Blockchain-Based Deduplication and Integrity Auditing over Encrypted Cloud Storage

Mingyang Song, Zhongyun Hua, Yifeng Zheng, Hejiao Huang, Xiaohua Jia, *Fellow, IEEE*

Abstract—Cloud computing promises great advantages in handling the exponential data growth. Secure deduplication can greatly improve cloud storage efficiency while protecting data confidentiality. In the meantime, when data are outsourced to the remote cloud, there is an imperative need to audit the integrity. Most existing works only consider the support for either secure deduplication or integrity auditing. Recently, there have been some research efforts aiming to integrate secure deduplication with integrity auditing. However, prior works are unsatisfactory in that they suffer from the leakage of ownership privacy and forgeability of auditing results for low-entropy data. In this paper, we propose a new scheme that delicately bridges secure deduplication and integrity auditing in encrypted cloud storage. In contrast with prior works, our scheme protects the ownership privacy and prevents the cloud service provider from forging the auditing results for low-entropy data. Furthermore, we propose a blockchain-based mechanism that helps to ensure key recoverability and reduce local storage cost of keys. Formal analysis is provided to justify the security guarantees. Experiment results demonstrate the modest performance overhead of our scheme.

Index Terms—Secure deduplication; integrity auditing; ownership privacy; blockchain.

1 Introduction

In the big data era, large volumes of digital data are exponentially produced everyday and the data growth tendency is accelerated. To save storage space, individuals and enterprises are more and more willing to outsource their data to cloud servers for storage. As reported by International Data Corporation, the amount of data stored in cloud will increase to 175ZB by 2025 [1]. Fortunately, almost 75% of the data in cloud are duplicate [2]. Therefore, data deduplication can greatly improve storage efficiency and thus is very meaningful. Nowadays, data deduplication techniques have been employed by many cloud service providers (CSPs) such as Google Drive [3], Dropbox [4] and Mozy [5].

From the view of users, they are more concerned about the security of their outsourced data. Data encryption is the most straightforward way of protecting data confidentiality and thus users are prone to encrypt their data before uploading them to cloud servers for storage. This requires the CSP to own the ability to deduplicate the encrypted data. In the past decades, many secure deduplication schemes have been proposed [6–9] and they can directly remove the duplicate copies in the encrypted domain without accessing the data content. Besides the data confidentiality, users are also concerned about the integrity of their data stored in the cloud, since data loss may happen in the cloud due to some objective exceptions such as hardware failure [10]. Remote data integrity auditing is an effective and widely used solution to help users check the integrity of their outsourced data [11, 12]. Therefore, it is important to simultaneously support secure deduplication and integrity auditing for a cloud storage system.

The previous integrity auditing schemes cannot be directly applied to secure deduplication systems. This is because an authentication tag for each data block should be computed to help a user check the integrity and users holding different secret keys generate different authentication tags for the same data block. When auditing an identical file, the authentication tags generated by different users cause considerable storage consumption. For example, a 4GB file uploaded by 100 users will require 2.5GB (62.5% of the data) additional storage to store the authentication tags for auditing. However, the storage overhead can reduce to 25.6MB if the CSP can perform deduplication to the authentication tags [13]. Nowadays, there already exist some efforts [10, 14–16] that combine existing secure deduplication and integrity auditing techniques. However, these studies only focus on deduplicating the authentication tags to improve the storage efficiency of cloud servers and do not consider the leakage of ownership privacy (OP) and forgeability of auditing results for low-entropy data.

Deduplicating both ciphertext and authentication tags leaks the OP of users, which means that a third-party auditor (TPA) and other users can get the information about 1) which files are owned by a user and 2) which users own an identical file. Although it is inevitable to leak OP to the cloud, we should ensure that no other parties except the cloud can know the OP. It has become a common practice to use a TPA proposed in [17] to help users audit data integrity [18–21]. The leakage of OP occurs when different users delegate the TPA to audit an identical file. Specifically, in previous methods [10, 14–16, 22–24], the users ($\mathcal{U}_\mathcal{A}$, $\mathcal{U}_\mathcal{B}$) owning an identical file use an identical deterministic file

- *Mingyang Song, Zhongyun Hua, Yifeng Zheng and Hejiao Huang are with School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen 518055, China (E-mail: songmingyang2022@gmail.com; huazhongyun@hit.edu.cn; yifeng.zheng@hit.edu.cn; hjhuang@aliyun.com).*
- *Xiaohua Jia is with department of Computer Science, City University of Hong Kong, Hong Kong 518057, China, and also with School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen 518055, China (E-mail: csjia@cityu.edu.hk).*
(Corresponding authors: Zhongyun Hua; Yifeng Zheng.)

tag to index the file in the CSP and send the same file tag to the TPA. The deterministic file tag leaks the first type OP of $\mathcal{U}_\mathcal{A}$ and $\mathcal{U}_\mathcal{B}$. Meanwhile, both the deterministic file tag and other auditing information leak the second type OP that $\mathcal{U}_\mathcal{A}$ and $\mathcal{U}_\mathcal{B}$ own the same file. For example, when two companies $\mathcal{U}_\mathcal{A}$ and $\mathcal{U}_\mathcal{B}$ store the same contract in the cloud. Then the TPA can snoop on that there is cooperation between $\mathcal{U}_\mathcal{A}$ and $\mathcal{U}_\mathcal{B}$, which is an indirect privacy leakage of $\mathcal{U}_\mathcal{A}$ and $\mathcal{U}_\mathcal{B}$. The leakage of OP has a similar impact as the side-channel attacks. When the auditing logs are publicly accessible to users for verifying the auditing results, a user can also snoop on the OP of other users from public auditing logs. To the best of our knowledge, there does not exist prior work that can support auditing and secure deduplication, while addressing the leakage of OP.

In the previous secure deduplication schemes supporting auditing [14, 16], the auditing results of the low-entropy data are unreliable. They use the message-locked encryption (MLE) key to generate authentication tags for authentication tag deduplication. However, the MLE is inherently deterministic [7] and its secret key is derived from the plaintext data itself (e.g., the hash value of plaintext data). When outsourcing a low-entropy file F that is obtained from a small message dictionary $S = \{F_1, ..., F_n\}$, the CSP can get the MLE key of F after n offline key generation attempts. Therefore, the cloud server can deduce the MLE key of low-entropy data such as the electronic medical records [15]. Using the deduced MLE key, the cloud server can further forge the authentication tags to deceive the TPA even if the data is damaged or lost [15]. Thus users cannot get reliable auditing results. As a result, previous studies that simultaneously support secure deduplication and integrity auditing mainly focus on deduplicating the authentication tags and do not consider the leakage of OP and forgeability of auditing results for low-entropy data caused by the deduplication of authentication tags.

To address these security issues, in this paper, we propose an auditable and secure deduplication scheme to protect OP and ensure reliable auditing results. To protect OP, we propose a new auditing technique with file tag randomization and re-signature strategies. Specifically, different users use different public keys and different random file tags to delegate auditing tasks to the TPA even for the same file, and the cloud server uses different re-signature keys and the same authentication tags to generate different proofs to the TPA for different users. To ensure reliable auditing results, we introduce multiple key servers into our system to participate in the generation of the signing key for authentication tags, following the state-of-the-art schemes [25–28]. Since the CSP can not obtain the secret value shared among key servers, it cannot deduce the signing key of authentication tags even for low-entropy data. Thus, the CSP cannot forge the authentication tags to deceive the TPA when the audited data is lost.

Besides the aforementioned security issues, the duplicate-faking attacks (DFA) launched by the initial uploader can destroy the association of the stored file and its file tag such that the subsequent uploaders owning the same file obtain a falsified copy [6]. Thus, the DFA should also be considered for a secure deduplication cloud storage system. Previous studies resist DFA by verifying the association of ciphertext and its file tag [6, 16], which is no longer available in our scheme since our scheme uses random file tags. Our scheme is designed to resist DFA by checking the association between the ciphertext of the initial uploader and that of the subsequent uploader through the authentication tags. Besides, the number of encryption keys linearly increases with the number of data being uploaded to the cloud in secure deduplication, which results in intensive key management overhead for users. Previous studies in [29–32] encrypt these encryption keys and outsource them to the cloud, but do not consider the risk of damage. To solve this issue, our scheme uses the blockchain to record the blind signatures of the data hash values returned from the key servers. Then the user only needs to store the seed of random number generator and can recover the encryption keys from the blind signatures on-chain. We also use the blockchain to record the auditing logs to solve the trust issues between users and the TPA.

The novelty and contributions of this paper are summarized as follows.

- We propose an auditable and secure deduplication scheme, which is the first time to protect the OP of users and ensure reliable auditing results of low-entropy data. Our scheme is designed to also consider the security issues raised in previous schemes such as the resistance of DFA.
- We introduce the blockchain into our scheme to manage the encryption keys. This can greatly reduce the key storage cost of users and guarantee the key recoverability.
- We conduct a comprehensive evaluation to theoretically prove and experimentally verify the properties of our scheme. Comparison results show that our scheme can protect the OP of users and ensure reliable auditing results with modest performance.

We organize the remainder of this paper as follows. Section 2 reviews the related works and Section 3 formulas the problem that we address in this study. Section 4 describes our method in detail and Section 5 proves the correctness and security of our scheme. Section 6 evaluates the performance of our scheme. Finally, we draw the conclusion in Section 7.

2 Related Work

2.1 Secure Deduplication

Secure deduplication is a useful technique to improve the storage efficiency of cloud while protecting the confidentiality of outsourced data. Traditional encryption methods are unavailable in secure deduplication, since an identical file will be encrypted into different ciphertexts by different users, which cannot be deduplicated. To address this problem, Douceur $et\ al.$ introduced the convergent encryption (CE) [33], which uses the hash value of outsourced data as the encryption key. Inspired by CE, Bellare $et\ al.$ proposed the MLE for the first time [6].

However, the MLE inherently suffers from brute-force attacks [7]. Thus Bellare $et\ al.$ [7] proposed the scheme DupLESS that uses a key server to participate in key generation [7]. Then the cloud needs to launch online brute-force

attacks by interacting with the key server instead of offline brute-force attacks. Nowadays, many schemes use a key server to enhance security [34, 35]. At present, most secure deduplication studies focus on client-side deduplication using file tags. This can mitigate the communication overhead and reduce the local computational cost, since subsequent file owners do not need to compute and upload an identical ciphertext. However, the client-side deduplication requires a reliable method to do the Proofs of Ownership (PoW) and easily suffers from DFA [6, 36]. To solve these, Halevi $et\ al.$ introduced a novel PoW method based on Merkle tree [37], which can prevent adversaries deceiving the CSP using only a file tag. Besides, some schemes were proposed to defense DFA [6, 38] in the client-side deduplication.

These traditional secure deduplication studies only consider data confidentiality. However, when outsourcing data to a cloud server, users are also concerned about data integrity, since data loss may happen in the cloud due to some objective reasons such as hardware failure. Thus, supporting auditing in a secure deduplication scheme is meaningful in practical applications.

2.2 Public Integrity Auditing

Ateniese $et\ al.$ [39] first considered the public auditing in a provable data possession (PDP) model, where the homomorphic linear authentication tags are used for auditing data integrity. In their subsequent work [40], the authors proposed a dynamic version of the prior PDP scheme. Later, Wang $et\ al.$ [41] proposed a public verifiable PDP scheme by employing a fully reliable TPA to reduce users' burden of auditing. Concerning the data recovery, Juels $et\ al.$ [42] proposed a novel primitive proof of retrievability (PoR) and then Shacham $et\ al.$ [43] improved it using publicly verifiable homomorphic authentication tags built from Boneh-Lynn-Shacham (BLS) signature technique [44]. Inspired by these works, many public auditing schemes have been proposed [17, 45] using the homomorphic signature technique. However, directly applying these previous auditing schemes in secure deduplication system will cause considerable storage costs to store lots of redundant authentication tags for an identical file.

2.3 Data Deduplication and Integrity Auditing

In 2013, Yuan $et\ al.$ proposed the first cloud storage system scheme that simultaneously supports data auditing and deduplication [22]. To improve the storage efficiency of CSP, this scheme performs deduplication on both the plaintext and authentication tags. However, it requires all users to compute and upload the authentication tags for an identical file, which sacrifices the local computational efficiency of users. Xu $et\ al.$ used blockchain to develop a deduplication scheme that supports plaintext auditing [23]. The scheme uses the blockchain to monitor the behavior of the TPA by recording auditing logs on-chain. However, it can only perform deduplication and auditing over plaintext, and doesn't consider the confidentiality of the outsourced data.

Li $et\ al.$ proposed the first cloud storage system supporting auditing and deduplication over encrypted data by using a fully trusted proxy server to generate the authentication tags [10]. But it is practically costly to implement such a fully trusted proxy server over insecure public networks. Later, Liu $et\ al.$ first proposed such a cloud storage system without a fully trusted proxy server [14]. This scheme achieves authentication tags deduplication using the MLE key as the signing key of authentication tags. However, it requires users to stay online to participate in auditing, which causes extra cost and inconvenience to users in practice. Besides, the scheme cannot guarantee that the auditing result of low-entropy data is reliable. To solve this defect, Gao $et\ al.$ proposed a new scheme [15], which generates authentication tags using the private key of the initial uploader, rather than the MLE key. However, this scheme can only guarantee the reliability of auditing results for the initial uploader, and that for the subsequent uploaders still cannot be guaranteed. Therefore, for these cloud storage systems supporting auditing and deduplication over encrypted data, the reliability of auditing results for low-entropy data hasn't been fully solved, yet.

Additionally, all the above schemes have been developed based on a strong assumption that the TPA is completely reliable, which is practically difficult. To solve this, Yuan $et\ al.$ proposed a secure deduplication scheme supporting auditing using the blockchain [24], which is a simple combination of the secure deduplication and auditing. It uses the smart contract to execute auditing tasks and fair arbitration. However, this scheme does not preform deduplication to authentication tags. When auditing an identical file, the authentication tags generated by different users cause considerable storage consumption to CSP. Besides, all the owners of an identical file should encrypt the file and upload the file to CSP, and also need to compute their own authentication tags for auditing. This causes large duplicate computations and communication costs. Tian $et\ al.$ [16] use the blockchain to record the file index table and auditing logs. This scheme uses two CSPs to solve the problem of single-point failure, which increases the economic cost and causes inconvenience to users. Users should stay online to interact with the cloud to update the authentication tags regularly. Besides, the above two schemes do not consider the leakage of OP and the forgeability of auditing results for low-entropy data.

