with CE, so their semantic security can be achieved if they are unpredictable. If the popular data are predictable, then they are vulnerable to BFA [2]. □

6.3 Data Integrity

The honest-but-curious SS can access all outsourced data. However, based on the HBC assumption, it does not maliciously modify, delete, or destruct user data. This is a reasonable assumption since SS needs to maintain its industry reputation and accountability. For a malicious client $\mathcal{A}_e$, according to Theorem 3, it can not successfully launch DFA to compromise data integrity without being detected. For legitimate users, they can use random keys or convergent keys to restore outsourced data and then verify the data integrity via integrity verification. Note that the key generation of CE is a hash function. If the data integrity is compromised, and the integrity verification can be passed, then the adversary finds collisions in the hash function, which breaks the assumption.

6.4 Tag Consistency

Based on the collision resistance of the hash function and the decryption correctness of HE, our data tags achieve validity. Assuming that a malicious client $\mathcal{A}_e$ constructs the same tag rT for two distinct files F_1 and F_2. Based on the decryption correctness of HE, we have $\mathrm{H}(F_c^1) = \mathrm{H}(F_c^2)$, where F_c^1 and F_c^2 are the convergent ciphertexts of F_1 and F_2 respectively. Since $F_1 \neq F_2$, then the probability that $F_c^1 = F_c^2$ is negligible. Therefore, C finds a collision in the hash function, which breaks the assumption. Besides, since random tags are encrypted by HE, they are indistinguishable from random bit strings of equal length based on the semantic security of HE, which proves that our schemes provide security for tags.

6.5 Forward Security

The forward security of the multi-tenant scheme can be easily drawn from the security of the proxy re-encryption in MKH-PRE. Based on the security of the proxy re-encryption, the original keys are useless after the key rotation. The original keys and new keys are both uniformly and randomly chosen from the key space, and the updated tags are consistent with the new keys and are indistinguishable from random bit strings based on the security of MKH-PRE.

TABLE 1
Security comparison for popularity-based encrypted deduplication schemes.

	Confidentiality	Resistance to PTA	Resistance to DFA	Scalability	Updatability
Stanek *et al*. [3]	✔	✗	✔	✗	✗
PerfectDedup [4]	✔	✗	✔	✗	✗
Single-tenant scheme	✔	✔	✔	✗	✗
Multi-tenant scheme	✔	✔	✔	✔	✔

6.6 Security Comparison

We compare the security properties of our schemes with state-of-the-art schemes [3] [4]. As shown in Table 1, existing schemes can not provide resistance to PTA, since they do not introduce the PoW protocol. Compared to the single-tenant scheme, the multi-tenant scheme further provides scalability and updatability based on the properties of MKH-PRE. Note that the confidentiality in Table 1 refers to the semantic security of unpopular data and the convergent security of popular data.

7 EVALUATION

We implement prototypes of our schemes, which consist of three entities: the client C, the storage server SS, and the crypto-service provider CSP. We use MD5 as the tag generation function for deterministic data tags and SHA-256 as the key generation function for convergent keys. The symmetric encryption algorithm used in our prototypes is AES-128-CTR. These cryptographic primitives are all implemented based on the OpenSSL library [36]. The codes of HE and MKH-PRE are implemented in different ways. HE can be implemented more efficiently than MKH-PRE, since it does not need to support the proxy re-encryption or homomorphic computation on ciphertexts under different key pairs. The code of HE is implemented based on the open-source library SEAL [37] [38]. The MKH-PRE is implemented based on the schemes of [39] and [40], and the code of MKH-PRE is implemented based on FNTRU [41]. All entities in our prototypes are implemented in C++.

We also implement the prototypes of existing schemes [3], [7]. We call the scheme proposed by Stanek *et al*. [3] *Sdedup* for short. We instantiate the bilinear map in the prototypes of [3] and [7] with type-F pairing provided by the PBC library [42]. Our experiments run on a machine equipped with a quad-core 2.7GHz Intel Core-i7-7500U, 5400RPM SATA hard disk, and 8GB RAM, and installed with 64-bit Ubuntu 20.04.12.

