Blockchain-Based Privacy-Preserving Deduplication and Integrity Auditing in Cloud Storage

Qingyang Zhang, *Member, IEEE*, Shuai Qian, Jie Cui, *Senior Member, IEEE*, Hong Zhong, *Member, IEEE*, Fengqun Wang, and Debiao He, *Member, IEEE*

***Abstract*—Ensuring cloud data security and reducing cloud storage costs have become particularly important. Many schemes expose user file ownership privacy when deduplicating authentication tags and during integrity auditing. Moreover, key management becomes more difficult as the number of files increases. Also, many audit schemes rely on third-party auditors (TPAs), but finding a fully trustworthy TPA is challenging. Therefore, we propose a blockchain-based integrity audit scheme supporting data deduplication. It protects file tag privacy during deduplication of ciphertexts and authentication tags, safeguards audit proof privacy, and effectively protects user file ownership privacy. To reduce key management costs, we introduce identity-based broadcast encryption (IBBE) that does not require interaction with key servers, eliminating additional communication costs. Additionally, we use smart contracts for integrity auditing, eliminating the need for a fully trusted TPA. We evaluate the proposed scheme through security and theoretical analyses and a series of experiments, demonstrating its efficiency and practicality.**

***Index Terms*—Data deduplication, tag deduplication, integrity auditing, blockchain.**

I. Introduction

An *Internet Data Center* (IDC) survey revealed that an individual's data volume can reach up to 5200 GB [1], and the high cost of local storage for such significant amounts of data has spurred an increase in cloud storage. According to the IDC's latest report, by 2027, global public cloud services are projected to generate \$1.34 trillion in total revenue [2]. Although cloud-storage services offer considerable convenience, incidents of data loss occur frequently. For instance, the Ernst and Young accounting firm was the victim of a ransomware attack by the LockBit organization in 2021, resulting in substantial theft of data [3]. In the absence of effective auditing strategies, a cloud service provider (CSP) may claim that data are securely stored to maintain its reputation. Furthermore, the IDC survey indicates that nearly 75% of the data outsourced to CSPs consist of redundant duplicates [4], which significantly waste CSP storage space. Therefore, to enhance the reliability and storage efficiency of cloud storage systems, integrity auditing technologies are crucial in verifying the correctness of the stored data, and data deduplication techniques are required to remove redundant files from the cloud.

As mentioned previously, the large number of redundant data files in cloud storage causes a significant waste of CSP storage space. The elimination of redundant files through deduplication can significantly reduce CSP storage costs. Therefore, to enhance the efficiency of cloud storage, data deduplication has been proposed. Currently, there are some related works and they also achieve certain results [5]. In their schemes, files are primarily subjected to convergent encryption, which allows identical files to be encrypted into the same ciphertext, enabling CSPs to perform redundant file deduplication. Convergent encryption involves using the plaintext hash of a file as the key for encrypting the file. However, these solutions have shortcomings, such as the presence of redundant file authentication tags.

Typically, the tag size is linearly related to the number of files and can even be larger than the files themselves. Therefore, further deduplication of authentication tags can significantly reduce CSP storage costs. Consequently, in some existing studies [6], [7], the hash value of a file is used as the signature key to generate authentication tags, thereby enabling the deduplication of duplicate tags. However, the same file tag is used during the auditing process, which can reveal the users who own the same files, leading to privacy leakage. Therefore, privacy protection measures should be applied to file labels during duplicate authentication tag deduplication. Moreover, most auditing schemes introduce blockchain technology to reduce reliance on fully trusted third-party audits (TPAs). However, in many current schemes [7], [8], storing audit proofs on a public blockchain still poses privacy leakage issues because the audit proofs for the same file are publicly known on the blockchain.

Received 28 September 2024; revised 27 January 2025; accepted 5 February 2025. Date of publication 11 February 2025; date of current version 9 April 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62202005, Grant U24A20243, Grant 62372002, and Grant 62272002; in part by the Natural Science Foundation of Anhui Province, China, under Grant 2208085QF198; and in part by the University Synergy Innovation Program of Anhui Province under Grant GXXT-2022-049. Recommended for acceptance by X. Jia. *(Corresponding author: Jie Cui.)*

Qingyang Zhang, Shuai Qian, Jie Cui, Hong Zhong, and Fengqun Wang are with the Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, School of Computer Science and Technology and Anhui Engineering Laboratory of IoT Security Technologies, Anhui University, Hefei 230039, China (e-mail: cuijie@mail.ustc.edu.cn).

Debiao He is with the School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China, and also with Shanghai Key Laboratory of Privacy Preserving Computation, MatrixElements Technologies, Shanghai 201204, China (e-mail: hedebiao@163.com).

Digital Object Identifier 10.1109/TC.2025.3540670

This also exposes the users who possess the file, thereby leaking the privacy of user file ownership. Therefore, in addition to the deduplication of duplicate files, further deduplication of authentication tags and protection of user file ownership privacy have become matters of significant importance.

In addition, the cost of key management for users is challenging. In many schemes [5], [6], [7], data owners hash the file or data block to obtain convergent keys, which are subsequently used to encrypt the file or data block. In this approach, clearly as the number of files outsourced to the cloud increases, the number of encryption keys that data owners need to store also increases; thus, data owners bear the burden of managing many encryption keys. Li et al. [9] and Zheng et al. [10] introduced a key server to help recover keys and alleviate the burden of key storage. However, this method requires frequent interactions between the data owner and key server.

To address these issues, we propose a blockchain smart-contract-based solution for privacy-preserving data integrity auditing and deduplication. The main contributions of this study are as follows.

- An auditing framework based on identity-based broadcast encryption and blockchain technology is proposed to address the need for integrity auditing technology without relying on TPA and alleviate user key management. The proposed framework leverages blockchain technology to avoid dependence on a fully trusted TPA. Additionally, IBBE technology is used for key management to eliminate reliance on a key management server.
- A scheme that randomizes file tags and audit proofs is proposed to address the leakage of user file ownership privacy. Even if file tags are public on the blockchain, they do not reveal user privacy. Additionally, the proposed scheme supports the deduplication of redundant authentication tags in cloud storage, further reducing CSP storage costs.
- A security analysis of the proposed scheme is conducted to ensure its theoretical feasibility. Subsequently, experimental tests are conducted on the proposed method. The results demonstrate the security and efficiency of the proposed scheme.

II. Related Work

A. Data Auditing Supporting Deduplication

To ensure the integrity of outsourced data and enhance the efficiency of cloud storage, a series of schemes for deduplication and data integrity auditing are proposed. In 2013, Yuan et al. [11] were the first to propose a cloud storage scheme for data integrity auditing based on data deduplication. However, their scheme requires all users to compute and upload identity verification tags for the same file, leading to high local computational costs. Xu et al. [7] utilized audit logs on the blockchain to supervise the partially trusted TPA, their scheme only allows for deduplication and auditing of plaintext, sacrificing the confidentiality of uploaded data.

Li et al. [12] proposed a cloud storage scheme that employs a trusted proxy server to generate authentication tags for the deduplication and auditing of encrypted data. However, relying on such a trusted proxy server is a strong assumption and extremely costly to implement in an insecure public network, it is hard to achieve in practice. Subsequently, Liu et al. [13] proposed a scheme without the need for a trusted proxy server, using the MLE key as the key for authentication tags to achieve tag deduplication. However, this method does not ensure the accuracy of audit results when data is low entropy. Later, Gao et al. [14] suggested using the initial uploader's key to replace the MLE key, but this only ensures the correctness of the audit results for the initial uploader, without guaranteeing the reliability of subsequent uploaders' audit results.

To reduce the excessive reliance on third-party auditors during the auditing process, Yuan et al. [5] used smart contracts to perform audit tasks, proposing a blockchain-based data deduplication scheme. However, in this scheme, the number of authenticators is linearly correlated with the number of files, which increases the storage overhead for the CSP. Tian et al. [6] solved the single-point failure problem by utilizing two SSPs, and implemented mutual audits between the two cloud servers. However, in order to periodically update verification tags, users must remain online.

In conclusion, these existing traditional schemes still have deficiencies. Some incur high local computational costs, some compromise the confidentiality of data, and others cannot eliminate interactions with key servers, among other issues. Additionally, these schemes do not take into account the protection of users' data ownership privacy.

B. Data Integrity Auditing

Deswarte et al. [15] introduced the "challenge-response" model to verify the integrity of outsourced data. Building on this, Ateniese et al. [16] proposed the concept of Provable Data Possession (PDP), enabling efficient remote data auditing through random sampling. Subsequently, a variety of public auditing schemes are introduced, including those based on certificate auditing, certificate-less auditing, and multi-cloud auditing. Among these, the public verifiable Provable Data Possession (PDP) scheme proposed by Wang et al. [17] leverages a fully trusted TPA to significantly reduce the auditing burden for users. However, the excessive dependence on Third-Party Auditors emerged as a pivotal concern in most existing auditing schemes, prompting the development of various decentralized integrity auditing approaches. Yang et al. [18] introduced a data integrity auditing scheme in a multi-copy, multi-cloud environment, while Jiang et al. [19] introduced another strategy using identity-based encryption methods, and Zhang et al. [20] proposed an efficient integrity auditing mechanism and secure deduplication scheme for blockchain storage, utilizing blockchain smart contracts to perform auditing tasks. These methods [18], [19], [20], along with other related research [21] effectively reduce dependency on TPA.

In addressing the restoration of damaged data, Juels et al. [16] introduced an innovative cryptographic primitive known as Proof-of-Retrievability (PoR), which utilizes the sentinel method. Subsequently, Shacham and Waters [22] enhanced this scheme by integrating erasure coding and BLS signatures [23] into their approach.

Many traditional auditing schemes primarily focus on the integrity of files being transferred to the cloud. However, for practical application scenarios, it is also essential to address the issue of eliminating redundant files in the cloud to save more storage space for the CSP.