In conclusion, previous cloud storage systems supporting auditing and deduplication over encrypted data mainly focus on the storage efficiency of the cloud, and do not consider the leakage of OP and the forgeability of auditing results for low-entropy data. Besides, some previous systems require users to stay online to participate in the auditing process, which goes against the goal of delegating auditing tasks to a TPA. In this paper, we aim to propose a new scheme to completely solve the above issues.

3 Problem Formulation

In this section, we first present the system model, threat model and design goals of our scheme and then give an overview of our scheme.

3.1 System Model

Table 1 summarizes the important notations used in this paper. The system model of our scheme includes five enti-

TABLE 1: Notations used in this paper.

Notation	Description
F	The original plaintext to be outsourced.
C	The ciphertext of F stored in the cloud.
c_i	The i^{th} block of ciphertext C.
$pk^{(u,f)}$	The public key of user $\mathcal{U}$ for delegating auditing.
$rk^{(u,f)}$	The re-signature key of $\mathcal{U}$ for file F.
$\beta^{(u,f)}$	The random file tag of F generated by $\mathcal{U}$.
$C^{(u)}$	The ciphertext of challenged blocks encrypted by $\mathcal{U}$.
z	The signing key of authentication tags.
p, q	Two large prime numbers.
$\mathbb{Z}_N$	A residue class ring.
$\mathbb{G}_1, \mathbb{G}_2$	Two multiplicative cyclic groups.
k	The length of encryption key.
θ	The seed for generating random numbers.
d	The number of sectors in a data block.
I	The number of key servers.
t	The threshold of key servers cryptosystem.
$\boldsymbol{H_1}(\cdot), \boldsymbol{H_2}(\cdot), \boldsymbol{H_3}(\cdot)$	Three hash functions.
$\boldsymbol{\pi_1}(\cdot,\cdot), \boldsymbol{\pi_2}(\cdot,\cdot)$	Two pseudo-random functions.
$\boldsymbol{e}(\cdot,\cdot)$	Bilinear pairing $\boldsymbol{e}$: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.
$\lvert\xi\rvert$	The bit length of arbitrary element ξ.

ties: users, CSP, TPA, key servers, and blockchain. We define them as follows:

- The users outsource their encrypted data to the CSP. They also delegate the TPA to check the integrity of the outsourced data. Since different users may outsource an identical file, all users can be divided into the initial uploader that outsources a new file to the CSP, and the subsequent uploader that outsources an existing file.
- The CSP provides storage services to users. It performs secure deduplication, receives the challenging information from the blockchain, and publishes the proof information on-chain during the process of auditing.
- The TPA receives auditing delegation from a user and generates challenging information for auditing. It publishes the challenging information on-chain and obtains the proof information from the blockchain. It also verifies the proof information and publishes the auditing result on-chain.
- The key servers interact with users to perform blind signature protocol and publish the intermediate data (i.e., blind signatures of data hash values) on-chain.
- The blockchain is a public distributed ledger system with various nodes. It records the blind signatures of data hash values in the key generation process and all auditing logs including the challenging information, proofs and results.

3.2 Threat Model

Based on the description of the system model in Section 3.1, we give the specific description of the threat model as follows.

- **Semi-honest users.** Users snoop on the OP of other users from the auditing logs on the blockchain since these data on-chain are publicly accessible to all entities and contain the information about file tag and file owner. A user also attempts to deduce the encryption keys of other users from the on-chain blind signature values.
- **Semi-honest CSP.** The stored data may be lost due to hardware failures. To maintain its commercial reputation, the CSP attempts to deceive the auditor using forged files and authentication tags when the files are lost. The CSP may also snoop on the encryption keys using on-chain blind signature values or launch brute-force attacks to obtain plaintext content.
- **Semi-honest TPA.** The TPA can execute the predefined rules honestly. But it may also snoop on the OP of users from the auditing logs.
- **Semi-honest key servers.** The key servers interact with users to help them generate encryption keys through a blind signature protocol. They publish the blind signatures of data hash values on-chain. Key servers are curious about the encryption keys of users.

Besides, the CSP or the key servers would not collude with any user for the reason of the commercial reputation. As claimed in [25, 46–48], we also hold the same assumption that the CSP can compromise several key servers less than a predefined number (i.e., t).

3.3 Design Goals

In this paper, we construct a blockchain-based secure deduplication scheme with auditing, which achieves the following design goals.

- **Functionality.** The scheme has two basic functions **deduplication** and **auditing**. This means that it allows the CSP to deduplicate both the ciphertext and authentication tags and users to audit the integrity of their outsourced data. The deduplication and integrity auditing should be supported at the same time.
- **Security.** The security goals of our scheme includes the **data confidentiality**, **reliability of auditing results**, **OP protection** and **resistance of DFA**. (1) The data confidentiality indicates that the scheme should ensure the data confidentiality of different kinds of data, including both high-entropy data and low-entropy data. (2) The reliability of auditing results means that the CSP cannot deceive the TPA by forging the ciphertext and its corresponding authentication tags when the real ciphertext is lost, especially for the low-entropy data. (3) The OP protection means that other entities except the CSP cannot get the information that which files are owned by a user and which users own an identical file. (4) The resistance of DFA means that the scheme can protect subsequent uploaders from losing their data under DFA.
- **Convenience for users.** The scheme can reduce the local storage cost of encryption keys, and does not require users to stay online to participate in the auditing process. Only the initial uploader needs to encrypt the whole file and generate authentication tags.

3.4 Overview of Our Scheme

This section presents an overview of our method and discusses how to achieve the design goals.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3237221

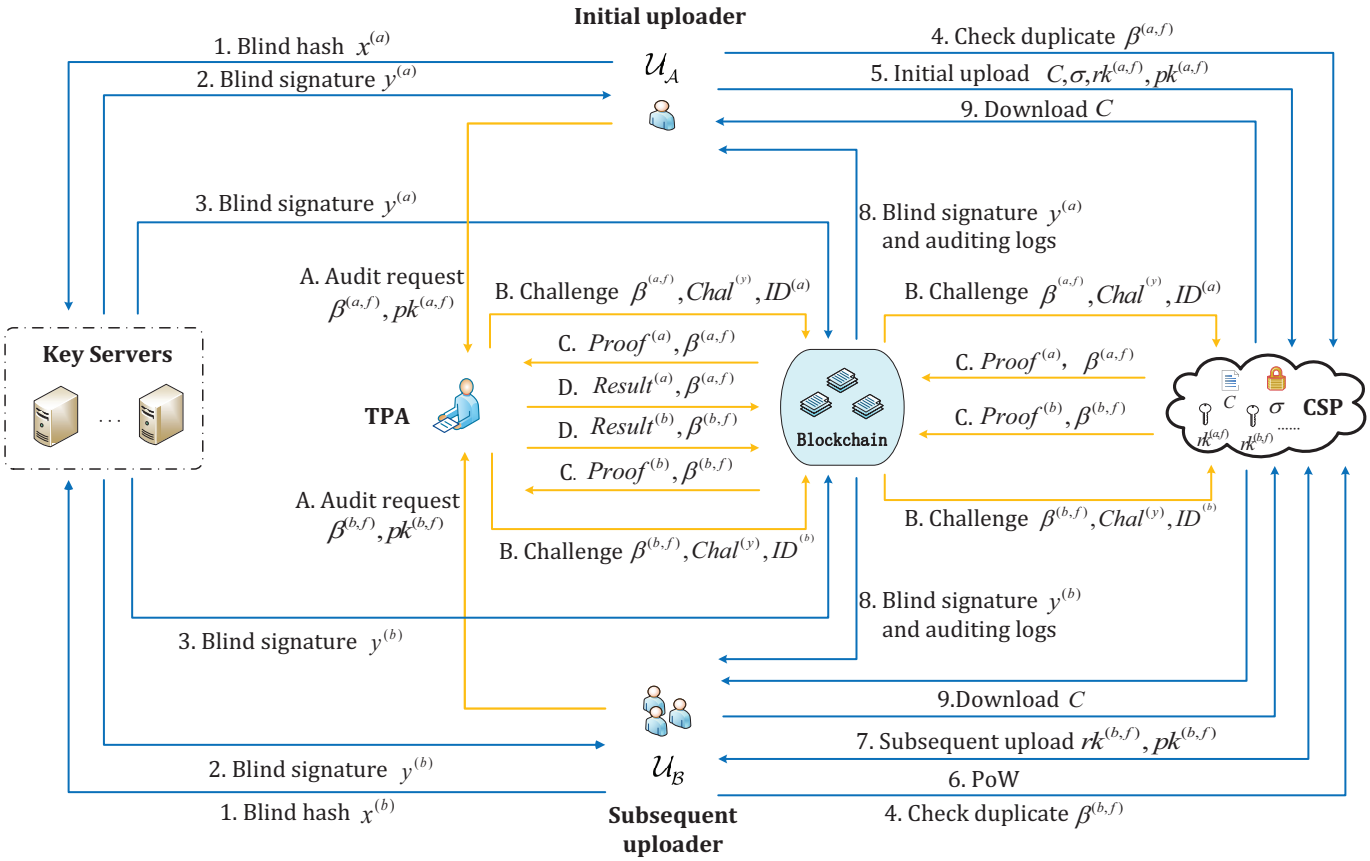


Fig. 1: The system model and workflow of the proposed scheme.

3.4.1 Overview

Fig. 1 shows the system model and workflow of our scheme. Before uploading a file, all users need to generate encryption keys by performing the blind signature protocol with key servers (the steps 1 and 2), and the key servers publish the blind signatures of data hash values on the blockchain (the step 3). During the data uploading process, a user needs to upload a random file tag to the CSP (the step 4) and the CSP detects duplication using the file tag. If the outsourced file does not exist, the user is an initial uploader and uploads the ciphertext, the authentication tags, the re-signature key and the public key (the step 5). Otherwise, the user is a subsequent uploader and performs PoW with the CSP to prove file ownership (the step 6). Then the user uploads the re-signature key and the public key (the step 7). For plaintext data recovery, all users first download the auditing logs and the blind signatures from the blockchain (the step 8). Then the user verifies the auditing result locally. If the outsourced file is integral, the user recovers the encryption keys from the blind signatures, downloads the ciphertext from the CSP (the step 9) and decrypts the ciphertext using the encryption keys.

During the auditing process, a user sends the file tag and the public key to the TPA and entrusts the TPA to audit the file (the step A). The TPA generates challenging data and publishes the file tag, challenging data, and user ID on the blockchain (the step B). After obtaining the challenging data, the file tag, and the user ID from the blockchain, the CSP uses the authentication tags, the re-signature key and the ciphertext to generate an integrity proof, and publishes the proof and the file tag on the blockchain (the step C). After obtaining the proof from the blockchain, the TPA verifies the proof locally and publishes the auditing results on the blockchain (the step D). Finally, the user can obtain the auditing result from the blockchain.

3.4.2 Design Goals Achievement

By design, our scheme can achieve all the design goals. It is obvious that it can achieve both secure deduplication and auditing. It can protect the data confidentiality of both high-entropy data and low-entropy data by adopting multiple key servers to participate in the key generation process. Then the encryption key (i.e., the signature value of data hash value signed by key servers) consists of the hash value of plaintext data and a secret value shared among multiple key servers, which can resist the brute-force attacks. The CSP only stores one copy of the ciphertext and authentication tags. Although the CSP stores a re-signature key for each file owner, it occupies little storage space and thus has high storage efficiency. For the computation cost of users, only the initial uploader needs to compute and upload the ciphertext and the authentication tags, which greatly reduces the computational and communication burden of subsequent uploaders. Besides, through the design of authentication tags, our scheme ensures that users can stay offline in auditing processes. The above design goals are easy to achieve. However, for the OP protection, reliability

of auditing results, resistance to DFA and recoverability of encryption keys, these goals should be specifically designed as follows.

OP protection. For previous studies, the OP leakage happens when a user uses a deterministic file tag to delegate an auditing task to the TPA, or different users use the same file tag to delegate auditing tasks to the TPA. To protect the OP, we propose a new auditing technique including file tag randomization and re-signature strategies. The file tag randomization strategy is to address the first situation. The file tag randomization strategy follows the idea of Yang *et al.* [28] scheme, which has been proven to have high security. In their scheme, they use the random file tag to resist brute-force attacks in the scenario of multi-domain deduplication. Although the design idea is similar, our focused problem and specific design are different from theirs. In our design, a random file tag is generated by multiplying the deterministic file tag with a random item. Then only the CSP with the secret large prime number can determine whether two different random file tags correlate to an identical file. Therefore, the OP is not leaked from file tags.

The re-signature strategy is used to address the second situation. Specifically, as depicted in Fig. 1, the initial uploader $\mathcal{U}_\mathcal{A}$ uploads the ciphertext C, the authentication tags σ, and the re-signature key $rk^{(a,f)}$ to the CSP. Meanwhile, the subsequent uploader $\mathcal{U}_\mathcal{B}$ generates its own re-signature key $rk^{(b,f)}$ for C and sends it to the CSP. Then the CSP stores the ciphertext C, the authentication tags σ and different re-signature keys $(rk^{(a,f)}, rk^{(b,f)})$ for all file owners. During the auditing process, different owners of the same file use different random file tags $(\beta^{(a,f)}, \beta^{(b,f)})$ and public keys $(pk^{(a,f)}, pk^{(b,f)})$ to delegate auditing tasks. Then the CSP uses the same authentication tags σ and different re-signature keys $(rk^{(a,f)}, rk^{(b,f)})$ to generate different user-related proofs $(Proof^{(a)}, Proof^{(b)})$ for the user $\mathcal{U}_\mathcal{A}$ and user $\mathcal{U}_\mathcal{B}$. Therefore, it can prevent the leakage of OP from auditing logs.

Resistance to DFA. The most straightforward way to resist DFA is to check the consistency of the file tag and ciphertext uploaded by the initial uploader. To protect the OP of users, different users generate random and different file tags for an identical file, which hinders the CSP from directly verifying the consistency of the ciphertext and file tag uploaded by the initial uploader in our system. Therefore, the resistance of DFA can only be achieved indirectly through other processes.