We first evaluate the time overheads of data upload in three scenarios: unpopular data upload, popularity conversion, and popular data upload. Then, we evaluate the overheads of the tag equality-testing and key update to demonstrate their efficiency. Finally, we use a real-world dataset to evaluate the storage overhead.

7.1 Comparison with Sdedup

One of the differences between our schemes and Sdedup is that we use HE/MKH-PRE to generate random tags and introduce a PoW protocol, while Sdedup uses the convergent threshold cryptosystem. Thus, we compare the performance of the cryptography tools in our schemes and Sdedup. The performance comparison is shown in Figure 9.

We select a test file F_t and analyze the performances of various cryptography tools by evaluating the time overheads of generating the random tag and proofs of ownership. Note that the overheads of the generations of the random tag and proofs all depend only on the file hash, not the file size, so we do not specify the size of F_t. The PoW protocols in single-tenant and multi-tenant schemes are denoted by PoW-S and PoW-M in Figure 9 respectively since different homomorphic encryption algorithms are used. The time overhead of the threshold cryptosystem in Sdedup is 31.1 ms, while the counterparts of HE/MKH-PRE and PoW-S/PoW-H are 0.4/3.3 ms and 2.8/7.9 ms respectively. Therefore, compared with Sdedup, the cryptography tools used in both single-tenant and multi-tenant schemes are more efficient. Besides, we can find that the speed of HE encryption is significantly faster than that of MKH-PRE.

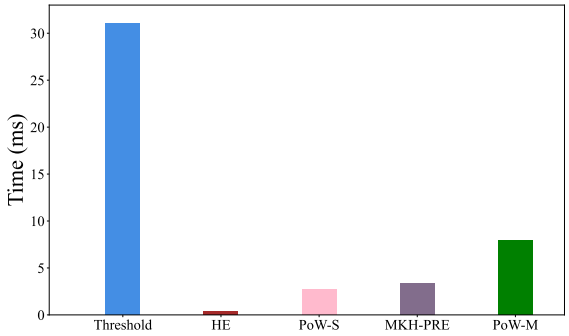


Fig. 9. The performance comparison among the threshold cryptosystem, HE, MKH-PRE, and PoW.

Then, we compare the time overheads for uploading unpopular data among Sdedup, our single-tenant and multi-tenant schemes with multiple test files ranging in size from 1 MiB to 500 MiB. The result is shown in Figure 10. The time overheads include the overheads of random tag generation, data encryption, PoW (only in our schemes), and data communication. When uploading a file of 1 MiB, our single-tenant and multi-tenant schemes can respectively reduce 51.9% and 17.4% time overheads compared with Sdedup. When the file size becomes 500 MiB, these two ratios become 1.0% and 0.9%. We can find that the time overheads for uploading unpopular files with relatively large sizes are close among these three schemes, although our schemes seem to have slightly less overhead. The reason is that the overheads of computing file hashes and data encryption account for about 90% of the total, and these two parts in Sdedup and our schemes are similar. The processes

of popularity conversion and uploading popular data in Sdedup and our schemes are basically the same, so the time overheads of these two processes are also close.

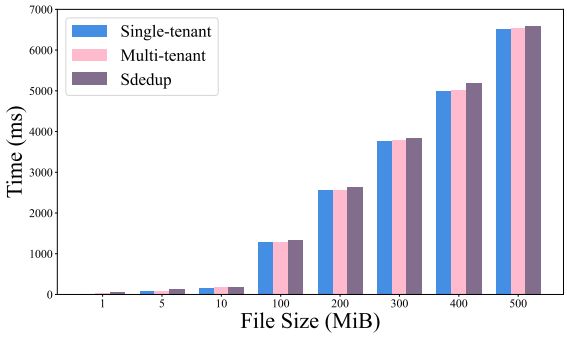


Fig. 10. The performance comparison between our schemes and Sdedup when uploading unpopular data.

7.2 Performance Comparison under Three Scenarios

We take the single-tenant scheme as an example to evaluate the time overhead of each part in three scenarios. The situation of the multi-tenant scheme is similar. The size of the test file is 10 MiB. We assume that SS has already stored 10,000 random tags. There are six main components in the process of data upload: random tag generation, PoW, popularity detection, ciphertext verification, data encryption (the random encryption for unpopular data), and communication.