C. Deduplication of Redundant Data

To enhance cloud storage efficiency while ensuring the confidentiality of outsourced data, secure data deduplication is an important and effective strategy. In traditional encryption methods, different users encrypt the same file into different ciphertexts using distinct encryption strategies, which hampers subsequent deduplication efforts. To facilitate privacy-preserving data deduplication, Douceur et al. [24] introduced convergent encryption, which produces a unique and fixed ciphertext for any given input file. Following this, several variants of convergent encryption are proposed [9], [12], [25], [26], [27]. Bellare and others formalized convergent encryption and its variants as the cryptographic primitive known as "Message-Locked Encryption" (MLE) [25]. To strengthen the resistance of MLE algorithms against brute-force attacks and reduce the user's key management overhead, some solutions introduce key servers to assist in generating MLE keys. Specifically, Zhang et al. [28] implemented an effective data deduplication function within the JointCloud system using a ring key mechanism. In the research by Zheng et al. [10], media data deduplication was facilitated through server-assisted MLE, which, however, presents a risk of single-point failure. Subsequent studies suggest the use of a group of key servers, Zhang et al. [28] proposed an activation mechanism that periodically changes group members, yet the dependency on key servers remains a challenge.

If existing data integrity auditing schemes are directly applied to deduplication schemes, the same file might generate numerous redundant authentication tags, unnecessarily occupying substantial storage space of the CSP. Moreover, many auditing schemes rely on third-party auditors, which, when integrated with deduplication schemes, inevitably expose users' file ownership privacy to the auditors. Some auditing schemes introduce blockchain to avoid reliance on centralized TPAs, but storing the audit results of the same files on the blockchain still poses privacy leakage issues.

III. Preliminaries

A. Bilinear Pairing

Given $\mathbb{G}_1$ and $\mathbb{G}_2$ are two finite cyclic multiplicative groups of prime order p, and $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear map with the following properties:

- ***Bilinearity***: For all $x, y \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, it holds that $e(x^a, y^b) = e(x, y)^{ab}$.
- ***Non-degeneracy***: If g is a generator of $\mathbb{G}_1$, then $e(g, g)$ is a generator of $\mathbb{G}_2$.
- ***Computability***: For all $x, y \in \mathbb{G}_1$, there exists an efficient algorithm to compute $e(x, y) \in \mathbb{G}_2$.

A related complexity assumption is given below.

Definition 1 (Discrete Logarithm (DL) Problem Complexity): Let $\mathbb{G}$ be a finite group of order p, and let $g \in \mathbb{G}$ and $x \in \mathbb{G}$ be given elements. The discrete logarithm challenge is to find an integer $k \in \mathbb{Z}_p$ such that $g^k = x$ within $\mathbb{G}$. This task's difficulty underpins the security of many cryptographic systems.

B. Identity-Based Broadcast Encryption (IBBE)

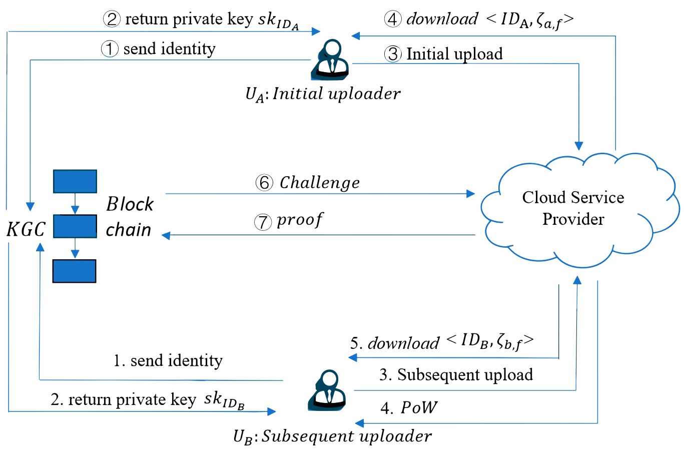
In Cécile Delerablée's scheme [29], the functionalities of Identity-Based Encryption (IBE) and Broadcast Encryption (BE) are combined, allowing for the encryption of messages designated for specific identity groups rather than individual recipients. The implementation of this IBBE scheme is pivotal for encrypting convergent keys, and operates as follows:

- $Setup(\lambda, u)$: Accepts the security parameter λ and a number of users u, and returns the master secret key msk and the public parameters pp.
- $Extract(ID_i, msk)$: Accepts the identity ID_i and the master secret key msk, and returns the secret key sk_{ID_i} for the identity ID_i.
- $Encrypt(pp, m, S)$: Accepts the public parameters pp, a message m, and a set of identities $S = \{ID_1, \ldots, ID_s\}$. It uses these inputs to generate a header Hdr and an encryption key K, and outputs the tuple (Hdr, S, C_M), where C_M is the encrypted message.
- $Decrypt(S, ID_i, sk_{ID_i}, Hdr, pp)$: Accepts a set of identities $S = \{ID_1, \ldots, ID_s\}$, an identity ID_i with its corresponding secret key sk_{ID_i}, the header Hdr, and public parameters pp. It decrypts the encryption key K, which is then used to decrypt the ciphertext C_M.

C. Blockchain

Blockchain networks consist of interconnected nodes that collectively adhere to a consensus protocol to maintain the ledger's integrity [30], [31]. Each record in this ledger, known as a block, contains a timestamp and an array of transactional data among other elements. The timestamp indicates the creation moment of the block. Typically, transactional data includes the details of the token exchanges between network participants. Blocks are securely linked in chronological order via cryptographic hashes, which fortify the blockchain against tampering and facilitate the verification of all stored data.

Smart contracts, self-executing contracts with the terms of the agreement directly written into code, are deployed on the blockchain to automate and enforce the integrity checks of data. When implemented for data integrity audits, smart contracts can programmatically verify the authenticity and completeness of the data stored in the blockchain without human intervention. These contracts can be triggered by predefined conditions, such as suspicious data alterations or periodic audits. Upon activation, the smart contract executes the necessary verification protocols.



② return private key sk_{ID_A}
① send identity
④ download $<ID_A, \zeta_{a,f}>$
③ Initial upload
U_A: Initial uploader
KGC
Block chain
⑥ Challenge
⑦ proof
Cloud Service Provider
5. download $<ID_B, \zeta_{b,f}>$
1. send identity
3. Subsequent upload
2. return private key sk_{ID_B}
4. PoW
U_B: Subsequent uploader

Fig. 1. System model and workflow diagram.

IV. Problem Formulation

In this part, we begin by presenting the scheme's system model, threat model, and design goals. This is followed by a brief summary of the scheme's framework.

A. System Model

The proposed scheme's system model comprises four entities: KGC, users, CSP and blockchain. Fig. 1 depicts the system model for the proposed scheme. We define them as follows:

- **KGC**: KGC stands for Key Generation Center, which generates a unique private key for each user joining the system.
- **User**: Users outsource encrypted data to the CSP. Since different users may outsource the same file, all users can be categorized into initial uploaders, who outsource files not yet stored by the CSP, and subsequent uploaders, who outsource files that have already been stored by the CSP.
- **CSP**: The CSP provides storage services to users and executes deduplication algorithms. When interacting with the subsequent uploader, the CSP initiates an ownership verification challenge to ensure that he is the true owner of the file data. During an audit, it receives challenge information to generate and publish audit proofs.
- **Blockchain**: The blockchain maintains a decentralized ledger system and supports smart contracts signed by data owners and the CSP for data integrity auditing.

Interaction logic of entities in the system model: The Key Generation Center (KGC) sends a private key to each user joining the system. After encrypting a file, the user uploads it to the Cloud Service Provider (CSP). The CSP determines whether the file is already stored to confirm whether the user is the initial uploader or a subsequent uploader and verifies the user's legitimacy. During file recovery, the encryption key for the file is reconstructed using the Identity-Based Broadcast Encryption (IBBE) method and the user's private key (assigned by the KGC), enabling decryption of the original file. For data auditing, a smart contract initiates an audit challenge for files stored on the CSP, and the audit results are published on the blockchain.

B. Threat Model

In this section, we introduce the threat model.

- **Semi-honest users**: Since the data on a blockchain is publicly accessible, there is a potential risk that users can snoop on the ownership privacy of other users in the blockchain.
- **Semi-honest CSP**: In cases where stored file data is lost due to hardware issues or other potential problems, the CSP, to maintain its reputation, might claim to have stored the data well and forge false audit proofs during an audit.

Furthermore, we assume that the CSP will not collude with any users.

C. Design Goals

In this paper, we design a deduplication scheme that also supports auditing operations. Based on this, we achieve the following design objectives:

- **Security of Outsourced Data**: Ensure that the CSP cannot recover the plaintext or infer the contents of the data files.
- **Deduplication**: To reduce storage overhead in the cloud, identical files and identity verification tags are not stored repeatedly.
- **Auditing**: Allow users to audit the integrity of their outsourced data, ensuring the reliability of the audit results. Moreover, instead of relying on a TPA, the auditing process is performed through a smart contract.
- **Privacy Protection of Identical Information**: Entities other than the CSP cannot determine which users possess the same files, protecting users' ownership privacy.
- **Key Management on the User Side**: Reduce the key management costs on the user side, ensuring that the number of keys stored by users is independent of the number of files they own.

V. The Proposed Scheme

A. Scheme Overview

In the scheme by Yang et al. [32], they use randomized file tags, building on this, we can further randomize the audit proofs to protect the privacy of user file ownership. Additionally, we use IBBE technology can avoid reliance on a key server. Table I summarizes the important notations used in this paper.

Initially, the KGC generates a private key for users joining the system. Before uploading files, users encrypt the files and upload them along with file tags. The CSP then determines if the outsourced file already exists. The initial uploader sends the ciphertext, file tags, authentication tags, and audit keys to CSP. Additionally, they apply broadcast encryption to key K_C, thus allowing other system users to decrypt it. Subsequent uploaders must accept an ownership verification challenge from the CSP. Upon passing the challenge, the CSP adds them to the file owner list. When recovering data, the user sends a download request to the CSP. The CSP first checks whether the user is legitimate, then returns the encrypted data to the user. Next, the user can recover the encryption key and decrypt the file. During

TABLE I
NOTATIONS AND THEIR DESCRIPTIONS

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 U for delegating auditing, and f represents the file F. The same applies below.
$ak_{u,f}$	The audit key of U for file F.
$\zeta_{u,f}$	The random file tag of F generated by U.
K_c	Encrypt the file encryption key.
p	A large prime number.
$\mathbb{Z}_p$	A residue class ring.
$\mathbb{G}_1, \mathbb{G}_2$	Two multiplicative cyclic groups.
k_l	The file encryption key.
s_F	The collection of user identities that possess the same file.
s	The number of sectors in a data block.
K_{IBBE}	A IBBE key, used for the encryption of convergent keys.
$H_1(\cdot), H_2(\cdot), H_3(\cdot), H_4(\cdot)$	Hash functions.
$\pi_1(\cdot), \pi_2(\cdot)$	Two pseudo-random functions.
$e(\cdot,\cdot)$	Bilinear pairing e: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.

an audit, the smart contract sends challenge information and other data to the CSP. After receiving the audit proof, it verifies its correctness and publishes the audit results.