- The initial uploader $\mathcal{U}_\mathcal{A}$ uploads the ciphertext $C^{(a)}$, authentication tags σ and file tag $\beta^{(a,f)}$ to the CSP. Then the CSP checks the association between σ and $C^{(a)}$ instead of the association between $\beta^{(a,f)}$ and $C^{(a)}$. Essentially, the CSP verifies that the signed content in σ is $C^{(a)}$, which will be described in the initial uploading process in Section 4.4.1. If the verification is invalid, the CSP rejects the data outsourcing request.
- When a subsequent uploader $\mathcal{U}_\mathcal{B}$ uploads the file tag $\beta^{(b,f)}$ to the CSP to detect duplication, the CSP can detect that $\beta^{(a,f)}$ and $\beta^{(b,f)}$ correlate to the same file. Then the CSP uses the authentication tags to check the file ownership of the user $\mathcal{U}_\mathcal{B}$. Essentially, the CSP verifies that the signed content in σ is $C^{(b)}$, which will be described in the subsequent uploading process in Section 4.4.2. If the verification is invalid, the CSP will treat $C^{(b)}$ as a new file.

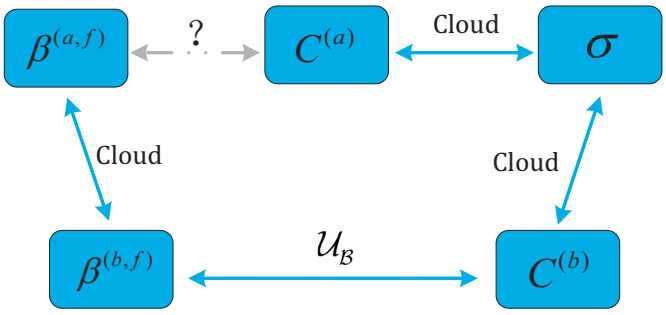


Fig. 2: Principle of resisting the DFA.

Fig. 2 shows how our design can resist DFA. The aforementioned verifications can ensure the association between $C^{(a)}$ and $C^{(b)}$ through σ. The association between $\beta^{(a,f)}$ and $\beta^{(b,f)}$ is checked through duplication detection which is a basic function of the cloud. The association between $\beta^{(b,f)}$ and $C^{(b)}$ can be ensured by the subsequent uploader $\mathcal{U}_\mathcal{B}$ since the user $\mathcal{U}_\mathcal{B}$ is the victim of DFA. Thus the association between the $\beta^{(a,f)}$ and $C^{(a)}$ can be ensured and subsequent uploaders will not suffer from DFA. We will provide a further theoretical proof in Section 5.2.4.

Reliability of auditing results. To deduplicate the authentication tags, previous methods generate authentication tags using the MLE key. Since the MLE key can be easily deduced by the CSP for low-entropy data, the CSP can deceive the TPA using forged ciphertext and its corresponding authentication tags even if the data is damaged or lost. We apply multiple key servers to ensure the reliability of auditing results. The user generates the signing key of authentication tags through the blind signature protocol with multiple key servers, which prevents the cloud deducing the signing key. Then for low-entropy data, the CSP cannot deceive the TPA since it cannot forge authentication tags without the signing key.

Recoverability of encryption keys. To ensure the recoverability of encryption keys, our scheme requires key servers to publish the blind signatures of data hash values on the blockchain. Due to the immutability of the blockchain, the recoverability of encryption keys is guaranteed. This design can also ensure that the public blind signatures of data hash values on the blockchain cannot leak the encryption keys, since the user and key servers perform the BLS blind signature protocol [44]. Besides, the introduction of blockchain can also significantly reduce the local key storage cost for users.

4 Proposed Scheme

In this section, we first introduce the preliminaries, and then present our scheme, which includes the system setup, key generation, data uploading, integrity auditing and data downloading.

4.1 Preliminaries

4.1.1 Bilinear Pairing of Composite Order

We use the bilinear pairing [49] to construct the methods of duplication detection and auditing, and design the secure deduplication and auditing scheme based on the discrete logarithm (DL) problem [50].

Given a security parameter κ, the composite bilinear parameter generator $\mathcal{G}en(\kappa)$ outputs a six tuple $(p, q, N = pq, \mathbb{G}_1, \mathbb{G}_2, \boldsymbol{e})$, where p and q are κ-bit primes and $\boldsymbol{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear pairing, which follows the properties:

- *Bilinear*: $\boldsymbol{e}(x^a, y^b) = \boldsymbol{e}(x, y)^{ab}$ for all $x, y \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_N$.
- *Non-degeneracy*: If g is a generator of $\mathbb{G}_1$, the $\boldsymbol{e}(g, g)$ is a generator of $\mathbb{G}_2$ with the order N.
- *Computability*: For all $x, y \in \mathbb{G}_1$, there exists an efficient algorithm to compute $\boldsymbol{e}(x, y) \in \mathbb{G}_2$.

The complexity assumption of the DL problem is given as follows.

Definition 1. *[50] Given $x, g \in \mathbb{G}$ as input, the complexity of the DL problem is equal to computing $a \in \mathbb{Z}_N$ such that $g^a = x$.*

4.1.2 Blockchain

A blockchain [51, 52] network is generally maintained by some nodes using a common consensus mechanism. The blockchain consists of linear set of data units, where each data unit is called a data block. The data in a block contains a timestamp, multiple transactions data, and so on. The timestamp records the time when the block was generated. The transaction data generally contain the events of transferring tokens from one node to another. The data recorded in the blocks are chained chronologically using a cryptographic hash function, which ensures the immutability and verifiability of blockchain data.

4.1.3 Authentication Tags

To allow cloud users to check the integrity of their outsourced data, integrity auditing is required whereby the CSP proves to its users that their outsourced data are correctly stored. Such integrity auditing techniques implement an algorithm that appends a tag (authentication tag) to every data segment (block) before storing them on the cloud [13]. These authentication tags are computed under a secret signing key generated by the user and are further used for the CSP to generate integrity proofs.

4.1.4 Multiple Servers-aided Message-locked Encryption

Multiple Servers-aided Message-locked Encryption includes the following steps.

- The Multiple Servers-aided MLE Key Generation: A user first computes the hash values of the outsourced data blocks and then generates encryption keys by running the blind signature protocol with multiple key servers. The signatures of data hash values are used as the encryption keys.
- Encryption: The user encrypts the outsourced data blocks using the encryption keys and an existing symmetric encryption algorithm (e.g., ASE-256).
- Decryption: The user decrypts the ciphertext using the encryption keys and the symmetric decryption algorithm.

4.2 System Setup

In this phase, the key servers $\{\mathcal{KS}_i\}_{(1\leq i\leq I)}$ and CSP generate some public parameters and some secret parameters kept by themselves.

4.2.1 Setup of the Cloud

The CSP performs the setup phase as follows.

- Randomly choose a security parameter κ and run the composite order bilinear parameter generator $\mathcal{G}en(\kappa)$ to output a tuple $(p, q, N = pq, \mathbb{G}_1, \mathbb{G}_2, \boldsymbol{e})$.
- Randomly choose two generators $g_1, g_2 \in \mathbb{G}_1$ and an element $\mu \in \mathbb{G}_1$.
- Choose two pseudo-random functions $\boldsymbol{\pi_1} : \{1, ..., n\} \times \mathbb{Z}_N^* \rightarrow \{1, ..., n\}$ and $\boldsymbol{\pi_2} : \{1, ..., n\} \times \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$.
- Choose a hash function $\boldsymbol{H_1} : \mathbb{G}_1 \rightarrow \{0, 1\}^k$; a hash function $\boldsymbol{H_2} : \{0, 1\}^* \rightarrow \mathbb{G}_1$; a hash function $\boldsymbol{H_3} : \mathbb{G}_1 \rightarrow \mathbb{Z}_N$.
- Compute parameter $\nu = {g_2}^q$.

Finally, the CSP publishes the public parameters $\{\nu, g_1, \mu, N, \boldsymbol{e}, \boldsymbol{\pi_1}, \boldsymbol{\pi_2}, \boldsymbol{H_1}, \boldsymbol{H_2}, \boldsymbol{H_3}\}$ to all entities of the system and holds the secret parameters $\{p, q\}$.

4.2.2 Setup of Key Servers

The key server $\mathcal{KS}_i$ performs the setup phase as follows.

- $\mathcal{KS}_i$ randomly chooses an element $a_{i,0} \in \mathbb{Z}_N^*$ and a polynomial $f_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t-1}x^{t-1}$ over $\mathbb{Z}_N$ with degree at most $t-1$ such that $f_i(0) = a_{i,0}$.
- $\mathcal{KS}_i$ computes $\{g_1^{a_{i,\epsilon}}\}_{(0\leq\epsilon\leq t-1)}$ and $\{f_i(j)\}_{(1\leq j\leq I, j\neq i)}$. Then $\mathcal{KS}_i$ publishes $\{g_1^{a_{i,\epsilon}}\}_{(0\leq\epsilon\leq t-1)}$ and sends $f_i(j)$ to $\mathcal{KS}_j$ via secret channel.
- $\mathcal{KS}_i$ obtains $\{f_j(i)\}_{(1\leq j\leq I, j\neq i)}$ from all the other key servers $\{\mathcal{KS}_j\}_{(1\leq j\leq I, j\neq i)}$. For $j = 1, 2, ..., I, j \neq i$, if Eq. (1) holds, $\mathcal{KS}_i$ accepts $f_j(i)$.

$$g_1^{f_j(i)} \stackrel{?}{=} \prod_{\epsilon=0}^{t-1} g_1^{a_{j,\epsilon} \cdot i^\epsilon} \tag{1}$$

- $\mathcal{KS}_i$ computes its secret share $s_i = \sum_{j=1}^{I} f_j(i)$ and its public share $ps_i = g_1^{s_i}$.

Finally, the secret key $s = \sum_{i=1}^{I} a_{i,0}$ is shared among all key servers $\{\mathcal{KS}_i\}_{(1\leq i\leq I)}$.

4.3 Key Generation

When a user $\mathcal{U}$ wants to outsource a file $F = m_1||m_2||\cdots||m_n$ to the CSP, the user first interacts with the key servers to generate the encryption keys as follows.

- The user $\mathcal{U}$ chooses a seed $\theta \in \mathbb{Z}_N^*$ and generates random numbers $\{r_\epsilon = \boldsymbol{\pi_2}(\epsilon, \theta)\}_{(1\leq\epsilon\leq n+1)}$.
- The user $\mathcal{U}$ first computes the initial data hash values $h_\epsilon = \boldsymbol{H_2}(m_\epsilon), (1 \leq \epsilon \leq n)$ and $h_{n+1} = \boldsymbol{H_2}(F)$, and then computes the blind hash values $x_\epsilon = h_\epsilon^{r_\epsilon}, (1 \leq \epsilon \leq n+1)$. Finally, the user sends the blind hash values $\{x_\epsilon\}_{(1\leq\epsilon\leq n+1)}$ to all the key servers.
- After receiving the blind hash values $\{x_\epsilon\}_{(1\leq\epsilon\leq n+1)}$, every key server $\mathcal{KS}_i$ computes the blind signatures

$y_\epsilon^{(i)} = x_\epsilon^{s_i} (1 \le \epsilon \le n+1)$. Then it sends $\{y_\epsilon^{(i)}\}_{(1\le\epsilon\le n+1)}$ to the user $\mathcal{U}$ and publishes them on the blockchain.

- Upon obtaining $\{\{y_\epsilon^{(i)}\}_{(1\le\epsilon\le n+1)}\}_{(1\le i\le I)}$, the user $\mathcal{U}$ verifies whether Eq. (2) holds for $i = 1, 2, ..., I$, $\epsilon = 1, 2, ..., n+1$.

$$\boldsymbol{e}(x_\epsilon, ps_i) \stackrel{?}{=} \boldsymbol{e}(y_\epsilon^{(i)}, g_1) \tag{2}$$

After obtaining t valid signatures. $\mathcal{U}$ computes

$$w_i = \prod_{1\le k\le t, k\ne i} \frac{k}{k-i}, (1 \le i \le t) \tag{3}$$

$$z_\epsilon = \prod_{i=1}^{t} (y_\epsilon^{(i)})^{r_\epsilon^{-1}\cdot w_i}, (1 \le \epsilon \le n+1) \tag{4}$$

Finally, the user $\mathcal{U}$ can compute the encryption keys $\{sk_\epsilon = \boldsymbol{H_1}(z_\epsilon)\}_{(1\le\epsilon\le n)}$, and the signing key $z = \boldsymbol{H_3}(z_{n+1})$ of authentication tags.

The blockchain is used to record the blind signatures of data hash values in the key generation process. The blind signatures cannot leak any useful information about encryption keys, which will be proved in Section 5.2.1. The recoverability of encryption keys can also be ensured due the immutability of blockchain. The user only need to store the seed θ and can recover the encryption keys from the blind signatures on-chain. This can significantly reduce the key storage cost for users and achieves considerable convenience and key security for users.

4.4 Data Uploading

After generating the encryption keys $\{sk_i\}_{(1\le i\le n)}$ and the signing key z of authentication tags, the user $\mathcal{U}$ interacts with the CSP to check whether the file already exists in the cloud as follows.

- The user $\mathcal{U}$ chooses two random values $s^{(u)}$ and $x^{(u)}$ from $\mathbb{Z}_N$ as the re-signature key $rk^{(u,f)} = \{s^{(u)}, x^{(u)}\}$. The user also chooses a random $r^{(u)}$ from $\mathbb{Z}_N$ and computes a random file tag $\beta^{(u,f)} = g_1^z \cdot \nu^{r^{(u)}}$ and the public key $pk^{(u,f)} = g_1^{z\cdot s^{(u)}}$.
- The user $\mathcal{U}$ sends $\beta^{(u,f)}$ and the length of outsourced file Len to the CSP.
- For each Len-length file F^* stored in the cloud, the CSP checks whether the Eq. (5) holds or not.

$$\boldsymbol{e}(\beta^{(u,f)}, g_1)^p \stackrel{?}{=} \boldsymbol{e}(\beta^{(u^*,f^*)}, g_1)^p \tag{5}$$

If it holds, the file F has already existed in the cloud and the CSP replies Dup to the user. Otherwise, the file does not exist in the cloud and the CSP replies $NoDup$ to the user.