As shown in Figure 11, the overheads of random tag generation respectively account for 60.2%, 64.4%, and 95.2% of the total overheads in these three scenarios, which are the most time-consuming. The overheads of the random tag generation include the overheads of generating the convergent ciphertext, deterministic tag, and random tag. The overheads of PoW and popularity detection in these scenarios only account for 1.8% $\sim$ 2.6% and 0.3% $\sim$ 0.6% respectively, which only have little impact on performance. Compared with the unpopular data upload, the popularity conversion has the overhead of ciphertext verification instead of data encryption. The overheads of the ciphertext validation include the overheads of generating the deterministic tag, once HE encryption, and once HE decryption, which are close to the overheads of the data encryption. So, these two scenarios have close overheads. It is obvious that the popular data upload has the lowest overhead since it requires neither data encryption nor ciphertext verification. The communication overhead in popular data upload is also very low since the whole data do not need to be uploaded. As a result, the overhead for the popular data upload is reduced by 39.2% and 35.8% respectively compared with the unpopular data upload and popularity conversion.

We use multiple test files to evaluate the time overheads for data upload in three scenarios. In Figure 12, we can see that the overhead for uploading data will become lower once the data become popular. When the size of the outsourced data is 500 MiB, uploading popular data can save 34.6% and 32.2% of the time overhead compared with uploading unpopular data and popularity conversion.

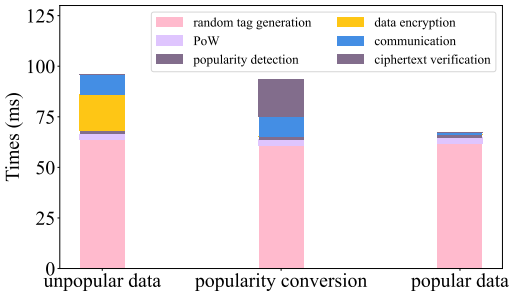


Fig. 11. The time overheads of each component in each scenario.

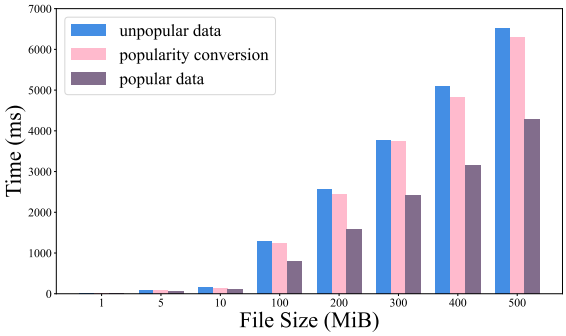


Fig. 12. The performance comparison in three scenarios.

7.3 Performance of Tag Equality-testing

We compare the time overheads of the linear and binary search for random tags. The result is shown in Figure 13. The performance evaluation of linear and binary searches is under the single-tenant scheme. We insert 1,000 random tags continuously into the linear structure and the AVL tree. The binary search based on the AVL tree is more efficient. The insertion of the 1,000th random tag takes only 15 ms using binary search, while the delay of the linear search is 99 ms.

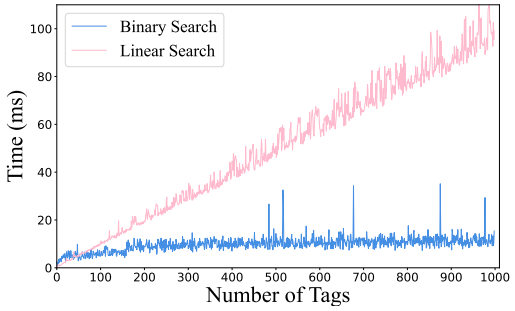


Fig. 13. Linear and binary search for random tags.