B. System Setup

The trusted KGC generates private keys for users, and then generates other system parameters.

1) KGC System Initialization: Specifically, the KGC runs a parameter generator $Gen(\kappa)$, then it outputs a tuple ($\mathbb{G}_1$, $\mathbb{G}_2$, e), where $\mathbb{G}_1$ and $\mathbb{G}_2$ are two finite cyclic multiplicative groups of prime order p and e: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ denote a bilinear map, where p is a large prime number.

The KGC establishes the cryptographic framework by initializing the following parameters and protocols:

- A large prime number m is selected, along with a secret value γ from $\mathbb{Z}_p^*$, and generators g_1, g_2, h from $\mathbb{G}_1$.
- An element $\chi \in \mathbb{G}_1$ is defined, and $\varpi = g_2^{\gamma}$ is computed.
- The master key is established as γ.
- A unified chunking strategy is implemented to ensure that users with identical files obtain identical chunks.
- Four hash functions are selected: $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_3 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_4 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and dynamic pseudorandom functions are selected for different numbers of blocks n allocated to different files, where n refers to the number of blocks:

$$\pi_1 : \{1,\ldots,n\} \times \mathbb{Z}_p^* \rightarrow \{1,\ldots,n\},$$
$$\pi_2 : \{1,\ldots,n\} \times \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*.$$

The public parameters $\mathbf{pp}$ of the system are published:

$$\mathbf{pp} = (\varpi, v = e(g_2, h), h, h^{\gamma}, \ldots, h^{\gamma^m}, H_1, \ldots, H_4, \pi_1, \pi_2, \chi).$$

2) User Registration: When a new user U_t with identity ID_t joins the system, the KGC generates a secret key for them as follows:

$$sk_{ID_t} = g_2^{\frac{1}{\gamma + H_1(ID_t)}}.$$

In subsequent stages, users can use their private key sk_{ID_t} to recover the file encryption key.

C. Data Uploading

The data uploading algorithm is divided into two parts: initial upload and subsequent upload. A user uploads his file data to the CSP, which then checks if the file has already been stored, thus distinguishing the type of data upload. Specifically, data upload process is as follows: user U_t selects $x_t, s_t \in \mathbb{Z}_p$ and configures the audit key $ak_{t,f} = \{x_t, s_t\}$ and randomly selects $r_t \in \mathbb{Z}_p$. U_t computes $K_C = H_2(F)$ and the file tag $tag_{t,f} = g_1^{K_C}$, besides, U_t then computes a random file tag $\zeta_{t,f} = g_1^{K_C \cdot r_t}$. Then U_t sends the file tag $tag_{t,f}$ and the random file tag $\zeta_{t,f}$ to the CSP. The CSP checks whether the file F has already been stored by verifying whether the file tag $tag_{t,f}$ exists.

If, upon inspection, it is found that the file tag already exists, it indicates that the Cloud Service Provider has previously stored the same file. Consequently, a message stating "File Duplicate" will be returned to the uploader. Conversely, if, upon inspection, the file tag is found to be absent, it implies that the CSP has not stored the file previously, and a message stating "File Not Duplicate" will be returned to the uploader.

1) Initial Upload: Assuming the file uploaded by user U_A is verified by the cloud and has not yet been stored, consider U_A as the initial uploader; U_A collaborates with the CSP to carry out the protocol for the initial data file upload. U_A first randomly selects $k_l \in \mathbb{Z}_p^*$ as the file's encryption key, generates the ciphertext $c = Enc(k_l, F)$, and then encrypts k_l through $K_l = k_l \oplus K_C$, note that the key k_l here is integrated, and l does not have an independent meaning; it is only used to distinguish it from the notation of other keys. Subsequently, U_A calculates the public key $pk_{a,f} = g_1^{H_3(k_l) \cdot s_a}$ and divides the ciphertext c into n blocks where each block possess s sectors and then generate authentication tags $\{\tau_i\}_{1 \le i \le n}$ for each ciphertext block $\{c_{i,j}\}_{1 \le i \le n, 1 \le j \le s}$ as follows, where c_i is the i-th ciphertext block and s is the number of sectors in the block:

$$\tau_i = [H_4(c_i||i) \cdot \chi^{\sum_{j=1}^{s} c_{i,j}}]^{H_3(k_l)}, 1 \le i \le n \tag{1}$$

User U_A randomly selects $k_F \in \mathbb{Z}_p^*$ and generates the IBBE header $Hdr = (C_{F_1}, C_{F_2}) = (\varpi^{-k_F}, h^{(\gamma + H_1(ID_A)k_F)})$, then encrypts k_F using symmetric encryption: $d_F \leftarrow Enc(K_C, k_F)$, and encrypts K_C with the IBBE key $K_{IBBE} = v^{k_F}$ to produce ck_F. U_A sends $(ID_A, c, K_l, \{\tau_i\}, Hdr_F, d_F, ck_F, ak_{a,f}, pk_{a,f})$ to the CSP.

The CSP randomly selects a seed θ from Z_p^*, and subsequently generates a random number $a_i = \pi_2(i, \theta)$, where $1 \le i \le n$, and verifies whether the following equation holds:

$$e\left(\prod_{i=1}^{n}(H_4(c_i||i)^{a_i} \cdot \prod_{j=1}^{s} \chi^{a_i \cdot c_{i,j}}), pk_{a,f}\right) \stackrel{?}{=} e\left(\prod_{i=1}^{n} \tau_i^{a_i}, g_1^{s_a}\right) \tag{2}$$

If the verification passes, $H_1(ID_A)$ is added to s_F; Otherwise, the CSP rejects this storage request.

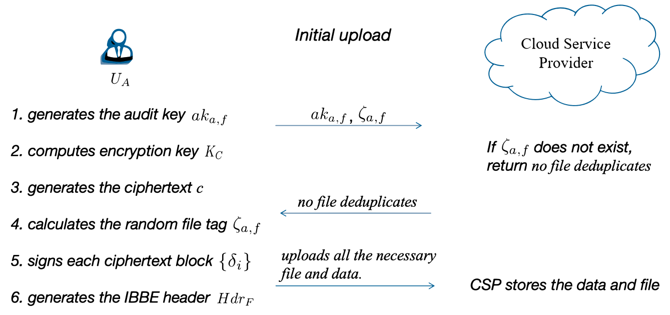

Initial upload
Cloud Service Provider
U_A
1. generates the audit key $ak_{a,f}$
2. computes encryption key K_C
3. generates the ciphertext c
4. calculates the random file tag $\zeta_{a,f}$
5. signs each ciphertext block $\{\delta_i\}$
6. generates the IBBE header Hdr_F
$ak_{a,f}, \zeta_{a,f}$
no file deduplicates
uploads all the necessary file and data.
If $\zeta_{a,f}$ does not exist, return no file deduplicates
CSP stores the data and file

Fig. 2. Initial upload.

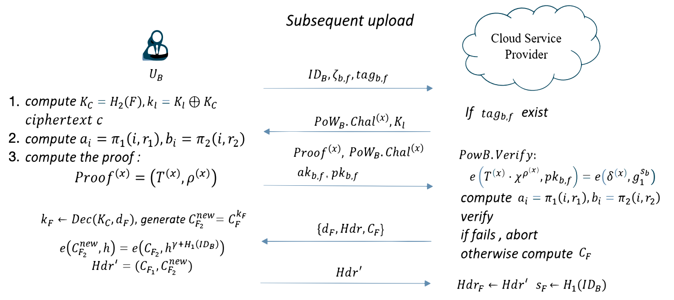

Subsequent upload
Cloud Service Provider
U_B
1. compute $K_C = H_2(F), k_l = K_l \oplus K_C$ ciphertext c
2. compute $a_i = \pi_1(i, r_1), b_i = \pi_2(i, r_2)$
3. compute the proof : $Proof^{(x)} = (T^{(x)}, \rho^{(x)})$
$k_F \leftarrow Dec(K_C, d_F)$, generate $C_{F_2}^{new} = C_F^{k_F}$
$e(C_{F_2}^{new}, h) = e(C_{F_2}, h^{\gamma + H_1(ID_B)})$
$Hdr' = (C_{F_1}, C_{F_2}^{new})$
$ID_B, \zeta_{b,f}, tag_{b,f}$
$PoW_B.Chal^{(x)}, K_l$
$Proof^{(x)}, PoW_B.Chal^{(x)}$
$ak_{b,f}, pk_{b,f}$
$\{d_F, Hdr, C_F\}$
Hdr'
If $tag_{b,f}$ exist
$PowB.Verify$:
$e(T^{(x)} \cdot \chi^{\rho^{(x)}}, pk_{b,f}) = e(\delta^{(x)}, g_1^{s_b})$
compute $a_i = \pi_1(i, r_1), b_i = \pi_2(i, r_2)$
verify
if fails , abort
otherwise compute C_F
$Hdr_F \leftarrow Hdr'$ $s_F \leftarrow H_1(ID_B)$

Fig. 3. Subsequent upload.