The reply with $NoDup$ means that the user is an initial uploader, while the reply with Dup means that the user is a subsequent uploader. Note that the efficiency of detecting duplication in the cloud is not the focus of this study. Many existing works (e.g., B+ tree and method in [53]) have focused on this issue and we directly use an existing method in our scheme. Besides, for convenience, we use $\mathcal{U}_\mathcal{A}$ to present an initial uploader and $\mathcal{U}_\mathcal{B}$ to present a subsequent uploader in the remainder of this study.

4.4.1 Initial Uploading

When receiving the reply with $NoDup$ from the CSP, the initial uploader $\mathcal{U}_\mathcal{A}$ interacts with the CSP to execute the initial uploading as follows.

- The initial uploader encrypts the n data blocks $F = m_1||m_2||\cdots||m_n$ using an existing symmetric encryption algorithm (e.g., the AES-256) with the secret keys $\{sk_i\}_{(1\le i\le n)}$ and generates the ciphertext $C = c_1||c_2||\cdots||c_n$. For each c_i, the user first divides it into d sectors with equal length and then signs it using z to generate the authentication tag as follows.

$$\sigma_i = [\boldsymbol{H_2}(c_i||i) \cdot \prod_{j=1}^{d} \mu^{c_{i,j}}]^z, \qquad (1 \le i \le n) \tag{6}$$

Then the user $\mathcal{U}_\mathcal{A}$ sends the ciphertext C, authentication tags $\sigma = \{\sigma_i\}_{(1\le i\le n)}$, re-signature key $rk^{(a,f)}$, and public key $pk^{(a,f)}$ to the CSP.

- The CSP chooses a seed α from $\mathbb{Z}_N^*$ and generates the random numbers $a_i = \boldsymbol{\pi_2}(i, \alpha)$ $(1 \le i \le n)$. The CSP checks the association of the ciphertext $C = c_1||c_2||\cdots||c_n$ and the authentication tags $\{\sigma_i\}_{(1\le i\le n)}$ using the Eq. (7)

$$\begin{aligned} &\boldsymbol{e}(\prod_{i=1}^{n} \boldsymbol{H_2}(c_i||i)^{a_i} \cdot \mu^{\sum_{j=1}^{d}\sum_{i=1}^{n} a_i\cdot c_{i,j}}, pk^{(a,f)}) \\ \stackrel{?}{=}&\boldsymbol{e}(\prod_{i=1}^{n} \sigma_i^{a_i}, g_1^{s^{(a)}}) \end{aligned} \tag{7}$$

If the above equation holds, the CSP stores the file tag $\beta^{(a,f)}$, the re-signature key $rk^{(a,f)}$, the ciphertext C and the authentication tags $\{\sigma_i\}_{(1\le i\le n)}$. Otherwise, the CSP rejects the storage requirement, since the uploaded ciphertext does not match the authentication tags.

Note that our scheme can also achieve the block-level deduplication by adding two extra interactions. Specifically, before uploading the ciphertext, the user $\mathcal{U}_\mathcal{A}$ sends the hash values of the n ciphertext blocks $\{\boldsymbol{H_2}(c_i)\}_{(1\le i\le n)}$ to the CSP to further detect duplicate blocks. Then the CSP checks whether it has stored the same blocks and creates a set B to indicate which blocks are not stored and should be uploaded according to the $\{\boldsymbol{H_2}(c_i)\}_{(1\le i\le n)}$. The CSP sends B to the user. The user $\mathcal{U}_\mathcal{A}$ uploads the ciphertext blocks $\{c_i\}(i \in B)$ to the CSP, rather than the whole ciphertext C.

4.4.2 Subsequent Uploading

When receiving the reply with Dup from the CSP, the subsequent uploader $\mathcal{U}_\mathcal{B}$ executes the PoW protocol with the CSP.

- When detecting the user is a subsequent uploader, the CSP chooses two seeds r_1, r_2 from $\mathbb{Z}_N^*$ and a random number $l_1 \in [1, n]$. Then the CSP sends the challenging data $Chal^{(x)} = \{r_1, r_2, l_1\}$ to the user $\mathcal{U}_\mathcal{B}$.
- The user $\mathcal{U}_\mathcal{B}$ computes $a_i = \boldsymbol{\pi_1}(i, r_1)$ and $b_i = \boldsymbol{\pi_2}(i, r_2)$ for each $1 \le i \le l_1$. Then the user computes $c_{a_i} = Enc(sk_{a_i}, m_{a_i})$, $p_j = \sum_{i=1}^{l_1} b_i \cdot c_{a_i,j}$ $(1 \le j \le d)$, $\rho^{(x)} = \sum_{j=1}^{d} p_j$ and $T^{(x)} = \prod_{i=1}^{l_1} \boldsymbol{H_2}(c_{a_i}||a_i)^{b_i}$.
- The user $\mathcal{U}_\mathcal{B}$ sends the $Proof^{(x)} = \{T^{(x)}, \rho^{(x)}\}$, re-signature key $rk^{(b,f)}$ and public key $pk^{(b,f)}$ to the CSP.

- The CSP computes $a_i = \boldsymbol{\pi_1}(i, r_1)$ and $b_i = \boldsymbol{\pi_2}(i, r_2)$ for each $1 \le i \le l_1$. Then it computes $\delta^{(x)} = \prod_{i=1}^{l_1} \sigma_{a_i}^{b_i}$.
- The CSP verifies whether the Eq. (8) holds.

$$\boldsymbol{e}(T^{(x)} \cdot \mu^{\rho^{(x)}}, pk^{(b,f)}) \stackrel{?}{=} \boldsymbol{e}(\delta^{(x)}, g_1^{s^{(b)}}) \tag{8}$$

If it holds, the user $\mathcal{U}_{\mathcal{B}}$ is an actual owner of F and the CSP stores the re-signature key $rk^{(b,f)}$. Otherwise, the user $\mathcal{U}_{\mathcal{B}}$ is not an actual owner.

4.5 Integrity Auditing

The user $\mathcal{U}$ sends the file tag $\beta^{(u,f)}$ and public key $pk^{(u,f)}$ to the TPA to delegate the auditing task. Then the TPA interacts with the CSP through the blockchain as follows.

- The TPA chooses two seeds r_3, r_4 from $\mathbb{Z}_N^*$ and a random number $l_2 \in [1, n]$. Then it publishes the challenging data $Chal^{(y)} = \{r_3, r_4, l_2\}, ID^{(u)}, \beta^{(u,f)}$ on the blockchain, where $ID^{(u)}$ is the ID of the user.
- After obtaining the $Chal^{(y)}, ID^{(u)}, \beta^{(u,f)}$ from the blockchain, the CSP first retrieves all the files owned by the user $\mathcal{U}$. For each file of the user $\mathcal{U}$, the CSP checks Eq. (5) to find the audited file F. The CSP computes $a_i = \boldsymbol{\pi_1}(i, r_3)$ and $b_i = \boldsymbol{\pi_2}(i, r_4)$ for each $1 \le i \le l_2$. Then the CSP computes

$$\begin{cases} \delta^{(y)} = (\prod_{i=1}^{l_2} \sigma_{a_i}^{b_i})^{s^{(u)} \cdot x^{(u)}} \\ p_{a_i} = \sum_{j=1}^{d} x^{(u)} \cdot c_{a_i,j}, (1 \le i \le l_2) \\ T^{(y)} = (\prod_{i=1}^{l_2} \boldsymbol{H_2}(c_{a_i} || a_i)^{b_i})^{x^{(u)}} \end{cases} \tag{9}$$

and publishes the proof information $Proof^{(y)} = \{T^{(y)}, \{p_{a_i}\}_{(1 \le i \le l_2)}, \delta^{(y)}\}, \beta^{(u,f)}$ on the blockchain
- After obtaining the $Proof^{(y)}, \beta^{(u,f)}$ from the blockchain, the TPA computes $b_i = \boldsymbol{\pi_2}(i, r_4), (1 \le i \le l_2)$, $\rho^{(y)} = \sum_{i=1}^{l_2} b_i \cdot p_{a_i}$ and verifies the Eq. (10).

$$\boldsymbol{e}(T^{(y)} \cdot \mu^{\rho^{(y)}}, pk^{(u,f)}) \stackrel{?}{=} \boldsymbol{e}(\delta^{(y)}, g_1) \tag{10}$$

If the above equation holds, the TPA publishes ($\beta^{(u,f)}$, integral) on the blockchain. Otherwise, it publishes ($\beta^{(u,f)}$, non-integral)
- The user $\mathcal{U}$ can get the auditing result from the blockchain.

Although the random file tag is used to protect OP in the challenging data, the CSP can still detect which file is audited using the method of duplication detection. Using the Eq. (5), the CSP can compare the random file tag in the challenging data with that stored on the cloud, and then detect which file is audited. For the proof information, the TPA can know the association between the integrity proofs and challenges and thus obtains the integrity proof accurately. This is because the random file tags in the proof information and challenging information are the same for each auditing. Besides, even though our auditing scheme publishes all the auditing logs on-chain, it can still protect the OP of the user $\mathcal{U}$. We will provide the theoretical proof in Section 5.2.3.

4.6 Data Downloading

When a user $\mathcal{U}$ would like to recover a file, the user should first verify the auditing result. If the requested file is integral, the user can retrieve the ciphertext file from the cloud and recover the encryption keys from the blockchain. Then the user can recover the plaintext from the ciphertext.

The user $\mathcal{U}$ first downloads the latest auditing logs (i.e., the challenging information, integrity proofs and auditing result) of the requested file $\beta^{(u,f)}$. If the on-chain auditing result published by the TPA is non-integral, the user $\mathcal{U}$ detects the corruption of the requested file. Otherwise, the user $\mathcal{U}$ computes $b_i = \boldsymbol{\pi_2}(i, r_4), (1 \le i \le l_2)$ and verifies the on-chain integrity proof $(T^{(y)}, \{p_{a_i}\}_{(1 \le i \le l_2)}, \delta^{(y)})$ published by the CSP using Eq. (10). If the verification is valid, the user $\mathcal{U}$ can recover the plaintext as follows.

- The user $\mathcal{U}$ sends a file downloading request with the user ID $ID^{(u)}$ and the file tag $\beta^{(u,f)}$ to the CSP.
- The CSP first retrieves all the files of the user with ID $ID^{(u)}$, and then checks Eq. (5) for each file to find the requested file F. If no file satisfies Eq. (5), the CSP rejects the request. Otherwise, the CSP sends the requested ciphertext $C = c_1 || c_2 || \cdots || c_n$ to the user.
- The user downloads the blind signatures $\{\{y_\epsilon^{(i)}\}_{(1 \le \epsilon \le n)}\}_{(1 \le i \le I)}$ from the blockchain.
- The user first generates n random numbers $\{r_\epsilon\}_{(1 \le \epsilon \le n)}$ using the secret seed θ, and then computes $\{z_\epsilon\}_{(1 \le \epsilon \le n)}$ using Eqs. (3) and (4). Finally, the user $\mathcal{U}$ computes the encryption keys $\{sk_\epsilon = \boldsymbol{H_1}(z_\epsilon)\}_{(1 \le \epsilon \le n)}$.
- The user decrypts the ciphertext C using the encryption keys $\{sk_\epsilon\}_{(1 \le \epsilon \le n)}$. Finally, the plaintext file $F = m_1 || m_2 || \cdots || m_n$ can be obtained.

For the convenience of reading and understanding, we list the private parameters of each entity in Table 2, and other parameters are publicly available to all entities.

TABLE 2: Private parameters of each entity.

Entity	Parameters
User ($\mathcal{U}$)	$\theta, z, r^{(u)}, rk^{(u,f)} = \{x^{(u)}, s^{(u)}\}$
CSP	$p, q, rk^{(u,f)} = \{x^{(u)}, s^{(u)}\}$
Key server ($\mathcal{KS}_i$)	s_i
TPA	-
Blockchain	-

5 Scheme Analysis

In this section, we prove the correctness of our scheme and analyze its security.

5.1 Correctness

5.1.1 Correctness of Deduplication

We prove that the CSP can detect duplicate files using the random file tags. For two users $\mathcal{U}$ and $\mathcal{U}^*$ owning an identical file F, even though they generate different random file tags, the CSP can still detect duplication using Eq. (5).

We use z and z^* to present the signatures of file hash values for the files owned by the user $\mathcal{U}$ and the user $\mathcal{U}^*$, respectively. Suppose that the user $\mathcal{U}^*$ has uploaded the ciphertext C and the file tag $\beta^{(u^*,f^*)} = g_1^{z^*} \cdot \nu^{r^{(u^*)}}$ ($\nu = g_2^q$ and q is a large prime) to the CSP. When the user $\mathcal{U}$ uploads

his/her file tag $\beta^{(u,f)} = g_1^z \cdot \nu^{r^{(u)}}$ to the CSP for checking duplication, the CSP uses its secret large prime p to compute the two sides of Eq. (5) as follows.

$$\begin{aligned}\boldsymbol{e}(\beta^{(u,f)}, g_1)^p &= \boldsymbol{e}(g_1^z \nu^{r^{(u)}}, g_1)^p \\ &= \boldsymbol{e}(g_1^z, g_1)^p \cdot \boldsymbol{e}(\nu^{r^{(u)}}, g_1)^p \\ &= \boldsymbol{e}(g_1^z, g_1)^p \cdot \boldsymbol{e}(g_2^{q \cdot r^{(u)}}, g_1)^p \\ &= \boldsymbol{e}(g_1^z, g_1)^p \cdot \boldsymbol{e}(g_2, g_1)^{p \cdot q \cdot r^{(u)}} \\ &= \boldsymbol{e}(g_1^z, g_1)^p\end{aligned} \tag{11}$$

$$\begin{aligned}\boldsymbol{e}(\beta^{(u^*,f^*)}, g_1)^p &= \boldsymbol{e}(g_1^{z^*} \nu^{r^{(u^*)}}, g_1)^p \\ &= \boldsymbol{e}(g_1^{z^*}, g_1)^p \cdot \boldsymbol{e}(\nu^{r^{(u^*)}}, g_1)^p \\ &= \boldsymbol{e}(g_1^{z^*}, g_1)^p \cdot \boldsymbol{e}(g_2^{q \cdot r^{(u^*)}}, g_1)^p \\ &= \boldsymbol{e}(g_1^{z^*}, g_1)^p \cdot \boldsymbol{e}(g_2, g_1)^{p \cdot q \cdot r^{(u^*)}} \\ &= \boldsymbol{e}(g_1^{z^*}, g_1)^p\end{aligned} \tag{12}$$

It has been proved that $\boldsymbol{e}(\beta^{(u,f)}, g_1)^p = \boldsymbol{e}(\beta^{(u^*,f^*)}, g_1)^p$ only when $z = z^*$ [28]. Besides, z is equal to z^* only when the users $\mathcal{U}$ and $\mathcal{U}^*$ have the same file. Thus, the CSP can detect duplicate files through the file tags.