The overheads of inserting 1,000 random tags continuously in our single-tenant/ multi-tenant schemes and the static/dynamic schemes in [7] are shown in Figure 14. We can find that the single-tenant and multi-tenant schemes respectively have the highest and lowest overheads for the tag equality-testing, while the static and dynamic schemes in [7] are in the middle. The use of the MKH-PRE mildly

increases the overhead of tag equality-testing. But continuously inserting 1,000 random tags into the AVL tree in multi-tenant schemes only needs 71 ms, which is also very efficient. The distributed decryption in MKH-PRE can improve the performance of the tag equality-testing in the multi-tenant scheme. The decryption processes of multiple CSPs are performed in parallel, which ensures that the multi-keys decryption does not bring significant time overhead. Note that the fluctuation in Figure 14 is mainly due to the instability of the decryption time and the adjustment of the tree height when inserting nodes. Compared with [7], the superiority of our schemes is that our tag equality-testing does not need the involvement of clients and does not require the assumption that the clients are online when performing the tag equality-testing. During the evaluation of the static/dynamic schemes in [7], we set up a client that is always online.

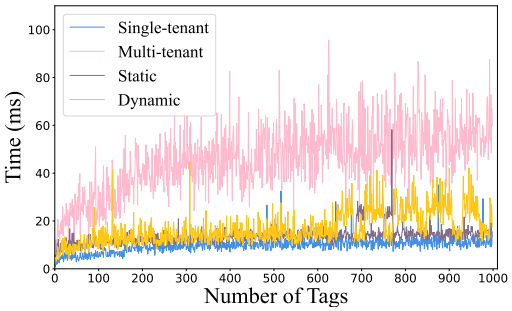


Fig. 14. Inserting 1,000 random tags continuously into our schemes and the structure in [7].

We also evaluate the overhead of each part in the tag equality-testing. Specifically, we evaluate the overheads of inserting 10, 100, 200, 300, 400, and 500 tags, respectively. Figure 15 shows the evaluation result in the single-tenant scheme. The overheads of the tag equality-testing include the overheads of the subtraction of multiple HE ciphertexts (denoted as HE.Sub), the HE decryption (denoted as HE.Dec), the adjustment of the AVL tree, and the communication overhead. The overheads for the HE decryption and communication dominate the total. These overheads account for more than 90% of the total. Figure 16 shows the evaluation result in the multi-tenant scheme. The overhead for the HE decryption (MP.Dec) accounts for more than 94% of the total. The reason is that the homomorphic decryption in MKH-PRE is significantly slower than HE. Besides, we can find that the communication overhead does degrade the performance in the single-tenant scheme, but in the multi-tenant scheme, the performance bottleneck of the tag equality-testing is the homomorphic decryption.

7.4 Performance of Key Update

The performance of the key update in the multi-tenant scheme is shown in Figure 17. We can find that the key update is efficient. Update 500 random tags only needs less than 2.6 s. The reason is that our key update is transparent to users, and the process of the re-encryption only consists of twice polynomial multiplications. The re-encryption is

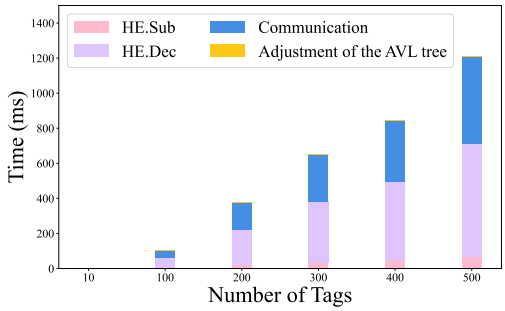


Fig. 15. The overhead of each part in the tag equality-testing (single-tenant scheme).

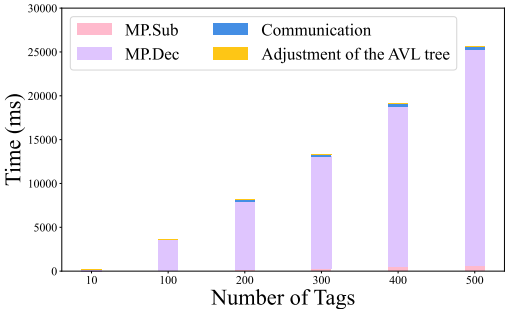


Fig. 16. The overhead of each part in the tag equality-testing (multi-tenant scheme).

implemented based on NTRU [28] [40], in which the polynomial coefficients are relatively small, and the polynomial multiplications can be efficiently performed using the Fast Fourier Transform (FFT) [43].