2) Subsequent upload: Assuming user U_B is a subsequent uploader, then U_B will interact with the CSP to execute the Proof of Ownership protocol (PoW). The specific process is as follows:

- The CSP selects two random seeds, r_1 and r_2, from $\mathbb{Z}_p^*$ and a random number l_1 within the range $[1, n]$;
- Then, the CSP transmits the challenge data $Chal^{(x)} = \{r_1, r_2, l_1\}$ along with K_l to user U_B;
- User U_B first computes $K_C = H_2(F)$, then uses the key K_C to recover k_l ($k_l = K_l \oplus K_C$), and subsequently encrypts file F into ciphertext c using k_l; user U_B uses the challenge information to compute random numbers $a_i = \pi_1(i, r_1), b_i = \pi_2(i, r_2)$ for each $1 \le i \le l_1$ and calculates

$$\begin{cases} p_j = \sum_{i=1}^{l_1} b_i \cdot c_{a_i, j}, & (1 \le j \le s) \\ \vartheta = \sum_{j=1}^{s} p_j \\ \Gamma = \prod_{i=1}^{l_1} H_4(c_{a_i} || a_i)^{b_i} \end{cases} \tag{3}$$

- User U_B randomly selects $x_b, s_b \in \mathbb{Z}_p$, and sets $ak_{b,f} = x_b, s_b$ and $pk_{b,f} = g_1^{H_3(k_l) \cdot s_b}$.$U_B$ then sends $Proof^{(x)} = \{\Gamma, \vartheta\}$, audit keys $ak_{b,f}$ and public key $pk_{b,f}$ to the CSP;
- The CSP also computes $a_i = \pi_1(i, r_1), b_i = \pi_2(i, r_2)$ for each $1 \le i \le l_1$ and $\Psi = \prod_{i=1}^{l_1} \tau_{a_i}^{b_i}$ and verifies if the equation (4) holds:

$$e(\Gamma \cdot \chi^{\vartheta}, pk_{b,f}) \overset{?}{=} e(\Psi, g_1^{s_b}) \tag{4}$$

- If the verification by the CSP fails, then user U_B is not a legitimate owner of the file; otherwise, the CSP computes (5) and returns $\{d_F, Hdr, C_F\}$ to U_B.

$$C_F = h^{\prod_{j=1}^{s_i}(\gamma + H_1(ID_j))(\gamma + H_1(ID_B))} \tag{5}$$

Since the CSP does not know the secret value γ, it cannot directly obtain the result of C_F through equation (5). However, it can be derived through the following steps:

$$\begin{aligned} C_F &= h^{\prod_{j=1}^{s_i}(\gamma + H_1(ID_j))(\gamma + H_1(ID_B))} \\ &= h^{(\gamma + H_1(ID_1)) \cdot (\gamma + H_1(ID_2)) \cdot \dots \cdot (\gamma + H_1(ID_B))} \\ &= h^{\gamma^{s_i}} \cdot h^{H_1(ID_1) \cdot \dots \cdot H_1(ID_B)} \\ &\quad \cdot h^{\gamma^{s_i} \sum_{j=1}^{s_i+1} H_1(ID_j) + \gamma^{(s_i - 1)} \sum_{j,t=1}^{s_i+1} H_1(ID_j) H_1(ID_t) + \dots} \end{aligned}$$

By substituting the aforementioned system public parameters, the result of C_F can be computed.

- Upon receipt, user U_B decrypts d_F: $k_F \leftarrow Dec(K_C, d_F)$, calculates $C_{F_2}^{(new)} = C_F^{k_F}$, and verifies the validity of $C_{F_2}^{(new)}$ through the following equation:

$$e\left(C_{F_2}^{(new)}, h\right) \overset{?}{=} e\left(C_{F_2}, h^{\gamma + H_1(ID_B)}\right) \tag{6}$$

  After successful verification, user U_B sets $Hdr' = (C_{F_1}, C_{F_2}^{(new)})$ and sends it to the CSP;
- In this refined approach, CSP updates the file header by setting Hdr_F to Hdr', and subsequently incorporates $H_1(ID_B)$ into the set s_F.

D. Data Integrity Auditing

CSP and the data owner U_t establish a smart contract, which defines the random file tag $\zeta_{t,f}$ of user U_t and the public key $pk_{t,f}$ of file F. Upon activation, the smart contract generates unique seeds for the indexes and coefficients corresponding to the data blocks. Then, these challenge messages are conveyed to the CSP. Upon receiving the parameters, the CSP retrieves the relevant data blocks and generates a data integrity proof, which is then transmitted back to the smart contract by the CSP. In the described scenario, the user sends a commission, referred to as deposit$_{\text{user}}$, to the smart contract, and concurrently, the CSP sends a deposit, deposit$_{\text{CSP}}$, to the same contract. If the data integrity check during the audit process is successful, the smart contract automatically transfers the user's commission, deposit$_{\text{user}}$, as an auditing fee to the miner and refunds the CSP's deposit, deposit$_{\text{CSP}}$. If the integrity check fails, the smart contract deducts the CSP's deposit, deposit$_{\text{CSP}}$, and transfers it to the miner. The detailed procedure is outlined as follows:

- The smart contract selects the nonce of the block closest to system time t, denoted as nonce(t) and the current number of blocks in the blockchain, num(t). The selected information cannot be falsified or counterfeited due to the characteristics of blockchain. The contract then computes $r_3 = H_1(t || \text{num}(t))$ and $r_4 = H_1(t || \text{nonce}(t))$. It subsequently selects a random integer l_2 from the range [1, n] and issues the challenge $Chal^{(y)} = \{r_3, r_4, l_2\}, ID_t, \zeta_{t,f}$ to the CSP;
- Upon receiving $Chal^{(y)}$, for each index i, the CSP calculates a_i using $\pi_1(i, r_3)$ and b_i using $\pi_2(i, r_4)$, and then

proceeds to compute:

$$\begin{cases} p_{a_i} = \sum_{i=1}^{l_2} x_t \cdot b_i \cdot c_{a_i,j}, \qquad (1 \le j \le s) \\ \hat{\Psi} = (\prod_{i=1}^{l_2} \tau_{a_i}^{b_i})^{s_t \cdot x_t} \\ \hat{\Gamma} = (\prod_{i=1}^{l_2} H_4(c_{a_i}||a_i)^{b_i})^{x_t} \end{cases} \tag{7}$$

The CSP then sends the computed proof $Proof^{(y)} = \{\hat{\Gamma}, \hat{\Psi}, \{p_{a_i}\}_{1 \le i \le l_2}\}$ along with $\zeta_{t,f}$ back to the smart contract;

- Upon receiving the proof, the smart contract executes Algorithm 1. The smart contract calculate $\hat{\vartheta} = \sum_{j=1}^{s} p_{a_i}$ and then verifies the validity of the proof using the equation:

$$e(\hat{\Gamma} \cdot \chi^{\hat{\vartheta}}, pk_{t,f}) \overset{?}{=} e(\hat{\Psi}, g_1) \tag{8}$$

Depending on the outcome, the contract either publishes ($\zeta_{t,f}$, integral) if the proof is valid, or ($\zeta_{t,f}$, non-integral) if the proof fails.

E. Data Retrieval

User U_y dispatches a download request denoted as $download\langle ID_y, \zeta_{y,f}\rangle$ to the CSP. Initially, the CSP validates the presence of the user within the system s_F; if the user is not recognized, the request is denied and the process terminates. Conversely, if the user is validated, the CSP locates files based on random file tags, subsequently transmitting the data tuple $\{c, ck_F, K_l, Hdr, s_F\}$ to U_y.

Upon receipt, U_y first recovers the negotiate key K_{IBBE} and utilizes this to derive the cipher key K_C via the decryption $K_C \leftarrow Dec(K_{IBBE}, ck_F)$. Following this, U_y computes the logical XOR between K_l and K_C, recovering the key $k_l = K_l \oplus K_C$, which is then used to decrypt the ciphertext c, thereby regenerating the original file $F \leftarrow Dec(k_l, c)$.

To ensure the integrity and authenticity of the retrieved file F, U_y calculates $K'_C = H_2(F)$ and confirms its equivalence to K_C, as indicated by determining whether $K'_C \overset{?}{=} K_C$ holds. The IBBE key K_{IBBE} in the system can be recovered through the following method: Assuming $s'_F = s_F - \{H_1(ID_y)\}$ and let S_F be $|s_F|$, S'_F be $|s'_F|$, then we define $p_s(\gamma) = \frac{1}{\gamma}(\prod_{j=1}^{S'_F}(\gamma + H_1(ID_j)) - \prod_{j=1}^{S'_F} H_1(ID_j))$, note that $j \ne y$. Then, K_{IBBE} can be recovered as follows:

$$\begin{aligned} &[e(C_{F_1}, h^{p_s(\gamma)}) \cdot e(sk_{ID_y}, C_{F_2})]^{\frac{1}{\prod_{j=1,j\ne y}^{S'_F} H_1(ID_j)}} \\ &= \left[e\left(\varpi^{-k_F}, h^{\frac{1}{\gamma}\left(\prod_{j=1}^{S'_F}(\gamma+H_1(ID_j))-\prod_{j=1}^{S'_F} H_1(ID_j)\right)}\right)\right. \\ &\left.\cdot e\left(g_2^{\frac{1}{\gamma+H_1(ID_y)}}, h^{k_F \prod_{j=1}^{S_F}(\gamma+H_1(ID_j))}\right)\right]^{\frac{1}{\prod_{j=1,j\ne y}^{S'_F} H_1(ID_j)}} \\ &= \left[e\left(g_2^{-\gamma k_F}, h^{\frac{1}{\gamma}\left(\prod_{j=1}^{S'_F}(\gamma+H_1(ID_j))-\prod_{j=1}^{S'_F} H_1(ID_j)\right)}\right)\right. \\ &\left.\cdot e(g_2, h)^{\frac{k_F \prod_{j=1}^{S_F}(\gamma+H_1(ID_y))}{\gamma+H_1(ID_y)}}\right]^{\frac{1}{\prod_{j=1,j\ne y}^{S'_F} H_1(ID_j)}} \\ &= \left[e(g_2, h)^{k_F \prod_{j=1}^{S'_F} H_1(ID_j)}\right]^{\frac{1}{\prod_{j=1,j\ne y}^{S'_F} H_1(ID_j)}} \\ &= e(g_2, h)^{k_F} = v^{k_F} = K_{IBBE} \end{aligned}$$

Through the aforementioned calculations, the IBBE key K_{IBBE} can be recovered.