5.1.2 Correctness of Auditing

Assume that the user $\mathcal{U}$ delegates the TPA to audit a ciphertext file C by sending the file tag $\beta^{(u,f)}$ and public key $pk^{(u,f)}$ to the TPA. The TPA verifies the integrity proof data $Proof^{(y)} = \{T^{(y)}, \{p_{a_i}\}_{(1 \le i \le l_2)}, \delta^{(y)}\}$ by verifying that Eq. (10) is correct, where $\rho^{(y)} = \sum_{i=1}^{l_2} b_i \cdot p_{a_i}$. The TPA computes the two sides of Eq. (10) as follows

$$\begin{aligned}&\boldsymbol{e}(T^{(y)} \cdot \mu^{\rho^{(y)}}, pk^{(a,f)}) \\ =&\boldsymbol{e}((\prod_{i=1}^{l_2} \boldsymbol{H_2}(c_{a_i}||a_i)^{b_i})^{x^{(u)}} \cdot \mu^{\sum_{j=1}^{d} x^{(u)} \cdot \sum_{i=1}^{l_2} b_i \cdot c_{a_i,j}}, g_1^{z \cdot s^{(u)}}) \\ =&\boldsymbol{e}(\prod_{i=1}^{l_2} \boldsymbol{H_2}(c_{a_i}||a_i)^{b_i} \cdot \mu^{\sum_{j=1}^{d} \cdot \sum_{i=1}^{l_2} b_i \cdot c_{a_i,j}}, g_1^{z \cdot s^{(u)} \cdot x^{(u)}})\end{aligned} \tag{13}$$

$$\begin{aligned}&\boldsymbol{e}(\delta^{(y)}, g_1) \\ =&\boldsymbol{e}(\prod_{i=1}^{l_2} ((\boldsymbol{H_2}(c_{a_i}||a_i) \cdot \prod_{j=1}^{d} \mu^{c_{a_i,j}})^z)^{b_i \cdot s^{(u)} \cdot x^{(u)}}, g_1) \\ =&\boldsymbol{e}(\prod_{i=1}^{l_2} ((\boldsymbol{H_2}(c_{a_i}||a_i) \cdot \prod_{j=1}^{d} \mu^{c_{a_i,j}})^z)^{b_i}, g_1^{s^{(u)} \cdot x^{(u)}}) \\ =&\boldsymbol{e}(\prod_{i=1}^{l_2} \boldsymbol{H_2}(c_{a_i}||a_i)^{b_i} \cdot \mu^{\sum_{j=1}^{d} \cdot \sum_{i=1}^{l_2} b_i \cdot c_{a_i,j}}, g_1^{z \cdot s^{(u)} \cdot x^{(u)}})\end{aligned} \tag{14}$$

It can be seen that $\boldsymbol{e}(T^{(y)} \cdot \mu^{\rho^{(y)}}, pk^{(a,f)}) = \boldsymbol{e}(\delta^{(y)}, g_1)$, which verifies the correctness of the auditing method. Besides, we will further prove that the CSP can not deceive the TPA using forged ciphertext and its corresponding authentication tags in Section 5.2.2.

5.2 Security Analysis

We analyze the security that is expected to achieve in our scheme from the aspects of data confidentiality, reliability of auditing results, OP protection and resistance to DFA.

5.2.1 Data Confidentiality

Our scheme should protect the data confidentiality, which indicates that only the file owners can obtain the plaintext content $F = \{m_i\}_{(1 \le i \le n)}$ and other unauthorized entities cannot get the content. Since the well-known AES-256 is used to encrypt the F and it is semantically secure, the adversary can only recover the plaintext by restoring the encryption keys $\{sk_i\}_{(1 \le i \le n)}$. On one hand, the encryption keys $\{sk_i\}_{(1 \le i \le n)}$ may be recovered using on-chain blind signatures. On the other hand, to achieve deduplication, the encryption keys are deterministic and related to the hash values of data blocks. This strategy can be used by the adversary to launch brute-force attacks. We analyze the data confidentiality from the above two aspects.

Since the CSP knows more knowledge than other entities, we only need to prove that the data is secure when the adversary is the CSP. First, we state that

Theorem 1. *The CSP cannot deduce the encryption keys using the on-chain blind signatures generated in the key generation phase.*

Proof. Let $\mathcal{C}$ be a challenger, $\mathcal{A}$ be an adversary (the CSP) that can attack our scheme with advantage $\epsilon(\kappa)$.

- **Init:** For the given parameters $\mathbb{G}_1, \mathbb{Z}_N^*$ and $r \in \mathbb{Z}_N^*$, the challenger $\mathcal{C}$ sends the public parameters $\mathbb{G}_1, \mathbb{Z}_N^*$ to $\mathcal{A}$.
- **Challenge:** The adversary $\mathcal{A}$ selects two messages $m_0, m_1 \in \mathbb{G}_1$, and submits them to $\mathcal{C}$. Then $\mathcal{C}$ flips a fair binary coin α^*, and returns an ciphertext of $m_{\alpha^*} \in \{m_0, m_1\}$. Then the ciphertext $C_{\alpha^*} = m_{\alpha^*}^r$ is outputted to $\mathcal{A}$.
- **Guess:** The $\mathcal{A}$ outputs a guess $\alpha^{'}$ of α^*. If $\alpha^{'}$ is equal to the α^*, the challenger $\mathcal{C}$ outputs 1. Otherwise, the $\mathcal{C}$ outputs 0.

Then we have that

$$\Pr[\mathcal{A}(\alpha^* = \alpha^{'})] = \frac{1}{2} + \epsilon(\kappa) \tag{15}$$

When r is a random element from $\mathbb{Z}_N^*$, the $m_{\alpha^*}^r$ is also a random element in $\mathbb{G}_1$ from the view of $\mathcal{A}$. This means that the adversary $\mathcal{A}$ obtains no information related to α^*. Hence,

$$\Pr[\mathcal{A}(\alpha^* = \alpha^{'})] = \frac{1}{2} \tag{16}$$

Thus, the adversary cannot obtain any information about the encryption keys using these on-chain blind signatures generated in the key generation phase. □

Besides, we analyze the resistance of brute-force attacks. Suppose that the adversary $\mathcal{A}$ knows the ciphertext $C = \{c_i\}_{(1 \le i \le n)}$ and the file tag, namely $\beta^{(u,f)} = g_1^z \cdot \nu^{r^{(u)}} = g_1^z \cdot g_2^{q \cdot r^{(u)}}$. It may perform brute-force attacks using the file tag $\beta^{(u,f)}$ and the method of duplication checking in Eq. (5), or using the ciphertext $\{c_i\}_{(1 \le i \le n)}$.

In the general case, the CSP can only compromise at most $t - 1$ key servers which is discussed in our threat

model in Section 3.2. Thus the adversary $\mathcal{A}$ cannot compute the parameter $z = \boldsymbol{H_3}(F)^s$, because s is a secret shared among all the key servers and should be recovered from t key servers. Thus the adversary $\mathcal{A}$ can not launch brute-force attacks using the file tag. When launching the brute-force attacks using the ciphertext $\{c_i\}_{(1\leq i\leq n)}$, it should also compute the encryption keys $sk_i = \boldsymbol{H_2}(m_i)^s$. Since the CSP cannot obtain the secret s shared among key servers, it also cannot launch brute-force attacks using the ciphertext. Therefore, the CSP cannot get the plaintext even though it compromises $t-1$ key servers.

5.2.2 Reliability of Auditing Result

During the auditing process, the CSP can use a forged file and its corresponding authentication tags to deceive the auditor if it obtains the secret signing key of authentication tags [43]. Thus, the reliability of auditing results depends on whether the CSP can deduce the signing secret key z.

Theorem 2. *The CSP cannot deduce the signing key z of authentication tags using its background knowledge if the Discrete Logarithm (DL) assumption holds.*

Proof. As discussed in our threat model in Section 3.2, the CSP cannot collude with several key servers that is equal to or more than t. Then it cannot obtain the secret s shared among all the key servers. Thus, the CSP cannot deduce the signing key $z = \boldsymbol{H_3}(F)^s$ through brute-force attacks.

The CSP can obtain the public key $pk^{(u,f)}$, re-signature key $rk^{(u,f)}$ of the file and file tag $\beta^{(u,f)}$, where $pk^{(u,f)} = g_1^{z\cdot s^{(u)}}$, $rk^{(u,f)} = \{s^{(u)}, x^{(u)}\}$ and $\beta^{(u,f)} = g_1^z \cdot \nu^{r^{(u)}}$. For the $\beta^{(u,f)}$, since the CSP has the secret p, it can deduce z through the following way. Using the bilinear pairing $\boldsymbol{e}$ and the method of duplication checking (the left-hand side of Eq. (5)), the CSP can get $\boldsymbol{e}(\beta^{(u,f)}, g_1)^p = \boldsymbol{e}(g_1^z, g_1)^p = \boldsymbol{e}(g_1^p, g_1)^z$. Meanwhile, since the CSP has the parameters g_1, p, it can compute $\boldsymbol{e}(g_1^p, g_1)$. For convenience, we use g to replace the $\boldsymbol{e}(g_1^p, g_1)$ and g^z to replace the $\boldsymbol{e}(g_1^p, g_1)^z$. The difficulty of computing z from g, g^z is equal to solving the DL problem.

Besides, for the $pk^{(u,f)}, rk^{(u,f)}$, the CSP can compute $g_1^{s^{(u)}}$, since it has the $rk^{(u,f)} = \{x^{(u)}, s^{(u)}\}$ and g_1. We also use g and g^z to replace $g_1^{s^{(u)}}$ and $pk^{(u,f)} = g_1^{z\cdot s^{(u)}}$, respectively. The difficulty of computing z from g, g^z is also equal to solving the DL problem.

For the CSP, the difficulty of computing z from its background knowledge is equal to solving the DL problem. Thus it is hard for any probabilistic polynomial-time adversary to obtain z. As a result, the CSP cannot deduce the signing key z of authentication tags.

□

Since the CSP cannot deduce the signing key of authentication tags, it may use a wrong signing key z' to forge the authentication tags $\{\sigma_i'\}_{(1\leq i\leq n)} = \{\sigma_i' = [\boldsymbol{H_2}(c_i'||i)\cdot\prod_{j=1}^{d}\mu^{c_{i,j}'}]^{z'}\}_{(1\leq i\leq n)}$ for the forged ciphertext $C' = c_1'||c_2'||\cdots||c_n'$. Then the CSP can generate the proof data $Proof^{(y)} = \{T'^{(y)}, \{p_{a_i}'\}_{(1\leq i\leq l_2)}, \delta'^{(y)}\}$, where $T'^{(y)} = (\prod_{i=1}^{l_2}\boldsymbol{H_2}(c_{a_i}'||a_i)^{b_i})^{x^{(u)}}$, $\rho'^{(y)} = \sum_{i=1}^{l_2} b_i\cdot p_{a_i}' = \sum_{i=1}^{l_2} b_i\cdot\sum_{j=1}^{d} x^{(u)}\cdot c_{a_i,j}'$, $\delta'^{(y)} = (\prod_{i=1}^{l_2}\sigma'^{b_i}_{a_i})^{s^{(u)}\cdot x^{(u)}}$. When receiving the proof information, the TPA verifies the following equations.

$$\begin{aligned}&\boldsymbol{e}(T'^{(y)}\cdot\mu^{\rho'^{(y)}}, pk^{(a,f)})\\ =&\boldsymbol{e}(\prod_{i=1}^{l_2}\boldsymbol{H_2}(c_{a_i}'||a_i)^{b_i}\cdot\mu^{\sum_{j=1}^{d}\cdot\sum_{i=1}^{l_2}b_i\cdot c_{a_i,j}'}, g_1^{z\cdot s^{(u)}\cdot x^{(u)}})\end{aligned} \tag{17}$$

$$\begin{aligned}&\boldsymbol{e}(\delta'^{(y)}, g_1)\\ =&\boldsymbol{e}(\prod_{i=1}^{l_2}\boldsymbol{H_2}(c_{a_i}'||a_i)^{b_i}\cdot\mu^{\sum_{j=1}^{d}\cdot\sum_{i=1}^{l_2}b_i\cdot c_{a_i,j}'}, g_1^{z'\cdot s^{(u)}\cdot x^{(u)}})\end{aligned} \tag{18}$$

Since z' in Eq. (18) is not equal to the z in Eq. (17), the Eq. (18) is not equal to Eq. (17). Then the CSP cannot deceive the TPA. Thus we can conclude that our scheme can ensure the reliability of auditing results even though the CSP colludes with $t-1$ key servers.

Besides, our scheme can ensure that the decrypted content is integral. The user can check the auditing result before downloading and decrypting the ciphertext. The user can download the latest auditing logs of the requested file. If the on-chain auditing result published by the TPA is non-integral, the user can detect the corruption of the requested file and does not decrypt the ciphertext. Otherwise, the user verifies the on-chain integrity proof published by the CSP using Eq. (10). Only when the verification of integrity proof is valid, then the user downloads and decrypts the ciphertext. Thus, our scheme can ensure that the decrypted content is integral.

5.2.3 OP Protection

According to the threat model discussed in Section 3.2, the adversary $\mathcal{A}$ may snoop on the OP from the auditing logs on-chain. Here, we analyze how our scheme can protect the OP.

Since our scheme uses random file tags, it is obvious that the adversary $\mathcal{A}$ cannot exactly know the files owned by a user. We mainly prove that the adversary $\mathcal{A}$ cannot obtain which users own an identical file.

To snoop on OP, the TPA tends to publish the same challenging data $\{r_3, r_4, l_2\}$ in different auditing processes, which results in the same $\{a_i = \boldsymbol{\pi_1}(i, r_3)\}_{(1\leq i\leq l_2)}$ and $\{b_i = \boldsymbol{\pi_2}(i, r_4)\}_{(1\leq i\leq l_2)}$ generated by the CSP in different auditing processes. Suppose that two users $\mathcal{U}$ and $\mathcal{U}'$ outsource an identical ciphertext C to the CSP and delegate the TPA to audit the same file. We prove that even though the challenged blocks $c_{a_i}(1\leq i\leq l_2)$ and the random coefficients $b_i(1\leq i\leq l_2)$ are the same in two auditing processes, the auditing logs on-chain still cannot leak the OP of $\mathcal{U}$ and $\mathcal{U}'$. Table 3 lists the public information during the auditing processes for the two users $\mathcal{U}$ and $\mathcal{U}'$. We next prove that the public information cannot leak that $\mathcal{U}$ and $\mathcal{U}'$ own an identical file.