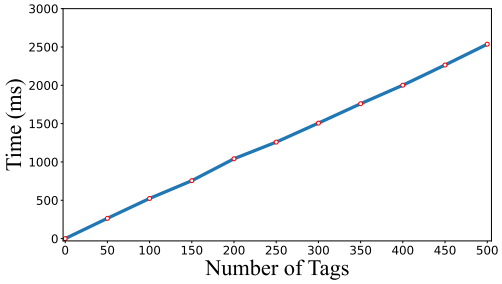


Fig. 17. The performance of key update.

7.5 Storage Efficiency

We evaluate the storage efficiency of our schemes based on a real-world dataset: *PB dataset* [44], which contains the metadata of 679,515 unique torrents from The Pirate Bay, collected on December 5th, 2008. The dataset does not provide the granularity of torrent contents, so we consider each torrent to correspond to one file for measurement. As analyzed in [3], this simplification does not positively impact the evaluation results. We sum the number of "seeders" (peers already having the whole file) and "leechers"

(peers having only part of the file, but intending to get the whole file in the future) of a file as the data popularity. The files with zero size and zero popularity are removed, then we can get a dataset consisting of 442,332 unique files. We randomly select 200, 400, 600, 800, and 1000 files from the dataset to evaluate storage efficiency. The storage overheads of our single-tenant and multi-tenant schemes are almost the same, so we do not differentiate them. We compare our schemes with a baseline scheme *non-dedup*, which does not deduplicate any duplicate files. Besides, we set the popularity threshold t to 50, 100, and 200 to evaluate its impact on storage efficiency.

Figure 18 shows the evaluation result for storage overhead. As the number of selected files increases, our schemes save storage overhead more significantly compared with non-dedup. When 200 files are stored, the storage overheads for non-dedup and our schemes are 3674 MB (non-dedup), 1424 MB (t=50), 1915 MB (t=100), and 2364 MB (t=200) respectively. Our schemes achieve storage savings of 35.7%-61.2%. When 1000 files are stored, the storage savings can be increased to 41.9%-66.5%. Moreover, we can find that a smaller threshold t can result in higher storage efficiency. This is because a smaller t can lead to more popular files, while our schemes save storage overhead by deduplicating popular files.

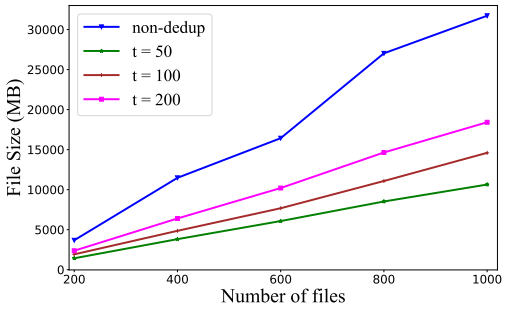


Fig. 18. Storage overheads of our schemes.

8 RELATED WORK

After MLE [10] and DupLESS [2] are proposed, researchers present many novel encrypted deduplication schemes. Liu *et al*. [11], [12] and Yu *et al*. [13] design encrypted deduplication schemes without additional independent servers. Besides, Shin *et al*. [45] propose decentralized server-aided encryption for secure deduplication to alleviate the single point of failure in DupLESS. Li *et al*. [46] [47] propose defense schemes for frequency analysis attacks in encrypted deduplication systems. Zhao *et al*. [48] propose the updatable MLE (UMLE), which allows the efficient update of the encrypted files stored in the cloud server. SGXDedup [49] is proposed to speed up encrypted deduplication via Intel SGX. Yang *et al*. [50] and Xu *et al*. [51] propose access control schemes for encrypted deduplication. Yu *et al*. [52] and Zhang *et al*. [53] propose encrypted deduplication schemes against side-channel attacks. R-MLE2 [8] and μR-MLE2 [7] are proposed to provide lock-dependent security for secure deduplication. None of the above schemes considers data popularity. The state-of-the-art popularity-based secure deduplication schemes are [3] and [4]. As described in Section 2.3 and 6.6, these schemes need to deploy trusted third parties, are vulnerable to popularity tamper attacks, and cannot provide scalability and updatability, while our schemes address these limitations.