VI. Scheme Analysis

In this section, we discuss the correctness of the proposed scheme and perform a security analysis.

A. Analysis of Correctness in Auditing

Suppose a smart contract initiates an audit challenge, dispatching the requisite challenge parameters to CSP. Upon the receipt of the audit proof from the CSP, the smart contract undertakes the verification of data integrity by assessing the validity of the equation (8). The procedure for this verification is executed by the smart contract as delineated below:

$$\begin{aligned} &e\left(\hat{\Gamma} \cdot \chi^{\hat{\vartheta}}, pk_{a,f}\right) \\ &= e\left(\prod_{i=1}^{l_2}(H_4(c_{a_i}||a_i)^{x_a} \cdot \prod_{j=1}^{s} \chi^{x_a \cdot c_{a_i,j}})^{b_i}, g_1^{H_3(k_l)\cdot s_a}\right) \\ &= e\left(\prod_{i=1}^{l_2} H_4(c_{a_i}||a_i)^{b_i} \cdot \chi^{\sum_{j=1}^{s} \cdot \sum_{i=1}^{l_2} b_i \cdot c_{a_i,j}}, g_1^{H_3(k_l)\cdot s_a \cdot x_a}\right) \\ &= e\left(\prod_{i=1}^{l_2}(((H_4(c_{a_i}||a_i) \cdot \prod_{j=1}^{s} \chi^{c_{a_i,j}})^{H_3(k_l)}))^{b_i}, g_1^{s_a \cdot x_a}\right) \\ &= e\left(\prod_{i=1}^{l_2}(((H_4(c_{a_i}||a_i) \cdot \prod_{j=1}^{s} \chi^{c_{a_i,j}})^{H_3(k_l)}))^{b_i \cdot s_a \cdot x_a}, g_1\right) \\ &= e(\hat{\Psi}, g_1) \end{aligned}$$

From the above calculations, it is evident that $e(\hat{\Gamma} \cdot \chi^{\hat{\vartheta}}, pk_{a,f}) = e(\hat{\Psi}, g_1)$. Through such calculations, the smart contract verifies the integrity of data based on the audit proof provided by the CSP, ensuring the correctness of the auditing process.

B. Security Analysis

We discuss the security aspects of the proposed scheme as follows:

Theorem 1: The proposed scheme secures the confidentiality of data and the privacy of users' data ownership.

Proof: In the proposed scheme, users encrypt files $c \leftarrow \text{Enc}(k_l, F)$, where k_l is randomly chosen from $\mathbb{Z}_p^*$ by the users. The CSP cannot derive the key k_l, thus cannot recover the plaintext F. During auditing, the plaintext data is not disclosed to the CSP, nor can the CSP deduce plaintext F during interactions with the smart contract. Furthermore, based on the information contained in the audit proofs returned by the CSP, $Proof^{(y)} = \{\hat{\Gamma}, \hat{\Psi}, \{p_{a_i}\}_{1 \le i \le l_2}\}$, where $p_{a_i} = \sum_{i=1}^{l_2} x_t \cdot b_i \cdot c_{a_i,j}$, $(1 \le$

$j \leq s)$, $\hat{\Psi} = (\prod_{i=1}^{l_2} \tau_{a_i}^{b_i})^{s_t \cdot x_t}$, $\hat{\Gamma} = (\prod_{i=1}^{l_2} H_4(c_{a_i}||a_i)^{b_i})^{x_t}$, no adversary can derive any useful information about the plaintext.

As mentioned in section IV-B, attackers could potentially infer from random file tags or audit logs whether different users possess the same file. We will next analyze how the proposed scheme prevents violations of users' data ownership privacy. We assume that the challenge information sent to the CSP by the smart contract during auditing remains unchanged; that is, when conducting audits for users t and $t^{\#}$, both send identical ciphertext c and the same questioned blocks and random coefficients to the CSP. Despite these similarities, it remains impossible to deduce from the audit outcomes that t and $t^{\#}$ hold the same file.

Assume that users t and $t^{\#}$ each possess the same file F, and independently generate random file tags $\zeta_{t,f}$ and $\zeta_{(t^{\#},f^{\#})}$ for F, it is computationally infeasible for any probabilistic polynomial-time adversary to distinguish $\zeta_{t,f}$ and $\zeta_{(t^{\#},f^{\#})}$ from random elements in $\mathbb{G}_1$, nor to associate $\zeta_{t,f}$ and $\zeta_{(t^{\#},f^{\#})}$ with the same file F. Therefore, no effective file ownership privacy information can be derived from the random file tags of users t and $t^{\#}$.

In data integrity auditing, the audit proofs generated with different users' audit keys are distinct. Users t and $t^{\#}$ each randomly select their audit keys, $\{x_t, s_t\}$ and $\{x_{(t^{\#})}, s_{(t^{\#})}\}$, respectively. Therefore, the audit proofs received by users t and $t^{\#}$, due to the differences in x_t and $x_{(t^{\#})}$, result in different p_{a_i} and $\hat{\Gamma}$, and due to the differences in both $x_t, x_{(t^{\#})}, s_t$, and $s_{(t^{\#})}$, $\hat{\Psi}$ are also different. Hence, it's impossible for all users to infer from the audit proofs $Proof^{(y,t)} = \{\hat{\Gamma}_t, \hat{\Psi}_t, \{p_{a_i}\}_{1 \leq i \leq l_2}\}$ and $Proof^{(y,t^{\#})} = \{\hat{\Gamma}_{(t^{\#})}, \hat{\Psi}_{(t^{\#})}, \{\hat{p_{a_i}}\}_{1 \leq i \leq l_2}\}$ that users t and $t^{\#}$ own the same file F. Thus, the auditing process also does not leak ownership privacy. In summary, the proposed scheme satisfies data confidentiality and protects users' ownership privacy.

Theorem 2: The proposed scheme achieves deduplication of tags.

Proof: In this paper, the authentication tags are defined as $\tau_i = [H_4(c_i||i) \cdot \prod_{j=1}^{s} \chi^{c_{i,j}}]^{H_3(k_l)}, 1 \leq i \leq n$, where $k_l \leftarrow K_l \oplus K_C$, and $K_C = H_2(F)$. Therefore, as long as subsequent uploaders are genuine owners of the file F, they can recover the file's encryption key k_l, thus ensuring that identical authentication tags can be generated. Consequently, the CSP needs to store the set of authentication tags $\{\tau_i\}_{(1 \leq i \leq n)}$ only once. Therefore, the proposed scheme has achieved the goal of eliminating duplicate authentication tags.

Theorem 3: Under the assumption that the discrete logarithm (DL) holds true, it is impossible for the CSP to parse the signing key in order to successfully meet the audit challenge.

Proof: The proposed scheme ensures the reliability of audit results. As described in section V-B, the signing key for authentication tags $\{\tau_i\}_{(1 \leq i \leq n)}$ is $\epsilon = H_3(k_l)$, where k_l is randomly chosen by user t, and CSP cannot directly know and obtain it. If CSP somehow obtained the signing key ϵ, it could generate audit proofs with forged ciphertexts and authentication tags to deceive the smart contract. We analyze how CSP could possibly obtain ϵ. According to the proposed scheme, user t sends the random file tag $\zeta_{u,f}$, the public key $pk_{t,f}$, and the audit key $ak_{t,f}$ to CSP.

For the file tag $\zeta_{t,f} = g_1^{K_C \cdot r_t}$, It is clear that the random file tags do not contain information about the signing key ϵ.

For the public key $pk_{t,f} = g_1^{H_3(k_l) \cdot s_t}$, since CSP knows the audit key $ak_{t,f} = \{x_t, s_t\}$ and has parameter g_1, it can compute $g_1^{s_t}$. Viewing $g_1^{s_t}$ as g and the public key $pk_{t,f} = g_1^{H_3(k_l) \cdot s_t}$ as g^{ϵ}, it is evident that deriving ϵ is equivalent to addressing the Discrete Logarithm (DL) problem, a task that is unachievable for any probabilistic polynomial-time adversary. Consequently, the CSP is unable to access this signing key, thus securing ϵ.

Further assuming ϵ is secure and unknown to CSP, CSP might compute a false signing key ϵ' to sign forged ciphertext c' as follows:

$$\tau'_i = \left[H_4(c'_i||i) \cdot \prod_{j=1}^{s} \chi^{c'_{i,j}} \right]^{\epsilon'}, 1 \leq i \leq n$$

CSP then generates an audit proof $Proof' = \{\hat{\Gamma}', \hat{\Psi}', \{p'_{a_i}\}_{1 \leq i \leq l_2}\}$ for this forged ciphertext and authentication tags:

$$\begin{cases} p'_{a_i} = \sum_{i=1}^{l_2} x_t \cdot b_i \cdot c'_{a_i,j}, & (1 \leq j \leq s) \\ \hat{\Psi}' = \left(\prod_{i=1}^{l_2} \tau'^{b_i}_{a_i} \right)^{x_u \cdot s_t} \\ \hat{\Gamma}' = \left(\prod_{i=1}^{l_2} H_4(c'_{a_i}||a_i)^{b_i} \right)^{x_t} \end{cases}$$

The smart contract verifies the audit proof $Proof'$ through the following equations:

$$\begin{aligned} & e\left(\hat{\Gamma}' \cdot \chi^{\hat{\vartheta}'}, pk_{u,f}\right) \\ & = e\left(\prod_{i=1}^{l_2} H_4(c'_{a_i}||a_i)^{b_i} \cdot \chi^{\sum_{j=1}^{s} \cdot \sum_{i=1}^{l_2} b_i \cdot c'_{a_i,j}}, g_1^{\epsilon \cdot s_u \cdot x_t} \right) \\ & \neq e\left(\prod_{i=1}^{l_2} H_4(c'_{a_i}||a_i)^{b_i} \cdot \chi^{\sum_{j=1}^{s} \cdot \sum_{i=1}^{l_2} b_i \cdot c'_{a_i,j}}, g_1^{\epsilon' \cdot s_t \cdot x_t} \right) \\ & = e(\hat{\Psi}', g_1) \end{aligned} \quad (9)$$

