Blockchain-Assisted Secure Deduplication for Large-Scale Cloud Storage Service

Zhongyun Hua, *Senior Member, IEEE*, Yufei Yao, Mingyang Song, Yifeng Zheng, Yushu Zhang, and Cong Wang, *Fellow, IEEE*

***Abstract*—Secure deduplication over encrypted data can greatly improve cloud storage efficiency and protect data privacy. Recently, there have been some research efforts aiming at designing secure deduplication schemes with the assistance of key servers (KSs). However, prior works are unsatisfactory in that they suffer from some limitations such as security degradation (the leakage at partial KSs will lead to all the ciphertexts being subject to offline brute-force attacks) or lack of scalability for handling the change of KSs. In this article, we propose a new secure deduplication scheme for large-scale cloud storage service, which, to our best knowledge, is the first server-aided scheme that supports both tolerance of partial KSs leakage and dynamic change of KSs. Our scheme divides all the KSs into multiple groups and each KS group keeps a randomly generated secret key using threshold cryptography. We design a file-related KS group selection mechanism for assisting encryption key generation, which guarantees that the identical files of different users can be encrypted using the same keys. Our scheme is designed to update the KS groups regularly for supporting the joining and leaving of the KSs as well as maintaining long-term security. We leverage the blockchain to help divide KSs into groups in a fair way and securely migrate group secret keys during KS group updating. Formal analysis is provided to verify the correctness of our scheme and justify its security, and both theoretical and experimental results demonstrate that it has modest performance overhead.**

***Index Terms*—Cloud storage, secure deduplication, server-aided encryption, key servers management.**

Manuscript received 7 April 2023; revised 20 December 2023; accepted 24 December 2023. Date of publication 5 January 2024; date of current version 12 June 2024. 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 2023A1515010714, in part by the Shenzhen Science and Technology Program under Grants JCYJ20220531095416037, JCYJ20230807094411024, RCBS20210609103056041, and ZDSYS20210623091809029, in part by the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies under Grant 2022B1212010005, and in part by the Research Grants Council of Hong Kong under Grants CityU 11217620, 11218521, 11218322, R6021-20F, R1012-21, RFS2122-1S04, C2004-21G, C1029-22G, and N_CityU139/21. *(Corresponding author: Yifeng Zheng.)*

Zhongyun Hua is with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China, and also with the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Shenzhen, Guangdong 518055, China (e-mail: huazhongyun@hit.edu.cn).

Yufei Yao, Mingyang Song, and Yifeng Zheng are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China (e-mail: yaoyufei0615@hotmail.com; songmingyang2022@gmail.com; yifeng.zheng@hit.edu.cn).

Yushu Zhang is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China (e-mail: yushu@nuaa.edu.cn).

Cong Wang is with the Department of Computer Science, City University of Hong Kong, Kowloon Tong, Hong Kong (e-mail: congwang@cityu.edu.hk).

Digital Object Identifier 10.1109/TSC.2024.3350086

I. Introduction

In THE era of Big Data, more and more digital data are generated every moment by all kinds of digital devices such as smartphones and various sensors. Cloud storage can provide low-cost and convenient data storage services, which have attracted many enterprises and individuals to store their data on the cloud [1]. However, almost 75% of data stored on the cloud are duplicated [2], and thus data deduplication can greatly improve cloud storage efficiency by storing only one copy of the duplicated data [3], [4], [5]. Most commercial cloud service providers (CSPs) support data deduplication (e.g., Google Drive [6] and Dropbox [7]). However, coming with the popularity of cloud storage services are acute data privacy concerns. So users may encrypt their data for protection before outsourcing the storage. This requires CSPs to support deduplication over encrypted data. Traditional symmetric encryption algorithms cannot allow secure cross-user deduplication, because each user holds an encryption key and different users encrypt identical files into different ciphertexts [8]. Convergent encryption (CE) technique [9] can well solve this issue by deriving the encryption key from the data itself. As a result, different users can encrypt identical files into exactly the same ciphertexts such that CSP can perform deduplication over ciphertexts. This kind of encryption is then formalized as a new cryptographic primitive called message-locked encryption (MLE) [10] and has been used in many secure deduplication schemes [11], [12], [13], [14], [15], [16].

Due to the convergence of ciphertexts, MLE, however, is susceptible to offline brute-force attacks (BFA) and DupLESS [17] is the first scheme that can defend against such attacks. A third-party key server (KS) holding a secret key is deployed in DupLESS to help user generate the encryption key. The users execute blind signature-based protocol [18] with the KS to obtain server-aided convergent keys. However, using KS would inevitably lead to trust issues, as a third-party KS may be compromised. Besides, deploying one KS may raise the single-point-of-failure issue to the system. To improve privacy and reliability, the work in [19] proposed a new secure deduplication scheme by deploying a fixed group of KSs that share a secret key through threshold cryptography [20].