9 CONCLUSION

In this paper, we first propose a single-tenant popularity-based encrypted deduplication scheme with fully random tags. We use HE to generate random tags, avoiding storing deterministic tags to record data popularity. Besides, we reduce the time complexity of tag equality-testing by the binary search in the AVL tree. We also design a PoW protocol to resist the popularity tamper attack. For scalability and key rotation, we expand our single-tenant scheme to a multi-tenant scheme by introducing MKH-PRE. In the multi-tenant scheme, users in different tenants use different HE key pairs to generate data tags, while the server could record the cross-tenant data popularity. The multi-tenant scheme also supports key rotation based on the proxy re-encryption of MKH-PRE. We implement prototypes of our schemes and evaluate their performances. The results show that our schemes have high storage efficiency and achieve efficient data encryption and key update.

10 ACKNOWLEDGEMENT

This work is supported by National Natural Science Foundation of China (61972215,62172238, 61972073); the National Key R&D Program of China (2018YFA0704703); Natural Science Foundation of Tianjin (20JCZDJC00640; The Fundamental Research Funds for the Central Universities of China.

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 *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002.

[2] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Usenix Conference on Security*, 2013.

[3] J. Stanek and L. Kencl, "Enhanced secure thresholded data deduplication scheme for cloud storage," *IEEE Transactions on Dependable & Secure Computing*, vol. PP, no. 4, pp. 1–1, 2016.

[4] P. Puzio, R. Molva, M. Önen, and S. Loureiro, "Perfectdedup: Secure data deduplication," Tech. Rep., 2015. [Online]. Available: https://doi.org/10.1007/978-3-319-29883-2_10

[5] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *ACM Conference on Computer & Communications Security*, 2011, p. 491.

[6] J. Xu, E. C. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," *ASIA CCS 2013 - Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pp. 195–206, 2013.

[7] T. Jiang, X. Chen, Q. Wu, J. Ma, W. Susilo, and W. Lou, "Secure and Efficient Cloud Data Deduplication with Randomized Tag," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 3, pp. 532–543, 2017.

[8] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages," in *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, ser. Lecture Notes in Computer Science, R. Canetti and J. A. Garay, Eds., vol. 8042. Springer, 2013, pp. 374–391.

[9] G. Ha, H. Chen, C. Jia, R. Li, and Q. Jia, "A secure deduplication scheme based on data popularity with fully random tags," in *20th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2021, Shenyang, China, October 20-22, 2021*. IEEE, 2021, pp. 207–214.

[10] M. Bellare and S. Keelveedhi, *Message-Locked Encryption and Secure Deduplication*. Springer Berlin Heidelberg, 2013.

[11] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 874–885.

[12] J. Liu, L. Duan, Y. Li, and N. Asokan, "Secure deduplication of encrypted data: Refined model and new constructions," pp. 374–393, 2018.

[13] C. Yu, "POSTER: efficient cross-user chunk-level client-side data deduplication with symmetrically encrypted two-party interactions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1763–1765.

[14] J. B. Alís, R. D. Pietro, A. Orfila, and A. Sorniotti, "A tunable proof of ownership scheme for deduplication using bloom filters," in *IEEE Conference on Communications and Network Security, CNS 2014, San Francisco, CA, USA, October 29-31, 2014*. IEEE, 2014, pp. 481–489.

[15] L. González-Manzano, J. M. de Fuentes, and K. R. Choo, "asepow: A proof of ownership mechanism for cloud deduplication in hierarchical environments," in *Security and Privacy in Communication Networks - 12th International Conference, SecureComm 2016, Guangzhou, China, October 10-12, 2016, Proceedings*, vol. 198. Springer, 2016, pp. 412–428.

[16] 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," *Concurr. Comput. Pract. Exp.*, vol. 32, no. 3, 2020.