From the aforementioned equation (9), it is known that since $\epsilon \neq \epsilon'$, therefore $e(\hat{\Gamma}' \cdot \chi^{\hat{\vartheta}'}, pk_{t,f}) \neq e(\hat{\Psi}', g_1)$. As a result, the CSP cannot generate a corresponding audit proof to deceive the smart contract by forging ciphertext and authentication tags. Therefore, the proposed scheme ensures the reliability of the audit results. One more point, when CSP verifies the uploader's file ownership in the subsequent upload stage, the user cannot pass the challenge by forging the ciphertext. Specifically, when a subsequent uploader uploads a file, the CSP verifies its file ownership by the equation (4), calculated as follows.

$$\begin{aligned} & e\left(\Gamma \cdot \chi^{\vartheta}, pk_{b,f}\right) \\ & = e\left(\prod_{i=1}^{l_1} H_4(c_{a_i}||a_i)^{b_i} \cdot \chi^{\sum_{j=1}^{s} \cdot \sum_{i=1}^{l_1} b_i \cdot c_{a_i,j}}, g_1^{\epsilon \cdot s_b} \right) \\ & \neq e\left(\prod_{i=1}^{l_1} H_4(c'_{a_i}||a_i)^{b_i} \cdot \chi^{\sum_{j=1}^{s} \cdot \sum_{i=1}^{l_1} b_i \cdot c'_{a_i,j}}, g_1^{\epsilon \cdot s_b} \right) \\ & = e(\Psi, g_1^{s_b}) \end{aligned} \quad (10)$$

TABLE II
COMPARISON OF FUNCTIONALITIES ACROSS DIFFERENT SOLUTIONS

Scheme	Data confidentiality	Authentication tag deduplication	User offline	Key management	Protection of ownership privacy	Audit without TPA
Xu et al.' scheme [7]	No	Yes	Yes	No	No	No
Tian et al.' scheme [6]	Yes	Yes	No	No	No	No
Yuan et al.' scheme [5]	Yes	No	No	No	No	Yes
Our proposed scheme	Yes	Yes	Yes	Yes	Yes	Yes

TABLE III
COMPARATIVE ANALYSIS OF COMPUTATIONAL COSTS FOR USERS IN THE DATA UPLOADING PROCESS

Scheme	All uploaders		Initial uploader	Subsequent uploader
	key generation	file tag generation		
Xu et al.' scheme [7]	—	t_h	$n(2t_h + t_g + 2t_x)$	$l_1(2t_r + t_a + t_z + t_g + t_x)$
Tian et al.' scheme [6]	nt_h	$t_h + t_x$	$2n(t_h + t_x) + n(t_g + 2t_c)$	$l_1(2t_r + 2t_c + t_h + t_z + t_g + t_x + t_a)$
Yuan et al.' scheme [5]	$nt_h + t_x + t_b$	t_h	$n(2t_h + t_g + t_c + 2t_x)$	$n(2t_h + t_g + t_c + 2t_x)$
Our proposed scheme	t_h	$t_x + t_g$	$n(t_h + t_g + 2t_x + st_a) + 3t_x + t_a + t_g + 3t_c$	$l_1(2t_r + st_a + t_g + t_h + t_x) + st_z + t_h + 3t_c + t_x + t_b$

TABLE IV
COMPARISON OF COMPUTATIONAL OVERHEADS DURING THE AUDITING PHASE

Scheme	User	CSP
Xu et al.' scheme [7]	l_2t_r	$l_2(2t_r + 2t_x + t_a + t_g + t_z)$
Tian et al.' scheme [6]	$nt_g + (n+1)t_x$	$l_2(2t_r + t_h + t_z + t_a)$
Yuan et al.' scheme [5]	l_2t_r	$l_2(2t_r + t_z + t_a + t_x + t_g) + t_h + h_m$
Our proposed scheme	—	$l_2(2t_r + 2t_x + t_a + t_z + 2t_g + t_h) + 2t_z + 2t_x$

It is also easy to see that only when the ciphertext is the same, the user can pass the ownership verification. It can be seen that the reliability of ownership verification is satisfied. Finally, the proposed scheme also guarantees that the file F recovered by the user from the CSP is correct, as explained in section V-E.

VII. THEORETICAL ANALYSIS AND PERFORMANCE ANALYSIS

A. Functionality and Theoretical Analysis

In this section, we conduct a functional comparison between the proposed scheme and schemes [5], [6] and [7]. Additionally, we analyzed the theoretical performance of the proposed scheme and the comparative schemes in terms of computational cost, communication cost, and storage cost.

Functionality: The proposed scheme encompasses the following functionalities: users encrypt their data before uploading it to the cloud, and an auditing functionality is implemented to ensure the integrity of the data stored in the cloud, the CSP performs duplicate data checks on outsourced data to avoid redundant storage. After the initial uploader uploads the data, subsequent uploaders possessing the same file data are challenged by the cloud for ownership verification upon attempting to upload this file, thereby achieving deduplication of encrypted data and authentication tags. Moreover, the proposed scheme can always protect the privacy of user file ownership. A functional comparison between the proposed scheme and other schemes is shown in Table II.

Computation cost: Assume t_x, t_r, t_h, t_z, t_g, t_a, t_b, t_c respectively denote the computational overhead of exponentiation in $\mathbb{G}_1$, generating pseudorandom numbers, hashing operations, multiplication operations in $\mathbb{Z}_p$, multiplication operations in $\mathbb{G}_1$, addition operations in $\mathbb{Z}_p$, bilinear mapping operations, and AES-256 operations. The comparison results are as illustrated in the Tables III and IV. During the key generation phase, the initial uploader in the proposed scheme randomly selects a key k_l, then computes $K_C = H_2(F)$, and finally performs an XOR operation to obtain K_l. Subsequent uploaders can use their private keys to recover the negotiated key K_{IBBE}, which then allows them to further recover the key k_l. In Xu et al.'s scheme [7], due to the execution of plaintext deduplication, no encryption operation is performed on the files. Consequently, users incur zero overhead during the key generation phase. The computational overhead in Yuan et al.'s scheme [5] is $nt_h + t_x + t_b$, in Tian et al.'s scheme [6] it is nt_h, and in the proposed scheme, it is t_h. In the generation of file tags, the proposed scheme utilizes K_C and a random number r to compute the file tag $tag_{t,f} = g_1^{K_C}$ and random file tag $\zeta_{t,f} = g_1^{K_C \cdot r}$,

TABLE V
COMPARISONS OF USER COMMUNICATION OVERHEAD

Scheme	Initial Upload With CSP	Subsequent Upload With CSP	Data Retrieval From CSP
Xu et al.' scheme [7]	$\|M\| + n\|\mathbb{G}_1\| + \|\mathbb{Z}_p\|$	$4\|\mathbb{Z}_p\|$	$\|M\|$
Tian et al.' scheme [6]	$\|C\| + (n+1)\|\mathbb{G}_1\| + n\|\mathbb{Z}_p\|$	$\|\mathbb{G}_1\| + (2n+2)\|\mathbb{Z}_p\|$	$\|C\| + n\|\mathbb{Z}_p\|$
Yuan et al.' scheme [5]	$\|C\| + n\|\mathbb{G}_1\| + 2\|\mathbb{Z}_p\|$	$\|C\| + n\|\mathbb{G}_1\| + 2\|\mathbb{Z}_p\|$	$\|C\|$
Our proposed scheme	$\|C\| + (n+7)\|\mathbb{G}_1\| + 3\|\mathbb{Z}_p\|$	$8\|\mathbb{G}_1\| + 6\|\mathbb{Z}_p\|$	$\|C\| + 4\|\mathbb{G}_1\|$

TABLE VI
COMPARISON OF STORAGE OVERHEADS

Scheme	User	CSP	Blockchain
Xu et al.' scheme [7]	$F\|\mathbb{Z}_p\|$	$\|M\| + n\|\mathbb{G}_1\| + \|\mathbb{Z}_p\|$	$\|\mathbb{G}_1\| + 6\|\mathbb{Z}_p\|$
Tian et al.' scheme [6]	$(F+1)\|\mathbb{Z}_p\| + F\|\mathbb{G}_1\|$	$(n+1)(\|\mathbb{G}_1\| + \|\mathbb{Z}_p\|) + \|C\|$	$\|\mathbb{G}_1\| + 5\|\mathbb{Z}_p\|$
Yuan et al.' scheme [5]	$(2+F_n)\|\mathbb{Z}_p\|$	$U(n+1)\|\mathbb{G}_1\| + \|C\|$	$5\|\mathbb{G}_1\| + \|\mathbb{Z}_p\|$
Our proposed scheme	$(F+1)\|\mathbb{G}_1\|$	$(n+7)\|\mathbb{G}_1\| + 3U\|\mathbb{Z}_p\| + \|C\|$	$\|3\mathbb{G}_1\| + 3\|\mathbb{Z}_p\|$

with a computational overhead of $t_x + t_g$. The computational overhead in scheme [7] is t_h, in scheme [5] is t_h, and in scheme [6], it is $t_h + t_x$.

During the upload phase, if the uploader is the initial uploader, the primary overhead comes from encrypting all data blocks and signing all ciphertext blocks, resulting in a computational overhead of $n(2t_h + t_g + 2t_x + st_a) + 3t_x + t_a + t_g + 3t_c$ in the proposed scheme. In scheme [7], the computation overhead is $n(2t_h + t_g + 2t_x)$, in scheme [5], the computational overhead is $n(2t_h + t_g + t_c + 2t_x)$, and in scheme [6], it is $2n(t_h + t_x) + n(t_g + 2t_c)$.

If the uploader is a subsequent uploader, the main overhead in the proposed scheme comes from computing the ownership proof and updating the broadcast encryption header. The computational overhead of the proposed scheme is $l_1(2t_r + st_a + t_g + t_h + t_x) + st_z + t_h + 3t_c + t_x + t_b$. In Scheme [7], the computational overhead is $l_1(2t_r + t_a + t_z + t_g + t_x)$, in scheme [5], the computational overhead during this stage is the same as that for the initial uploader, $n(t_h + t_g + t_c + 2t_x)$, and in scheme [6], it is $l_1(2t_r + 2t_c + t_h + t_z + t_g + t_x + t_a)$.