However, these works are inappropriate for large-scale cloud storage systems. To ensure a fast response, a large-scale cloud storage system should deploy many KSs to handle the vast system requests simultaneously. The works in [21], [22], [23]

proposed some such secure deduplication schemes for large-scale cloud storage. To balance the trade-off between efficiency and security, the schemes in [21], [22] divided all the available KSs into different KS groups and designed special ciphertext structures for supporting cross-user deduplication. The scheme in [23] treated all the KSs as one group and designed a proactivization mechanism to support addition and removal KSs in the system. However, these schemes still suffer from the following limitations: 1) The scheme in [21] supports only fixed-KS settings, which lacks scalability to deploy more KSs for handling the probably increasing storage requirements; 2) The scheme in [22] may incur the single-point-of-failure issue because each KS group has only one KS in the design; 3) For the schemes in [21], [22], [23], the leakage of the secret key in one KS group will degrade the security level of the whole system to that of MLE. Then all the ciphertexts stored on the cloud are subject to offline BFA; 4) The schemes in [21], [22] cannot sustain long-term security, because their KS group settings are fixed and a sophisticated adversary may corrupt some fixed KSs with long enough time [24].

Considering the demands of practical application, a server-aided secure deduplication scheme should have the following properties to support large-scale cloud storage. (1) It should achieve cross-user deduplication, which means that the identical files of different users can be detected as duplicated files by the cloud. (2) It should protect users' data with a high ability to resist commonly occurring security risks. (3) It should have the scalability to deal with the dynamic change of KSs. This is because more KSs should be deployed with the increase of storage requests and some KSs may be offline because of some internal or external reasons such as hardware failures. (4) The leakage at some KS groups affects only partial ciphertexts, rather than all the ciphertexts stored on the cloud. (5) It should provide long-term security for the ciphertexts stored on the cloud. To our best knowledge, there is not yet a server-aided secure deduplication scheme that simultaneously considers and addresses these issues.

In this paper, we propose a new secure deduplication scheme for large-scale cloud storage, aiming to address the above issues. Our scheme divides all the KSs into multiple groups and each KS group keeps a randomly generated secret key using threshold cryptography. We develop a file-related group selection mechanism that uses the file itself to determine the KS group for assisting encryption key and tag generation. Then the identical files of different users can be encrypted using the same encryption keys so that the CSP can perform ciphertext deduplication. To support the dynamic change of KSs and maintain long-term security, our scheme updates the KS group regularly. We introduce the blockchain into our system to design a fair KS grouping mechanism and use the secure migration mechanism introduced in [23] to migrate the secret keys between the old and new KS groups during KS group updating. Then the identical files uploaded before and after KS group updating can be detected as duplicated files. Considering that the outsourced files may have different levels of confidentiality, our system is further designed to provide multi-level protection to the outsourced files by encrypting them with the assistance of multiple KS groups.

We summarize the main contributions of this paper as follows.

- We propose a new secure deduplication scheme for large-scale cloud storage service, which, to our best knowledge, is the first server-aided scheme that supports both the tolerance of partial KSs leakage and dynamic change of KSs. To meet comprehensive security requirements, our scheme provides users with the feasibility to use multiple KS groups to assist in encrypting files while the cloud can perform deduplication.
- Our scheme is designed to update KS groups regularly to support the dynamic change of KSs, as well as maintain long-term security. We leverage the blockchain to help generate KS groups fairly and randomly.
- We formally prove the correctness of our scheme and justify its security. Both theoretical and experimental evaluations show its modest performance overhead.

The rest of this paper is organized as follows. Section II presents the problem statement of our design, including system model, threat model, and design goals. Section III presents the preliminaries. Section IV introduces the design of our scheme. Section V proves the correctness of our scheme and justifies its security. Section VI analyzes the performance of our scheme. Section VII presents the related work and Section VIII concludes this paper.

II. Problem Statement

In this section, we first define the system model and threat model of our design, and introduce our design goals.

A. System Model

Fig. 1 shows the system model of our design and the system includes four types of entities: CSP, user, KS, and blockchain, which are described as follows.