[17] J. Dave, A. Dutta, P. Faruki, V. Laxmi, and M. S. Gaur, "Secure proof of ownership using merkle tree for deduplicated storage," *Autom. Control. Comput. Sci.*, vol. 54, no. 4, pp. 358–370, 2020.

[18] G. M. Adel'son-Vel'skii and E. M. Landis, "An algorithm for organization of information," in *Doklady Akademii Nauk*, vol. 146, no. 2. Russian Academy of Sciences, 1962, pp. 263–266.

[19] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, M. Mitzenmacher, Ed. ACM, 2009, pp. 169–178.

[20] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 13:1–13:36, 2014.

[21] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 10031, 2016, pp. 3–33.

[22] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*. ACM, 2012, pp. 1219–1234.

[23] C. Peikert and S. Shiehian, "Multi-key FHE from lwe, revisited," in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 9986, 2016, pp. 217–238.

[24] Z. Brakerski and R. Perlman, "Lattice-based fully dynamic multi-key FHE with short ciphertexts," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 9814. Springer, 2016, pp. 190–213.

[25] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. ACM, 2019, pp. 395–412.

[26] S. Yasuda, Y. Koseki, R. Hiromasa, and Y. Kawai, "Multi-key homomorphic proxy re-encryption," in *Information Security - 21st International Conference, ISC 2018, Guildford, UK, September 9-12, 2018, Proceedings*, ser. Lecture Notes in Computer Science, vol. 11060. Springer, 2018, pp. 328–346.

[27] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Springer, 2012, pp. 868–886.

[28] J. Hoffstein, J. Pipher, and J. H. Silverman, "Ntru: A ring-based public key cryptosystem," in *Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings*. Springer, 2006, pp. 267–288.

[29] T. Berson, D. Dean, M. K. Franklin, D. K. Smetters, and M. Spreitzer, "Cryptology as a network service," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2001, San Diego, California, USA*. The Internet Society, 2001.

[30] R. W. F. Lai, C. Egger, D. Schröder, and S. S. M. Chow, "Phoenix: Rebirth of a cryptographic password-hardening service," in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*, E. Kirda and T. Ristenpart, Eds. USENIX Association, 2017, pp. 899–916.

[31] A. Everspaugh, R. Chatterjee, S. Scott, A. Juels, and T. Ristenpart, "The pythia PRF service," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 547–562.

[32] R. W. F. Lai, C. Egger, M. Reinert, S. S. M. Chow, M. Maffei, and D. Schröder, "Simple password-hardened encryption services," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, W. Enck and A. P. Felt, Eds. USENIX Association, 2018, pp. 1405–1421.

[33] 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, pp. 9:1–9:30, 2017. [Online]. Available: https://doi.org/10.1145/3032966

[34] A. Muffet, "Facebook: Password hashing and authentication," Available: https://video.adm.ntnu.no/ pres/54b660049af94/, 2015.

[35] M. Li, C. Qin, and P. P. C. Lee, "Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *2015 USENIX Annual Technical Conference, USENIX ATC '15, July 8-10, Santa Clara, CA, USA*. USENIX Association, 2015, pp. 111–124.

[36] "Openssl project," Available: http://www. openssl.org/.

[37] K. Laine, "Simple encrypted arithmetic library 2.3.1." https://www.microsoft.com/en-us/research/uploads/prod/ 2017/11/sealmanual-2-3-1.pdf., 2017.

[38] "Seal 2020. microsoft seal (release 3.6)." https://github.com/ Microsoft/SEAL. Microsoft, 2017.

[39] R. Li, C. Jia, and Y. Wang, "Multi-key homomorphic proxy re-encryption scheme based on ntru and its application," *Tongxin Xuebao/Journal on Communications*, vol. 42, no. 3, pp. 11 – 22, 2021.

[40] D. Nuñez, I. Agudo, and J. López, "Ntrureencrypt: An efficient proxy re-encryption scheme based on NTRU," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015*. ACM, 2015, pp. 179–189.

[41] "Fntru," https://github.com/vernamlab/FNTRU.