During the auditing phase, the proposed scheme does not require user participation, thus incurring no computational overhead for the user. CSP's computational overhead is $l_2(2t_r + 2t_x + t_a + t_z + 2t_g + t_h) + 2t_z + 2t_x$. In scheme [7], the user's computational overhead is l_2t_r, the CSP's computational overhead is $l_2(2t_r + 2t_x + t_a + t_g + t_z)$. In scheme [5], the user's computational overhead is l_2t_r, the CSP's computational overhead is $l_2(2t_r + t_z + t_a + t_x + t_g) + t_h + h_m$. In scheme [6], the user's computational overhead is $nt_g + (n+1)t_x$, the CSP's computational overhead is relatively low, which is $l_2(2t_r + t_h + t_z + t_a)$.

Communication cost: The communication overhead comparison of different schemes is as follows Table V. Since the proposed scheme does not require interaction with the key server during the key generation phase, there is no communication overhead at this stage, in the comparison schemes [5] and [6], users interact with the key server to generate convergent keys for each of the n blocks that the file F is divided into, resulting in a communication overhead of nZ_p, and in the comparison scheme [7] involves plaintext data uploads without encrypting the data, and thus does not incur any communication overhead with the key server. In the initial upload phase, users will upload a ciphertext of length $|C|$, an authentication tag of $(n|\mathbb{G}_1|)$ bits, a file tag of $|\mathbb{G}_1|$ bits, a random file tag of $|\mathbb{G}_1|$ bits, a public key of $|\mathbb{G}_1|$ bits, an audit key of $(2|\mathbb{Z}_p|)$ bits, a key K_l of $|\mathbb{G}_1|$ bits, an Hdr_F header of $(2|\mathbb{G}_1|)$ bits, a key d_F of $|\mathbb{Z}_p|$ bits, and a key ck_F of $|\mathbb{G}_1|$ bits. For subsequent uploaders, users need to receive a challenge message of $(2|\mathbb{Z}_p|)$ bits, a key K_l of $|\mathbb{G}_1|$ bits, and an Hdr_F header of $|\mathbb{G}_1|$ bits; upload a file tag of $|\mathbb{G}_1|$ bits, a random file tag of $|\mathbb{G}_1|$ bits, a file public key of $|\mathbb{G}_1|$ bits, an audit key of $(2|\mathbb{Z}_p|)$ bits, a file ownership proof of $(|\mathbb{G}_1| + 2|\mathbb{Z}_p|)$ bits, and an Hdr_F header of $(2|\mathbb{G}_1|)$ bits. In the data recovery phase, users need to download from the cloud a ciphertext of length $|C|$, and $4|\mathbb{G}_1|$ bits of other necessary data, ck_F, the key K_l, and the Hdr_F header.

Storage cost: Let F represent the number of files and U denote the number of users owning the same file. In scheme [5], since all users generate their own identity authenticators and need to store them, the storage overhead is related to both the number of users and the number of blocks. On the CSP side, the storage overhead is $U(n+1)|\mathbb{G}_1| + |C|$. In the proposed scheme, the authenticator is related to the file, not to the user's private keys, making the storage cost of the proposed scheme $(n+7)|\mathbb{G}_1| + 3U|\mathbb{Z}_p| + |C|$. On the user side, in scheme [7] users do not encrypt the file F, consequently, there is no overhead for storing encryption keys. The only storage requirement is for the file tag t, which has a size of $F|\mathbb{Z}_p|$. Scheme [5] requires storing keys (x, ssk) and F_n convergent keys, with a storage cost of $(2 + F_n)|\mathbb{Z}_p|$, scheme [6] requires storing t, t^*, sk, with a storage cost of $(F+1)|\mathbb{Z}_p| + F|\mathbb{G}_1|$. In the proposed scheme, users need to store their private key and the file tag ζ, resulting in a storage cost of $(F+1)|\mathbb{G}_1|$.

B. Performance Analysis

We establish a series of experiments to perform performance evaluations, testing the on-chain and off-chain computational costs associated with all comparison schemes. We implement the proposed scheme and the comparison schemes using Python

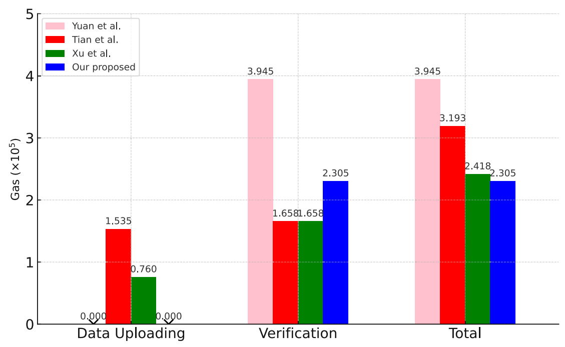


Fig. 4. Comparisons of on-chain operation cost.

TABLE VII
SUMMARY OF EXPERIMENTAL PARAMETERS

Parameter	Description	Value
Data size	Total size of file data	512 KB∼10 MB
Block size	Size of each data block	1 KB
Sector size	Size of each sector in a block	128 B
Algorithm	Hashing algorithm used	SHA-256
Hardware environment	Experimental device configuration	i7-12700 CPU (2.10 GHz), 8 GB RAM

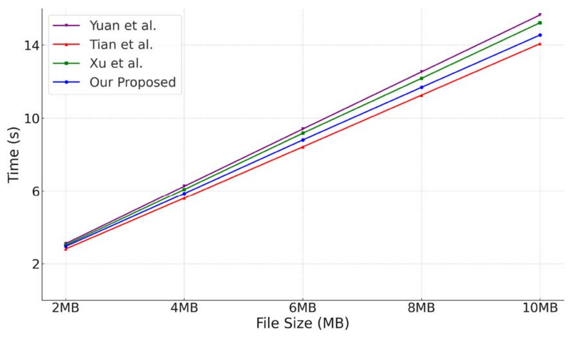


Fig. 5. Initial upload.

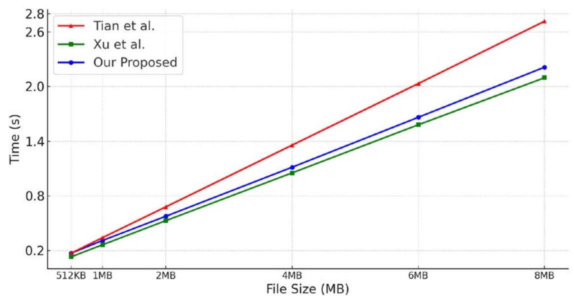


Fig. 6. Subsequent upload.

by invoking the charm-crypto Library[1], which relies on the GNU Multiple Precision Arithmetic Library (GMP)[2] and the Pairing-Based Cryptography Library (PBC)[3], and we select the BN254 curve for performing pairing operations. The hash function used to generate the encryption key is SHA-256, and the symmetric encryption algorithm is AES-256. We test the gas cost of on-chain operations by deploying a smart contract using Solidity. The computational cost for users is assessed on a desktop computer equipped with an i7-12700 CPU (2.10 GHz) and 8GB of memory, running the Ubuntu operating system. In the simulation, we configure the number of challenged blocks in the Proof of Ownership (PoW) challenge and the audit challenge to be 1/3 of the total number of blocks, with each block size set to 1KB and sector size set to 128B. All experiments are conducted 50 times to obtain the average computational results.

1) On-Chain Part Costs: We evaluate the on-chain computational cost, i.e., gas cost, for all schemes when uploading a 5MB file. The on-chain gas costs for all schemes are shown in the Fig. 4.

Scheme [6] and scheme [7] require the creation of an index table on the blockchain during the data upload phase, but the proposed scheme needn't create it, resulting in no gas cost at this stage. In the audit proof verification phase, we simulated the comparison schemes and the proposed scheme verifying audit proofs on-chain, among the all schemes, the gas overhead of the scheme [5] is the highest. This is attributed to the fact that their scheme performs more computations on the blockchain. Ultimately, in the total gas expenditures for all schemes, scheme [5] had the highest costs, while the proposed scheme was marginally lower than comparison schemes.

2) Off-Chain Part Costs: We further analyze the off-chain computational costs of all schemes, including initial data upload, subsequent data upload and generating audit proofs.

During the data upload phase, all schemes are divided into two parts: initial upload and subsequent upload, and the computational costs are shown in Fig. 5 and Fig. 6, respectively. During the initial upload phase, all schemes take over 14 seconds for files of size 10MB. Our scheme ranks second in terms of overhead because we optimized the computational cost of the key generation process. In our scheme, we randomly select k_l to encrypt the file without requiring the computation of the message-locked encryption key generation algorithm. However, our scheme incurs additional computational overhead due to the identity-based broadcast encryption performed. The total time for initial upload includes key generation, encrypting file F into ciphertext, generating file tags and the authentication tags. In the proposed scheme, during the initial upload, the user randomly selects a key k_l to encrypt file F and all other legitimate owners of the file can use their private keys to recover the key k_l, the overhead of key generation in the proposed scheme is lower than that of the comparison schemes. From the experimental results, it can be observed that during initial upload phase the computational overhead of our proposed scheme is slightly higher than that of scheme [7], while remaining lower than the computational overheads of both scheme [5] and scheme [6].

As scheme [5] executes the same steps during subsequent uploads as in the initial upload phase, it undoubtedly incurs the highest computational overhead. Therefore, we only compare the computational overhead of the proposed scheme with those

[1]https://github.com/JHUISI/charm.
[2]https://gmplib.org/.
[3]https://crypto.stanford.edu/pbc/download.html.