For the file tags in the challenging information, it is hard for any probabilistic polynomial-time adversary to distinguish $\beta^{(u,f)}$ or $\beta^{(u',f)}$ from a random element in $\mathbb{G}_1$. since the $r^{(u)}$ in $\beta^{(u,f)}$ is randomly chosen by $\mathcal{U}$ and the $r^{(u')}$ in $\beta^{(u',f)}$ is randomly chosen by $\mathcal{U}'$. Besides, it is also hard for any probabilistic polynomial-time adversary to detect that $\beta^{(u,f)}$ and $\beta^{(u',f)}$ correlate to an identical file

TABLE 3: Auditing logs on-chain.

User ID	File tag	Challenge	Public key	Proof	Result
$\mathcal{U}$	$\beta^{(u,f)} = g_1^z \cdot \nu^{r^{(u)}}$	$\{r_3, r_4, l_2\}$	$pk^{(u,f)} = g_1^{z \cdot s^{(u)}}$	$p_{a_i} = \sum_{j=1}^{d} x^{(u)} \cdot c_{a_i,j} (1 \le i \le l_2)$ $T^{(y,u)} = (\prod_{i=1}^{l_2} \boldsymbol{H_2}(c_{a_i} \| a_i)^{b_i})^{x^{(u)}}$ $\delta^{(y,u)} = (\prod_{i=1}^{l_2} \sigma_{a_i}^{b_i})^{s^{(u)} \cdot x^{(u)}}$	$Res^{(u)}$
$\mathcal{U}^{'}$	$\beta^{(u',f)} = g_1^z \cdot \nu^{r^{(u')}}$	$\{r_3, r_4, l_2\}$	$pk^{(u',f)} = g_1^{z \cdot s^{(u')}}$	$p'_{a_i} = \sum_{j=1}^{d} x^{(u')} \cdot c_{a_i,j} (1 \le i \le l_2)$ $T^{(y,u')} = (\prod_{i=1}^{l_2} \boldsymbol{H_2}(c_{a_i} \| a_i)^{b_i})^{x^{(u')}}$ $\delta^{(y,u')} = (\prod_{i=1}^{l_2} \sigma_{a_i}^{b_i})^{s^{(u')} \cdot x^{(u')}}$	$Res^{(u')}$

without the secret key of the CSP. Thus, the OP of $\mathcal{U}$ and $\mathcal{U}^{'}$ is not leaked from file tags. Besides, it is obvious that the challenging data $\{r_3, r_4, l_2\}$ cannot leak the OP, since they are selected by the TPA and independent on users.

For the public keys of the user $\mathcal{U}$ and the user $\mathcal{U}^{'}$, the $s^{(u)}$ in the public key $pk^{(u,f)} = g_1^{z \cdot s^{(u)}}$ is randomly chosen by $\mathcal{U}$ and the $s^{(u')}$ in $pk^{(u',f)}$ is randomly chosen by $\mathcal{U}^{'}$, $pk^{(u,f)}$ does not equal to $pk^{(u',f)}$. Therefore, the OP of $\mathcal{U}$ and $\mathcal{U}^{'}$ is not leaked from public keys.

For the integrity proofs, the CSP generates the proofs for different users using different re-signature keys. The re-signature key $\{x^{(u)}, s^{(u)}\}$ is randomly chosen by $\mathcal{U}$ and the re-signature key $\{x^{(u')}, s^{(u')}\}$ is randomly chosen by $\mathcal{U}^{'}$. Since the $x^{(u)}$ in p_{a_i} and the $x^{(u')}$ in p'_{a_i} are different, the p_{a_i} does not equal to p'_{a_i} for every $1 \le i \le l_2$. Since the $x^{(u)}$ in $T^{(y,u)}$ and the $x^{(u')}$ in $T^{(y,u')}$ are different, the $T^{(y,u)}$ does not equal to $T^{(y,u')}$. Since $\{x^{(u)}, s^{(u)}\}$ in $\delta^{(y,u)}$ does not equal to $\{x^{(u')}, s^{(u')}\}$ in $\delta^{(y,u')}$, $\delta^{(y,u)}$ does not equal to $\delta^{(y,u')}$. Therefore, the TPA and all users cannot detect that $\mathcal{U}$ and $\mathcal{U}^{'}$ own an identical file using the integrity proofs $\{T^{(y,u)}, \{p_{a_i}\}_{(1 \le i \le l_2)}, \delta^{(y,u)}\}$ and $\{T^{(y,u')}, \{p'_{a_i}\}_{(1 \le i \le l_2)}, \delta^{(y,u')}\}$. Besides, it is obvious that the auditing result cannot leak the OP, since auditing results for all files can be either true or false. From the analysis above, our scheme can protect the OP of users.

5.2.4 Resistance to DFA

We take $\mathcal{U}_\mathcal{A}$ as an initial uploader and $\mathcal{U}_\mathcal{B}$ as a subsequent uploader in the following analysis. Our scheme targets to prevent the subsequent uploaders from obtaining falsified copies, which is shown in Fig. 2 in Section 3.4.2.

- In the initial uploading process, our design requires the CSP to check the association between the ciphertext and authentication tags uploaded by the user $\mathcal{U}_\mathcal{A}$. Thus, to nullify the above primary verification of the CSP, the user $\mathcal{U}_\mathcal{A}$ can only upload falsified ciphertext $C^{(a)}$ and its corresponding falsified authentication tags σ to store poisoned ciphertext on the cloud.
- In the subsequent uploading process, the user $\mathcal{U}_\mathcal{B}$ uploads the file tag $\beta^{(b,f)}$ and the CSP can detect that $\beta^{(a,f)}$ and $\beta^{(b,f)}$ correlate to the same file. Then our design requires the CSP to check the file ownership of the user $\mathcal{U}_\mathcal{B}$ using the authentication tags σ uploaded by the user $\mathcal{U}_\mathcal{A}$. Because the ciphertext $C^{(b)}$ computed by the user $\mathcal{U}_\mathcal{B}$ is different from the falsified content $C^{(a)}$ in σ. The CSP treats $C^{(b)}$ as a new file and the DFA launched by the user $\mathcal{U}_\mathcal{A}$ is invalid.

As the description above, the resistance of DFA relies on the association verification between the ciphertext and authentication tags uploaded by the initial uploader and the PoW. We next further prove the above two operations.

Association Between Ciphertext and Authentication Tags: Assume that the initial uploader $\mathcal{U}_\mathcal{A}$ uploads a forged ciphertext $C' = c'_1 \| c'_2 \| \cdots \| c'_n$ and the real authentication tags $\sigma = \{\sigma_i = [\boldsymbol{H_2}(c_i \| i) \cdot \prod_{j=1}^{d} \mu^{c_{i,j}}]^z\}_{(1 \le i \le n)}$ of the real ciphertext $C = c_1 \| c_2 \| \cdots \| c_n$. Then the CSP checks the association between the ciphertext and authentication tags. The CSP computes

$$\begin{aligned} &\boldsymbol{e}(\prod_{i=1}^{n} \boldsymbol{H_2}(c'_i \| i)^{a_i} \cdot \mu^{\sum_{j=1}^{d} \sum_{i=1}^{n} a_i \cdot c'_{i,j}}, pk^{(a,f)}) \\ =&\boldsymbol{e}(\prod_{i=1}^{n} \boldsymbol{H_2}(c'_i \| i)^{a_i} \cdot \mu^{\sum_{j=1}^{d} \sum_{i=1}^{n} a_i \cdot c'_{i,j}}, g_1^{z \cdot s^{(a)}}) \end{aligned} \tag{19}$$

$$\begin{aligned} &\boldsymbol{e}(\prod_{i=1}^{n} \sigma_i^{a_i}, g_1^{s^{(a)}}) \\ =&\boldsymbol{e}(\prod_{i=1}^{n} ((\boldsymbol{H_2}(c_i \| i) \cdot \prod_{j=1}^{d} \mu^{c_{i,j}})^z)^{a_i}, g_1^{s^{(a)}}) \\ =&\boldsymbol{e}(\prod_{i=1}^{n} \boldsymbol{H_2}(c_i \| i)^{a_i} \cdot \prod_{i=1}^{n} \prod_{j=1}^{d} \mu^{a_i \cdot c_{i,j}}, g_1^{z \cdot s^{(a)}}) \\ =&\boldsymbol{e}(\prod_{i=1}^{n} \boldsymbol{H_2}(c_i \| i)^{a_i} \cdot \mu^{\sum_{j=1}^{d} \sum_{i=1}^{n} a_i \cdot c_{i,j}}, g_1^{z \cdot s^{(a)}}) \end{aligned} \tag{20}$$

Obviously, Eq. (19) is equal to Eq. (20) only when $c'_i = c_i$ $(1 \le i \le n)$. Thus, the CSP can check the association between the ciphertext and the authentication tags of the initial uploader.

PoW: When the subsequent uploader $\mathcal{U}_\mathcal{B}$ interacts with the CSP to upload an already existing file, the CSP should check the ownership of the user $\mathcal{U}_\mathcal{B}$ by verifying whether the Eq. (8) is correct. The CSP computes the two sides of Eq. (8) as follows,

$$\begin{aligned} &\boldsymbol{e}(T^{(x)} \cdot \mu^{\rho^{(x)}}, pk^{(b,f)}) \\ =&\boldsymbol{e}(\prod_{i=1}^{l_1} \boldsymbol{H_2}(c_{a_i} \| a_i)^{b_i} \cdot \mu^{\sum_{j=1}^{d} \sum_{i=1}^{l_1} b_i \cdot c_{a_i,j}}, g_1^{z \cdot s^{(b)}}) \end{aligned} \tag{21}$$

$$
\begin{aligned}
&\boldsymbol{e}(\delta^{(x)}, g_1^{s^{(b)}}) \\
=&\boldsymbol{e}(\prod_{i=1}^{l_1}((\boldsymbol{H_2}(c'_{a_i}||a_i)\cdot\prod_{j=1}^{d}\mu^{c'_{a_i,j}})^z)^{b_i}, g_1^{s^{(b)}}) \\
=&\boldsymbol{e}(\prod_{i=1}^{l_1}\boldsymbol{H_2}(c'_{a_i}||a_i)^{b_i}\cdot\mu^{\sum_{j=1}^{d}\sum_{i=1}^{l_1}b_i\cdot c'_{a_i,j}}, g_1^{z\cdot s^{(b)}})
\end{aligned} \tag{22}
$$

where c'_{a_i} and c_{a_i} are the ciphertexts of challenged blocks signed in authentication tags by $\mathcal{U}_\mathcal{A}$ and encrypted by the subsequent uploader $\mathcal{U}_\mathcal{B}$ respectively. It is obvious that Eq. (21) is equal to Eq. (22) only when $c'_{a_i} = c_{a_i}(1 \le i \le l_1)$. Thus the correctness of the PoW is verified.

Since the association between the ciphertext and authentication tags uploaded by the initial uploader can be verified by the CSP and the association between the ciphertext of the subsequent uploader and the authentication tags can be checked through PoW, we can conclude that the subsequent uploader will not suffer from DFA.

6 Performance Evaluation

In this section, we evaluate the performance of our scheme in terms of functionality, theoretical analysis, and simulation analysis, and compare it with state-of-the-art schemes.

6.1 Functionality and Security Comparisons

Our scheme allows users to outsource their encrypted data to the cloud and audit the data integrity. During the auditing process, the user can stay offline and only the initial user needs to generate and upload the authentication tags while the subsequent users owning the same file do not need. The cloud can perform deduplication to both the authentication tags and encrypted data. Besides, our scheme is designed to protect OP, ensure reliable auditing results for low-entropy data, resist DFA, and reduce key storage cost of users.

Table 4 lists the functionality and security comparisons of different auditing and deduplication schemes. The schemes in [22, 23] focus on plaintext data and thus cannot protect data confidentiality. The other schemes focus on ciphertext data. However, the schemes in [10, 14, 15, 24] do not consider the threat of DFA. The scheme in [10] achieves authentication tags deduplication by introducing an additional fully trusted proxy server and the scheme in [24] does not support authentication tags deduplication. The schemes in [14, 16] require the user to stay online for integrity auditing and authentication tags updates, respectively. In particular, all previous schemes do not consider the leakage of OP, unreliability of auditing results for low-entropy data and key management. Our scheme is the first solution to consider these security issues and completely solve them.

Besides the functionalities list in Table 4, our scheme can also resist brute-force attacks, which also indicates the superiority and availability of our design.

6.2 Theoretical Analysis

Here, we calculate the computation, communication, and storage cost of our scheme in theory. Since the schemes in [22, 23] focus on plaintext data and the scheme in [10] introduces an additional fully trusted proxy server to generate authentication tags, we only compare the related schemes [14–16, 24] in both theoretical and simulation analysis. Let h_e, h_{pr}, h_h, h_m, h_{m^*}, h_a, h_{bm}, h_{ec} denote the computational costs of an exponentiation operation over $\mathbb{G}_1$, a pseudo-random number generation, a hash operation, a multiplication operation over $\mathbb{Z}_N$, a multiplication operation over $\mathbb{G}_1$, an addition operation over $\mathbb{Z}_N$, a bilinear map operation, one-time AES-256 encryption operation, respectively.

6.2.1 Computation Cost

Table 5 shows the computational complexity for different schemes in the data uploading process. For a file with n blocks, all users should generate n encryption keys for the n data blocks and the signing key of authentication tags in the phase of key generation. A user first generates $n+1$ random numbers with a cost of $(n+1)h_{pr}$, computes $n+1$ hash values with a cost of $(n+1)h_h$ and computes $n+1$ blind hash values with a cost of $(n+1)h_e$. After obtaining blind signature values from t key servers, the user needs to verify these blind signature values with a cost of $t(n+1)h_{bm}$ and compute the encryption keys and signing key with a cost of $(n+1)(t(h_m+h_e)+h_h)$. Thus the total computation cost is $(n+1)(h_{pr}+2h_h+(t+1)h_e+t(h_m+h_{bm}))$ in the key generation process. The user also needs to compute a random file tag with a cost of $2h_e+h_{m^*}$. If the outsourced file does not exist on the cloud, the user needs to encrypt all the n blocks and generate n authentication tags for the n encrypted blocks with costs of nh_{ec} and $n(h_h+h_{m^*}+2h_e+dh_a)$, respectively. If the outsourced file exists on the cloud, the user needs to compute ownership proof with a cost of $l_1(2h_{pr}+h_{ec}+dh_a+h_{m^*}+h_h+h_e)+dh_m$.