[42] "Pbc library-the pairing-based cryptography library, [online]." Available: https://crypto.stanford.edu/pbc/.

[43] J. Hermans, F. Vercauteren, and B. Preneel, "Speed records for NTRU," in *Topics in Cryptology - CT-RSA 2010, The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5985. Springer, 2010, pp. 73–88.

[44] D. H. Fabio Hecht, Thomas Bocek, "The pirate bay 2008-12 dataset," Available: http://www.csg.uzh.ch/publications/data/piratebay/.

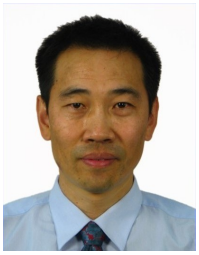
[45] Y. Shin, D. Koo, J. Yun, and J. Hur, "Decentralized Server-aided Encryption for Secure Deduplication in Cloud Storage," *IEEE Transactions on Services Computing*, vol. 1374, no. c, pp. 1–14, 2017.

[46] J. Li, P. P. Lee, C. Tan, C. Qin, and X. Zhang, "Information Leakage in Encrypted Deduplication via Frequency Analysis," *ACM Transactions on Storage*, vol. 16, no. 1, 2020.

[47] J. Li, Z. Yang, Y. Ren, P. P. Lee, and X. Zhang, "Balancing storage efficiency and data confidentiality with tunable encrypted dedu-

plication," *Proceedings of the 15th European Conference on Computer Systems, EuroSys 2020*, 2020.
[48] Y. Zhao and S. S. M. Chow, "Updatable Block-Level Message-Locked Encryption," Tech. Rep., 2019.
[49] Z. Yang, P. P. C. Lee, T. Chinese, and H. Kong, "Accelerating Encrypted Deduplication via SGX Yanjing Ren and Jingwei Li , University of Electronic Science and Technology of China ; This paper is included in the Proceedings of the," *Usenix Atc*, 2021.
[50] H. Tang, Y. Cui, C. Guan, J. Wu, J. Weng, and K. Ren, "Enabling ciphertext deduplication for secure cloud storage and access control," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016*. ACM, 2016, pp. 59–70.
[51] R. Xu, J. Joshi, and P. Krishnamurthy, "An integrated privacy preserving attribute-based access control framework supporting secure deduplication," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 2, pp. 706–721, 2021.
[52] C. Yu, S. P. Gochhayat, M. Conti, and C. Lu, "Privacy aware data deduplication for side channel in cloud storage," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 597–609, 2020.
[53] Y. Zhang, Y. Mao, M. Xu, F. Xu, and S. Zhong, "Towards thwarting template side-channel attacks in secure cloud deduplications," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 3, pp. 1008–1018, 2021.

Guanxiong Ha received the M.S. degree in computer science and technology from Nankai University, Tianjin, P. R. China, in 2021. He is currently working toward the Ph.D. degree from the College of Cyber Science, Nankai University, Tianjin, P. R. China. His research interests include cloud data security and applied cryptography.

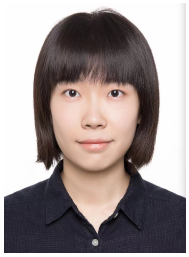


Chunfu Jia A Ph.D. supervisor, a professor and the Head of Department in the Department of cyber Sciences, Nankai University. His main research interests include network and system security, cryptography application and malware analysis.

Yixuan Huang was born in 1999. She is currently a Master Candidate at the College of Cyber Science, Nankai University, Tianjin, China. Her research interests mainly include homomorphic encryption.

Hang Chen is studying for a master's degree at the College of Cyber Science, Nankai University, Tianjin, China. Her main research interests are cryptography and data deduplication.

Ruiqi Li received the Ph.D. degree in computer science and technology from Nankai University in 2021. He is currently an Assistant Professor with the College of Safety Science and Engineering, Civil Aviation University of China. His current research interests include fully homomorphic encryption, lattice-based cryptography and cloud computing security.



Qiaowen Jia is currently a Ph.D candidate in Institute of Software, University of Chinese Academy of Sciences. Her research interest includes concurrent program and software verification.