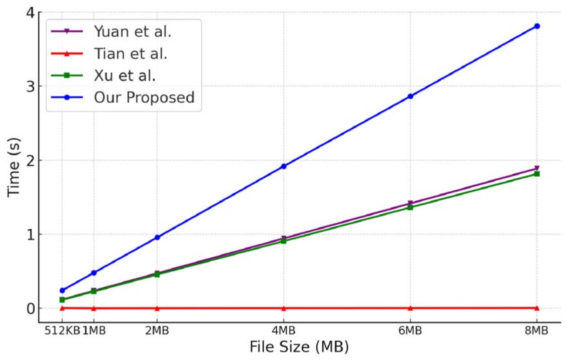


Fig. 7. Proof generation in data auditing.

of scheme [7] and scheme [6] in this phase. In the subsequent upload phase, the user generates ownership proofs and sends them to the CSP, the computational costs of the proposed scheme are slightly higher than those of scheme [7] but lower than those of scheme [6]. The computational overhead mainly arises from ownership verification challenges. In scheme [6], the computation of ownership proofs returned by the user is more complex and our proposed scheme requires additional calculations for broadcast encryption, resulting in a second-place ranking in terms of computational overhead. For a file size of 8MB, our scheme's computational overhead is approximately 2.2 seconds, compared to 2.1 seconds and 2.7 seconds for other schemes. During the data integrity audit, the proposed scheme involves more computations when generating proofs, resulting in higher computational costs compared to the comparison schemes, as shown in Fig. 7. In terms of audit proof generation overhead, our scheme takes approximately 3.7 seconds for a file size of 8MB, compared to around 1.7 seconds and 1.8 seconds for the other schemes. Scheme [6] minimizes computational overhead during the proof generation phase by offloading complex computations, such as proof aggregation, to the proof verification phase.

VIII. Conclusion

In this study, a blockchain-based data integrity auditing and deduplication scheme is proposed. The proposed scheme supports the simultaneous deduplication of duplicate files and their authentication tags. Privacy protection measures are implemented for file tags and audit proofs, to safeguard user file ownership privacy. Additionally, the proposed scheme leverages identity-based broadcast encryption technology to eliminate the need for user interaction with the key server, thereby reducing the number of keys that users need to manage, which lowers key management overhead. Moreover, smart contracts are used for data integrity auditing to avoid introducing a fully trusted TPA. In summary, the proposed scheme improves the practicality and efficiency of cloud-storage services.

Acknowledgment

The authors are very grateful to the anonymous referees for their detailed comments and suggestions regarding this paper.

References

[1] Computerworld, "By 2020, there will be 5,200 GB of data for every person on earth," 2012. Accessed: Jun. 20, 2024. [Online]. Available: https://www.computerworld.com/article/2493701/by-2020--there-will-be-5-200-gb-of-data-for-every-person-on-earth.html
[2] International Data Corporation (IDC), "Worldwide public cloud infrastructure as a service forecast, 2023–2027," 2023. Accessed: Jun. 20, 2024. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US51019723
[3] CyberTalk, "Top 5 cloud security breaches and lessons," 2022. Accessed: Jun. 20, 2024. [Online]. Available: https://www.cybertalk.org/2022/04/26/top-5-cloud-security-breaches-and-lessons/
[4] J. Gratz and D. Reinsel, "The digital universe decade-are you ready?" IDC White Paper, 2010.
[5] H. Yuan, X. Chen, J. Wang, J. Yuan, H. Yan, and W. Susilo, "Blockchain-based public auditing and secure deduplication with fair arbitration," *Inf. Sci.*, vol. 541, pp. 409–425, 2020.
[6] G. Tian et al., "Blockchain-based secure deduplication and shared auditing in decentralized storage," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 6, pp. 3941–3954, Nov./Dec. 2022.
[7] Y. Xu, C. Zhang, G. Wang, Z. Qin, and Q. Zeng, "A blockchain-enabled deduplicatable data auditing mechanism for network storage services," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1421–1432, Jul./Sep. 2021.
[8] Y. Miao, K. Gai, L. Zhu, K.-K. R. Choo, and J. Vaidya, "Blockchain-based shared data integrity auditing and deduplication," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 4, pp. 3688–3703, Jul./Aug. 2024.
[9] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1615–1625, Jun. 2014.
[10] Y. Zheng, X. Yuan, X. Wang, J. Jiang, C. Wang, and X. Gui, "Toward encrypted cloud media center with secure deduplication," *IEEE Trans. Multimedia*, vol. 19, no. 2, pp. 251–265, Feb. 2017.
[11] J. Yuan and S. Yu, "Secure and constant cost public cloud storage auditing with deduplication," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Piscataway, NJ, USA: IEEE, 2013, pp. 145–153.
[12] J. Li, J. Li, D. Xie, and Z. Cai, "Secure auditing and deduplicating data in cloud," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2386–2396, Aug. 2016.
[13] X. Liu, W. Sun, W. Lou, Q. Pei, and Y. Zhang, "One-tag checker: Message-locked integrity auditing on encrypted cloud deduplication storage," in *Proc. IEEE INFOCOM–IEEE Conf. Comput. Commun.*, Piscataway, NJ, USA: IEEE, 2017, pp. 1–9.
[14] X. Gao et al., "Achieving low-entropy secure cloud data auditing with file and authenticator deduplication," *Inf. Sci.*, vol. 546, pp. 177–191, 2021.
[15] Y. Deswarte, J.-J. Quisquater, and A. Saïdane, "Remote integrity checking: How to trust files stored on untrusted servers," in *Proc. Integrity Internal Control Inf. Syst. VI: IFIP TC11/WG11. 56th Work. Conf. Integrity Internal Control Inf. Syst. (IICIS), Nov. 13–14, 2003, Lausanne, Switzerland*. Boston, MA, USA: Springer, 2004, pp. 1–11.
[16] G. Ateniese et al., "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 598–609.
[17] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 847–859, May 2011.
[18] X. Yang, X. Pei, M. Wang, T. Li, and C. Wang, "Multi-replica and multi-cloud data public audit scheme based on blockchain," *IEEE Access*, vol. 8, pp. 144809–144822, 2020.
[19] J. Xue, C. Xu, J. Zhao, and J. Ma, "Identity-based public auditing for cloud storage systems against malicious auditors via blockchain," *Sci. China Inf. Sci.*, vol. 62, pp. 1–16, 2019.
[20] Q. Zhang, D. Sui, J. Cui, C. Gu, and H. Zhong, "Efficient integrity auditing mechanism with secure deduplication for blockchain storage," *IEEE Trans. Comput.*, vol. 72, no. 8, pp. 2365–2376, Aug. 2023.
[21] Q. Zhang et al., "Efficient blockchain-based data integrity auditing for multi-copy in decentralized storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 12, pp. 3162–3173, Dec. 2023.
[22] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.,* 2008, pp. 90–107.
[23] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.,* Berlin, Heidelberg: Springer, 2001, pp. 514–532.

[24] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proc. 22nd Int. Conf. Distrib. Comput. Syst.*, Piscataway, NJ, USA: IEEE, 2002, pp. 617–624.

[25] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, Berlin, Heidelberg: Springer, 2013, pp. 296–312.

[26] R. Chen, Y. Mu, G. Yang, and F. Guo, "BL-MLE: Block-level message-locked encryption for secure large file deduplication," *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 12, pp. 2643–2652, Dec. 2015.

[27] Y. Zhao and S. S. Chow, "Updatable block-level message-locked encryption," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 449–460.

[28] Y. Zhang, C. Xu, N. Cheng, and X. Shen, "Secure password-protected encryption key for deduplicated cloud storage systems," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 4, pp. 2789–2806, Jul./Aug. 2022.

[29] C. Delerablée, "Identity-based broadcast encryption with constant size ciphertexts and private keys," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Berlin, Heidelberg: Springer, 2007, pp. 200–215.

[30] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Bitcoin*, vol. 4, no. 2, p. 15, 2008.

[31] G. Wood et al., "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[32] 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 Trans. Services Comput.*, vol. 14, no. 5, pp. 1292–1305, Sep./Oct. 2021.

Qingyang Zhang (Member, IEEE) was born in Anhui Province, China, in 1992. He received the B.Eng. and Ph.D. degrees in computer science from Anhui University, in 2021. Currently, he is an Associate Professor with the School of Computer Science and Technology, Anhui University. His research interests include edge computing, computer systems, and security. He has over 30 scientific publications in reputable journals (e.g., PROCEEDINGS OF THE IEEE, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and IEEE TRANSACTIONS ON COMPUTERS) and international conferences.

Shuai Qian is currently a Research Student with the School of Computer Science and Technology, Anhui University. His research focuses on the deduplication and integrity auditing for cloud storage.

Jie Cui (Senior Member, IEEE) was born in Henan Province, China, in 1980. He received the Ph.D. degree from the University of Science and Technology of China, in 2012. Currently, he is a Professor and the Ph.D. Supervisor with the School of Computer Science and Technology, Anhui University. His research interests include applied cryptography, Internet of Things (IoT) security, vehicular ad hoc network, cloud computing security, and software-defined networking (SDN). He has over 150 scientific publications in reputable journals (e.g., IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and IEEE TRANSACTIONS ON COMPUTERS), academic books, and international conferences.

Hong Zhong (Member, IEEE) was born in Anhui Province, China, in 1965. She received the Ph.D. degree in computer science from the University of Science and Technology of China, in 2005. Currently, she is a Professor and the Ph.D. Supervisor with the School of Computer Science and Technology, Anhui University. Her research interests include applied cryptography, IoT security, vehicular ad hoc networks, cloud computing security, and software-defined networking (SDN). She has over 200 scientific publications in reputable journals (e.g., IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE TRANSACTIONS ON MULTIMEDIA, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and IEEE TRANSACTIONS ON CLOUD COMPUTING), academic books, and international conferences.

Fengqun Wang is currently working toward the Ph.D. degree with the School of Computer Science and Technology, Anhui University, Hefei, China. His research interests include IoT security, blockchain, and applied cryptography.

Debiao He (Member, IEEE) received the Ph.D. degree in applied mathematics from the School of Mathematics and Statistics, Wuhan University, Wuhan, China, in 2009. Currently, he is a Professor with the School of Cyber Science and Engineering, Wuhan University, Wuhan, China, and Shanghai Key Laboratory of Privacy Preserving Computation, MatrixElements Technologies, Shanghai, China. His research interests include cryptography and information security, in particular, cryptographic protocols. He has published over 100 research papers in refereed international journals and conferences, such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and the Usenix Security Symposium. He was the recipient of the 2018 IEEE Systems Journal Best Paper Award and the 2019 IET Information Security Best Paper Award. He is on the Editorial Board of several international journals, such as *Journal of Information Security and Applications* and *Human-centric Computing and Information Sciences*.