Table 6 shows the computational complexity for different schemes in the auditing process. The CSP computes the integrity proof with a cost of $l_2(2h_{pr}+2h_e+dh_a+h_m+h_h)+h_m+2h_e$, while the TPA verifies the integrity proof with a cost of $l_2(h_{pr}+h_m+h_a)+h_e+h_m+2h_{bm}$. Our scheme does not require users to participate in the auditing process.

6.2.2 Communication Cost

Table 7 shows the communication overhead for different schemes. All users should interact with key servers to generate the encryption keys and signing key of authentication tags. A user first needs to send $(n+1)$ blind hash values to all I key servers and receives $(n+1)$ blind signature values from every key server. Thus the communication overhead of the user is $(2n+2)I|\mathbb{G}_1|$ in the key generation process. For an initial uploader, the user needs to upload the ciphertext with $|C|$ bits, the authentication tags with $n|\mathbb{G}_1|$ bits, the file tag with $|\mathbb{G}_1|$ bits, the public key with $|\mathbb{G}_1|$ bits and the re-signature key with $2|\mathbb{Z}_N|$ bits. For a subsequent uploader, the user needs to receive the challenging information for PoW with $2|\mathbb{Z}_N|$ bits and upload the file tag with $|\mathbb{G}_1|$ bits, the re-signature key with $2|\mathbb{Z}_N|$ bits, the public key with $|\mathbb{G}_1|$ bits and the ownership proof with $|\mathbb{G}_1|+|\mathbb{Z}_N|$ bits. For the auditing, the user needs to send the file tag with $|\mathbb{G}_1|$ bits and the public key with $|\mathbb{G}_1|$ bits to the TPA for delegating auditing. For data retrieval, the user first needs to download the auditing log with $5|\mathbb{G}_1|+(l_2+2)|\mathbb{Z}_N|$ bits and the blind

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3237221

TABLE 4: Functionality and security comparisons of different auditing and deduplication schemes.

Scheme	Data confidentiality	Authentication tag deduplication	Authentication tag generation	User offline	Key management	Resistance to DFA	Protection of OP	Reliability of auditing results for low-entropy data
Scheme [22]	No	Yes	All Users	Yes	No	No	No	No
Scheme [23]	No	Yes	Initial user	Yes	No	No	No	No
Scheme [10]	Yes	Yes	Other entity	Yes	No	No	No	No
Scheme [14]	Yes	Yes	Initial user	No	No	No	No	No
Scheme [15]	Yes	Yes	Initial user	Yes	No	No	No	No
Scheme [24]	Yes	No	All users	No	No	No	No	No
Scheme [16]	Yes	Yes	Initial user	No	No	Yes	No	No
Our scheme	Yes	Yes	Initial user	Yes	Yes	Yes	Yes	Yes

TABLE 5: Comparisons of user computation cost in data uploading process.

Scheme	All users		Initial uploader	Subsequent uploader
	Key generation	File tag generation		
Scheme [14]	$n(2h_e + 2h_{m^*} + h_h + h_a)$	h_h	$n((d+1)h_{m^*} + h_h + (d+1)h_e + h_{ec})$	-
Scheme [15]	$(n+2)h_h + h_m + 2h_e$	h_h	$n(h_{ec} + (d+1)h_{m^*} + h_h + (d+1)h_e)$	$n(h_m + h_h + h_a + h_{ce}) + h_e$
Scheme [24]	$nh_h + h_e + h_{bm}$	h_h	$n(h_h + h_{m^*} + h_{ec} + 2h_e)$	$n(h_h + h_{m^*} + h_{ec} + 2h_e)$
Scheme [16]	nh_h	$h_h + h_e$	$2n(h_h + h_e) + n(h_{m^*} + h_{ec})$	$l_1(2h_{pr} + h_{ec} + h_h + h_m + h_a)$
Our Scheme	$(n+1)(h_{pr} + (t+1)h_e + 2h_h + t(h_m + h_{bm}))$	$2h_e + h_{m^*}$	$n(h_h + h_{m^*} + 2h_e + dh_a + h_{ec})$	$l_1(2h_{pr} + h_{ec} + dh_a + h_{m^*} + h_h + h_e) + dh_m$

TABLE 6: Comparisons of computation cost in data auditing.

Scheme	User	CSP	TPA
Scheme [14]	h_e	$l_2(d(h_m + h_a) + 2(h_{pr} + h_m + h_e + h_{m^*}))$	$2h_{bm} + h_{m^*} + l_2(2h_{pr} + h_h + (d+1)(h_e + h_{m^*}))$
Scheme [15]	-	$l_2(2h_{pr} + h_e + h_{m^*} + d(h_m + h_a))$	$l_2(2h_{pr} + h_h + (d+1)(h_e + h_{m^*})) + 2h_{bm}$
Scheme [24]	$l_2 h_{pr}$	$l_2(2h_{pr} + h_m + h_a + h_e + h_{m^*}) + h_h + h_{m^*}$	$l_2(h_h + h_e + h_{m^*}) + h_h + 3h_e + 2h_{bm} + h_{m^*}$
Scheme [16]	$nh_{m^*} + (n+1)h_e$	$l_2(2h_{pr} + h_h + h_m + h_a)$	$l_2(2(h_{pr} + h_e + h_{m^*}) + h_h) + 2h_{bm} + h_e + h_{m^*}$
Our Scheme	-	$l_2(2h_{pr} + 2h_e + dh_a + h_m + h_h) + h_m + 2h_e$	$l_2(h_{pr} + h_m + h_a) + h_e + h_m + 2h_{bm}$

signature values with $tn|\mathbb{G}_1|$ bits from the blockchain. The user also downloads the ciphertext with $|C|$ bits from the cloud.

6.2.3 Storage Cost

Table 8 shows the storage cost for different schemes. After outsourcing a file, a user only needs to store the file tag and the seed of random number generator, which occupies $|\mathbb{G}_1| + |\theta|$ bits. The CSP stores the ciphertext, the authentication tags, the file tag, and the re-signature key of the user, which occupies $|C| + 2|\mathbb{Z}_N| + (n+1)|\mathbb{G}_1|$ bits in total. The blockchain records all the data of the auditing process and the blind signatures in the key generation process, which occupies $(I(n+1)+5)|\mathbb{G}_1| + (l_2+2)|\mathbb{Z}_N|$ bits.

The comparison results show that our scheme slightly increases the computation cost and communication cost compared to the schemes in [14–16, 24] in some operations. This is because our scheme is designed to protect OP and ensure the reliability of auditing results for low-entropy data, and the other schemes cannot. Besides, although the cost of some operations slightly increases, the overall cost does not increase significantly, which is also verified by the simulation results in Section 6.3.

6.3 Simulation Results

In this section, we simulate our scheme and analyze its actual performance. We use C++ to implement the off-chain programs of our scheme with Pairing Based Cryptography (PBC) Library [54], GNU Multiple Precision Arithmetic (GMP) Library [55] and OpenSSL Library [56]. We test the off-chain computation cost of users using a windows laptop with 2.30GHz Intel Core i5-6200U CPU and 8GB memory. Our on-chain data is recorded in the Ethereum [51], and we use the gas consumption to estimate the on-chain operations. We use Solidity to implement a simple smart contract,

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3237221

TABLE 7: Comparisons of user communication overhead.

Scheme	Key generation with key servers	Initial upload with CSP	Subsequent upload with CSP	Auditing with TPA	Data retrieval from CSP	Data retrieval from blockchain
Scheme [14]	-	$\|C\|+(n+1)\|\mathbb{G}_1\|+3\|\mathbb{Z}_N\|$	$3\|\mathbb{Z}_N\|+\|\mathbb{G}_1\|$	$\|\mathbb{Z}_N\|+3\|\mathbb{G}_1\|$	$\|C\|$	-
Scheme [15]	-	$\|C\|+(n+4)\|\mathbb{G}_1\|+3\|\mathbb{Z}_N\|$	$5\|\mathbb{Z}_N\|+8\|\mathbb{G}_1\|$	$2\|\mathbb{G}_1\|$	$\|C\|$	-
Scheme [24]	-	$\|C\|+n\|\mathbb{G}_1\|+2\|\mathbb{Z}_N\|$	$\|C\|+n\|\mathbb{G}_1\|+2\|\mathbb{Z}_N\|$	$3\|\mathbb{Z}_N\|$	$\|C\|$	-
Scheme [16]	-	$\|C\|+(n+1)\|\mathbb{G}_1\|+nk$	$\|\mathbb{G}_1\|+(n+2)\|\mathbb{Z}_N\|$ $+nk$	$\|\mathbb{G}_1\|$	$\|C\|+nk$	-
Our Scheme	$(2n+2)I\|\mathbb{G}_1\|$	$\|C\|+(n+2)\|\mathbb{G}_1\|+2\|\mathbb{Z}_N\|$	$5\|\mathbb{Z}_N\|+3\|\mathbb{G}_1\|$	$2\|\mathbb{G}_1\|$	$\|C\|$	$(tn+5)\|\mathbb{G}_1\|$ $+(l_2+2)\|\mathbb{Z}_N\|$

TABLE 8: Comparisons of storage cost.

Scheme	User	CSP	Blockchain
Scheme [14]	$\|\mathbb{Z}_N\|+\|\mathbb{G}_1\|+nk$	$\|C\|+3\|\mathbb{Z}_N\|+n\|\mathbb{G}_1\|$	-
Scheme [15]	$\|\mathbb{Z}_N\|+2\|\mathbb{G}_1\|+nk$	$\|C\|+2\|\mathbb{Z}_N\|+(n+4)\|\mathbb{G}_1\|$	-
Scheme [24]	$2\|\mathbb{Z}_N\|+nk$	$\|C\|+2\|\mathbb{Z}_N\|+n\|\mathbb{G}_1\|$	$(2l_2+2)\|\mathbb{Z}_N\|+2\|\mathbb{G}_1\|+\|\mathbb{G}_2\|$
Scheme [16]	$\|\mathbb{Z}_N\|+\|\mathbb{G}_1\|+k$	$\|C\|+\|\mathbb{Z}_N\|+(n+1)\|\mathbb{G}_1\|+nk$	$4\|\mathbb{Z}_N\|+2\|\mathbb{G}_1\|$
Our Scheme	$\|\mathbb{G}_1\|+\|\theta\|$	$\|C\|+2\|\mathbb{Z}_N\|+(n+1)\|\mathbb{G}_1\|$	$(I(n+1)+5)\|\mathbb{G}_1\|+(l_2+2)\|\mathbb{Z}_N\|$

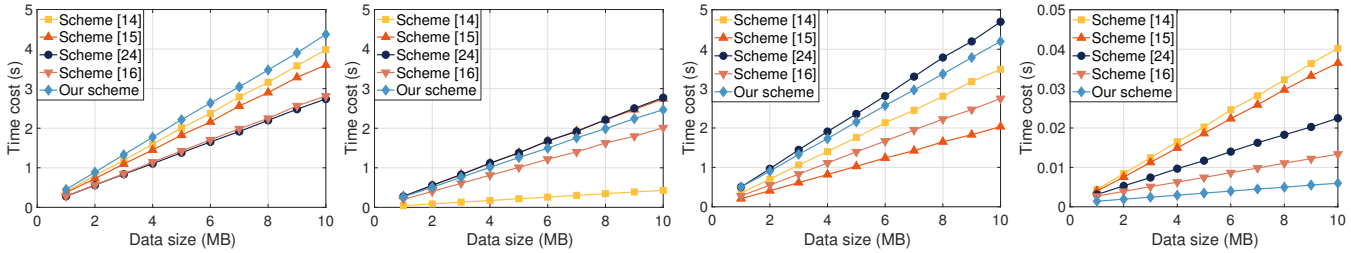


Fig. 3: Comparisons of actual off-chain computation cost for different auditable and secure deduplication schemes. (a) The computation cost of initial uploader; (b) the computation cost of subsequent uploader; (c) the computation cost of CSP in proof generation during auditing process; (d) the computation cost of TPA in proof verification during auditing process.

and use Web3j Library [57] to implement an interface to call data uploading and downloading functions.

During the simulation, the κ inputted by CSP is set as 512. We instantiate the bilinear pairing using the A1 type and encrypt the outsourced file using AES-256. The number of key servers (i.e., I) is set as 10 and the threshold t is set as 5. Besides, we execute all experiments 20 times to obtain an average result.

6.3.1 *Off-chain Computation Cost*

Here, we evaluate the actual computation cost of the initial uploader, subsequent uploader, CSP and TPA. We assume that the $1/3$ blocks are challenged in PoW and auditing, and set the block size as 1KB and the sector size as 128B.

Fig. 3 shows the computation cost of the initial uploader, subsequent uploader, CSP and TPA for different schemes. The spent time in Fig. 3(a) is the total time of one-time file uploading for the initial uploader. These operations include the key generation, file tag generation, encryption and authentication tags generation. The actual time cost of the initial uploader is close to that in [14–16, 24]. However, the scheme in [24] should limit the block size while the schemes in [14, 16] require users to stay online in auditing processes.

Fig. 3(b) is the total time of one-time file uploading for the subsequent uploader. The operations include key generation, file tag generation, and proof generation for PoW. It shows that the subsequent uploader in the scheme [14] needs significantly lower time than other schemes. This is because the scheme in [14] does not consider the file ownership security and the subsequent uploader does not need to compute ownership proof.

For the computation cost of the CSP shown in Fig. 3(c), our scheme has a slightly higher computation cost than the schemes in [14–16] during the integrity proof generation process, because our scheme needs an additional re-signature operation to protect the OP of users. However, in a cloud-based service, one cares more about the efficiency of local users, rather than cloud servers, since the cloud server usually has a large computation ability and these computa-

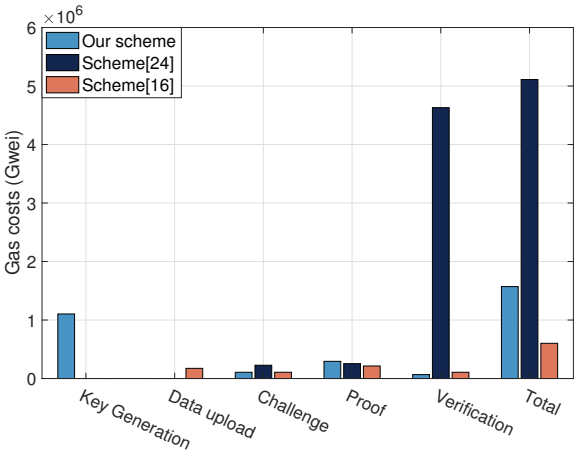


Fig. 4: Comparisons of on-chain operation cost for different auditable and secure deduplication schemes.

tions cannot cause a significant burden to it. Besides, from Fig. 3(d), our scheme has the lowest verifying time on TPA during the auditing process. As a result, the computation cost of our scheme is very similar compared to the previous state-of-the-art similar schemes. However, our scheme can achieve more security performance, as shown in Table 4.

6.3.2 On-chain Operation Cost

We also test the on-chain gas cost of our scheme. The experiment is designed under the situation that a user uploads and audits a file with size 10MB and 10 blocks. Fig. 4 shows the experiment results. Since only the schemes in [16, 24] are built based on blockchain, we compare our scheme with the schemes in [16, 24]. In the key generation process, to ensure the recoverability of encryption keys and reduce the key storage cost of users, our scheme publishes the blind signatures of data hash values on-chain, and thus has a gas cost. In the process of data uploading, since our scheme and the scheme in [24] do not need to create index tables on-chain, they have no gas cost, whereas the scheme in [16] has. In the challenging process, the gas cost of the scheme in [24] is higher than the other two schemes. In the proof process, our scheme has a higher gas cost than the other two schemes. Besides, in the verification process, the gas cost of the scheme in [24] is far higher than other schemes, because lots of on-chain computations are involved in the scheme in [24]. As a result, the total gas cost of the scheme in [24] is the highest, and our scheme has a slightly higher gas cost than the previous state-of-the-art scheme.

7 Conclusion and Future Work

In this paper, we proposed a new auditable and secure deduplication scheme for cloud storage. The users can audit the integrity of their outsourced files, while the CSP can deduplicate both the outsourced encrypted files and authentication tags on the cloud. Our scheme can protect the OP of users and ensure the reliability of auditing results for low-entropy data. By introducing the blockchain, our scheme can greatly reduce the key storage cost for users, guarantee key recoverability, and solve the trust issues between the users and TPA. Besides, our scheme can also protect users from losing data under DFA. We theoretically prove the correctness and security of our scheme. Analysis and experimental results show that our scheme can protect the OP of users and ensure the reliability of auditing results for low-entropy data with modest computation costs. However, the previous schemes cannot achieve these security effects. Since a user usually stores a large number of files to the cloud and audits these files one-by-one is time-consuming and inconvenient, our future work will extend the functionality of our scheme by supporting batch auditing.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62071142, in part by the Guangdong Basic and Applied Basic Research Foundation under Grants 2021A1515110027 and 2021A1515011406, and in part by the Shenzhen College Stability Support Plan under grant GXWD20201230155427003-20200824210638001.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, "Data age 2025: the digitization of the world from edge to core," *Seagate*, 2018.
[2] J. Gants, "Digital universe decade-are you ready?" *http://idcdocserv.com/925*, 2010.
[3] D. Quick and K.-K. R. Choo, "Google drive: Forensic analysis of data remnants," *Journal of Network and Computer Applications*, vol. 40, pp. 179–193, 2014.
[4] A. Dropbox, "file-storage and sharing service.(2016)."
[5] M. Mozy, "A file-storage and sharing service.(2016)."
[6] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2013, pp. 296–312.
[7] S. Keelveedhi, M. Bellare, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 179–194.
[8] 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*, 2015, pp. 874–885.
[9] J. Li, C. Qin, P. P. Lee, and X. Zhang, "Information leakage in encrypted deduplication via frequency analysis," in *2017 47th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2017, pp. 1–12.
[10] J. Li, J. Li, D. Xie, and Z. Cai, "Secure auditing and deduplicating data in cloud," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2386–2396, 2015.
[11] Y. Du, H. Duan, A. Zhou, C. Wang, M. H. Au, and Q. Wang, "Enabling secure and efficient decentralized storage auditing with blockchain," *IEEE Transactions on Dependable and Secure Computing*, 2021, to appear, doi:10.1109/TDSC.2021.3081826.
[12] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *2010 proceedings ieee infocom*. IEEE, 2010, pp. 1–9.
[13] D. Vasilopoulos, M. Önen, K. Elkhiyaoui, and R. Molva, "Message-locked proofs of retrievability with secure deduplication," in *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*, 2016, pp. 73–83.
[14] X. Liu, W. Sun, W. Lou, Q. Pei, and Y. Zhang, "One-tag checker: Message-locked integrity auditing on encrypted cloud deduplication storage," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.
[15] X. Gao, J. Yu, W.-T. Shen, Y. Chang, S.-B. Zhang, M. Yang, and B. Wu, "Achieving low-entropy secure cloud data auditing with file and authenticator deduplication," *Information Sciences*, vol. 546, pp. 177–191, 2021.
[16] G. Tian, Y. Hu, J. Wei, Z. Liu, X. Huang, X. Chen, and W. Susilo, "Blockchain-based secure deduplication and shared auditing in decentralized storage," *IEEE Transactions on Dependable and Secure Computing*, 2021, to appear, doi:10.1109/TDSC.2021.3114160.

[17] C. Wang, S. S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE transactions on computers*, vol. 62, no. 2, pp. 362–375, 2011.

[18] C. Zhang, Y. Xu, Y. Hu, J. Wu, J. Ren, and Y. Zhang, "A blockchain-based multi-cloud storage data auditing scheme to locate faults," *IEEE Transactions on Cloud Computing*, 2021, to appear, doi:10.1109/TCC.2021.3057771.

[19] W. Shen, J. Qin, J. Yu, R. Hao, J. Hu, and J. Ma, "Data integrity auditing without private key storage for secure cloud storage," *IEEE Transactions on Cloud Computing*, vol. 9, no. 4, pp. 1408–1421, 2019.

[20] Y. Su, J. Sun, J. Qin, and J. Hu, "Publicly verifiable shared dynamic electronic health record databases with functional commitment supporting privacy-preserving integrity auditing," *IEEE Transactions on Cloud Computing*, 2020, to appear, doi:10.1109/TCC.2020.3002553.

[21] J. Li, H. Yan, and Y. Zhang, "Efficient identity-based provable multi-copy data possession in multi-cloud storage," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 356–365, 2022.

[22] J. Yuan and S. Yu, "Secure and constant cost public cloud storage auditing with deduplication," in *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2013, pp. 145–153.

[23] Y. Xu, C. Zhang, G. Wang, Z. Qin, and Q. Zeng, "A blockchain-enabled deduplicatable data auditing mechanism for network storage services," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1421–1432, 2021.

[24] H. Yuan, X. Chen, J. Wang, J. Yuan, H. Yan, and W. Susilo, "Blockchain-based public auditing and secure deduplication with fair arbitration," *Information Sciences*, vol. 541, pp. 409–425, 2020.

[25] Y. Duan, "Distributed key generation for encrypted deduplication: Achieving the strongest privacy," in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, 2014, pp. 57–68.

[26] Y. Zhang, C. Xu, N. Cheng, and X. Shen, "Secure password-protected encryption key for deduplicated cloud storage systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2789–2806, 2021.

[27] X. Yang, R. Lu, J. Shao, X. Tang, and A. Ghorbani, "Achieving efficient secure deduplication with user-defined access control in cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 591–606, 2022.

[28] X. Yang, R. Lu, J. Shao, X. Tang, and A. A. Ghorbani, "Achieving efficient and privacy-preserving multi-domain big data deduplication in cloud," *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1292–1305, 2021.

[29] H. Kwon, C. Hahn, K. Kang, and J. Hur, "Secure deduplication with reliable and revocable key management in fog computing," *Peer-to-Peer Networking and Applications*, vol. 12, no. 4, pp. 850–864, 2019.

[30] H. Kwon, C. Hahn, D. Koo, and J. Hur, "Scalable and reliable key management for secure deduplication in cloud storage," in *2017 IEEE 10th international conference on cloud computing (CLOUD)*. IEEE, 2017, pp. 391–398.

[31] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE transactions on parallel and distributed systems*, vol. 25, no. 6, pp. 1615–1625, 2013.

[32] Y. Zhou, D. Feng, W. Xia, M. Fu, F. Huang, Y. Zhang, and C. Li, "Secdep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management," in *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2015, pp. 1–14.

[33] 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 *Proceedings 22nd international conference on distributed computing systems*. IEEE, 2002, pp. 617–624.

[34] Q. Xie, C. Zhang, and X. Jia, "Security-aware and efficient data deduplication for edge-assisted cloud storage systems," *IEEE Transactions on Services Computing*, 2022, to appear, doi:10.1109/TSC.2022.3195318.

[35] T. Jiang, X. Yuan, Y. Chen, K. Cheng, L. Wang, X. Chen, and J. Ma, "Fuzzydedup: Secure fuzzy deduplication for cloud storage," *IEEE Transactions on Dependable and Secure Computing*, 2022, to appear, doi:10.1109/TDSC.2022.3185313.

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

[37] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 491–500.

[38] G. Tian, H. Ma, Y. Xie, and Z. Liu, "Randomized deduplication with ownership management and data sharing in cloud storage," *Journal of Information Security and Applications*, vol. 51, p. 102432, 2020.

[39] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 598–609.

[40] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, 2008, pp. 1–10.

[41] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE transactions on parallel and distributed systems*, vol. 22, no. 5, pp. 847–859, 2010.

[42] A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 584–597.

[43] H. Shacham and B. Waters, "Compact proofs of retrievability," in *International conference on the theory and application of cryptology and information security*. Springer, 2008, pp. 90–107.

[44] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptology and information security*. Springer, 2001, pp. 514–532.

[45] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE transactions on parallel and distributed systems*, vol. 24, no. 9, pp. 1717–1726, 2012.

[46] Y. Zhang, C. Xu, N. Cheng, and X. S. Shen, "Secure password-protected encryption key for deduplicated cloud storage systems," *IEEE Transactions on Dependable and Secure Computing*, 2021, to appear, doi:10.1109/TDSC.2021.3074146.

[47] S. Li, C. Xu, Y. Zhang, Y. Du, and K. Chen, "Blockchain-based transparent integrity auditing and encrypted deduplication for cloud storage," *IEEE Transactions on Services Computing*, 2022, to appear, doi:10.1109/TSC.2022.3144430.

[48] M. Miao, J. Wang, H. Li, and X. Chen, "Secure multi-server-aided data deduplication in cloud computing," *Pervasive and Mobile Computing*, vol. 24, pp. 129–137, 2015.

[49] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of cryptography conference*. Springer, 2007, pp. 535–554.

[50] N. P. Smart, "The discrete logarithm problem on elliptic curves of trace one," *Journal of cryptology*, vol. 12, no. 3, pp. 193–196, 1999.

[51] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[52] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[53] 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, 2016.

[54] B. Lynn, "The pairing-based cryptography (pbc) library," 2010.

[55] T. Granlund, "Gnu multiple precision arithmetic library," *http://gmplib. org/*, 2010.

[56] "Openssl," [Online], https://www.openssl.org/.

[57] "Web3j library," [Online], https://github.com/web3j/web3j.

Mingyang Song received his B.E. and M.E. degrees in software engineering from Sun Yat-sen University, Guangzhou, China, in 2019 and 2021, respectively. He is currently pursuing the Eng.D. degree with the Department of Electronic Information, Harbin Institute of Technology, Shenzhen. His research interests include security and privacy related to cloud computing, applied cryptography, and blockchain.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3237221

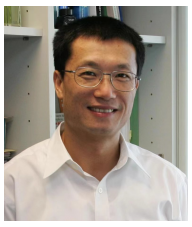
Zhongyun Hua (Member, IEEE) received the B.S. degree in software engineering from Chongqing University, Chongqing, China, in 2011, and the M.S. and Ph.D. degrees in software engineering from the University of Macau, Macau, China, in 2013 and 2016, respectively.

He is currently an Associate Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include chaotic system, chaos-based applications, data hiding, and multimedia security. He has published more than sixty papers on the subject, receiving more than 3900 citations.

Yifeng Zheng is an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He received the PhD degree in computer science from the City University of Hong Kong, Hong Kong, in 2019. He worked as a postdoc with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia and City University of Hong Kong. His work has appeared in prestigious venues such as ESORICS, DSN, ACM AsiaCCS, IEEE INFOCOM, IEEE ICDCS, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Information Forensics and Security, and IEEE Transactions on Services Computing. He received the Best Paper Award in the European Symposium on Research in Computer Security (ESORICS) 2021. His current research interests are focused on security and privacy related to cloud computing, IoT, machine learning, and multimedia.

Hejiao Huang received the B.S. and M.S. degrees in mathematics from Shaanxi Normal University, Xi'an, China, in 1996 and 1999, respectively, and the Ph. D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2004.

She was an Invited Professor with INRIA, Bordeaux, France. She is currently a Professor with the Harbin Institute of Technology, Shenzhen, China. Her research interests include cloud computing, network security, trustworthy computing, and formal methods for system design and wireless networks.



XiaoHua Jia (F'13) received the BSc and MEng degrees in 1984 and 1987, respectively, from the University of Science and Technology of China, and the DSc degree in 1991 in information science from the University of Tokyo. He is currently the chair professor with the Department of Computer Science at the City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks and mobile wireless networks. He is an editor of the *IEEE Transactions on Parallel and Distributed Systems (2006-2009), Wireless Networks, Journal of World Wide Web, Journal of Combinatorial Optimization, etc.* He is the general chair of ACM MobiHoc 2008, TPC co-chair of IEEE MASS 2009, area-chair of IEEE INFOCOM 2010, TPC co-chair of IEEE GlobeCom 2010-Ad Hoc and Sensor Networking Symposium, and Panel co-chair of IEEE INFOCOM 2011. He is a fellow of the IEEE Computer Society.