- *CSP:* The CSP provides convenient and on-demand storage services for users. For effective storage utilization, it performs deduplication among the files from different users.
- *User:* A user is an institution or individual that outsources files to the CSP for storage. Since different users may upload identical files, a user can be an initial uploader that uploads a new file or a subsequent uploader that uploads an already existing file. To protect file confidentiality, users encrypt their files and outsource the ciphertexts to the CSP.
- *KS:* The KS assists in the generation of encryption key and file tag. In a large-scale cloud storage system, many KSs are deployed and the available KSs may change dynamically, which means that some KSs may leave and some KSs may join the system at any time.
- *Blockchain:* The blockchain is a public distributed shared ledger with many nodes. It has the characteristics of tamper resistance, decentralization, and transparency [25].

As can be seen from the architecture of our system in Fig. 1, all KSs are divided into groups, and we use the blockchain to record group information (e.g., an on-chain file-to-KS table). After dividing all the KSs into groups, each group of KSs execute the secret initialization protocol to share a secret key. When a user wants to upload a file to the CSP, he/she selects the

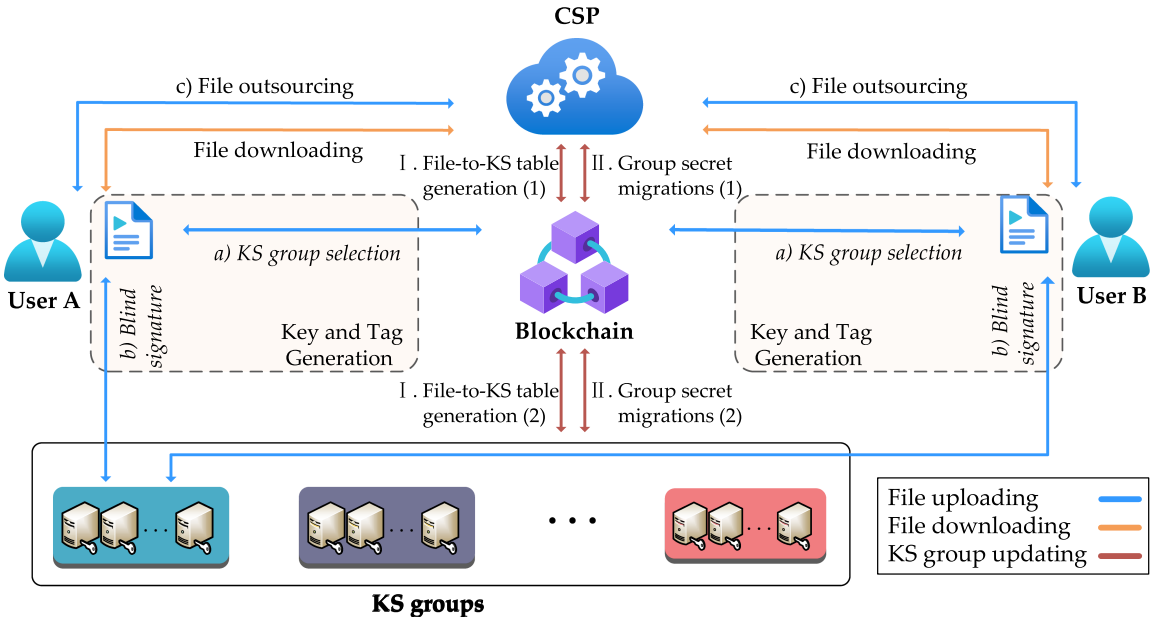

Fig. 1. System architecture of our scheme with three phases: file uploading, file downloading, and KS group updating. The file uploading involves KS group selection, blind signature, and file outsourcing, among which KS group selection and blind signature belong to key and tag generation. The KS group updating involves the sequential execution of file-to-KS table generation and group secret migrations.

corresponding KS group using the file information by interacting with the blockchain, and then generates an encryption key with the assistance of some KSs in that group. As a result, different users with identical files can select the same KS group and thus generate the same encryption keys, which can allow the CSP to perform cross-user ciphertext deduplication.

B. Threat Model

The threats of our model are from the CSP, KSs, users or their collusion. We discuss these possible threats in our model as follows:

- *CSP:* The CSP stores the ciphertexts and file tags uploaded by users. It honestly executes the protocols but may guess the private contents of its stored ciphertexts. For example, the CSP may launch offline BFA to ciphertexts using the information it kept.
- *KSs:* The KSs assist the user to generate the encryption key and tag. They honestly execute the protocols but may snoop on the private contents of the files. Besides, a KS may affect the partition of KSs such that several collusive KSs can be divided into the same group to recover the shared group secret key maliciously. Once this happens, the security level of the ciphertexts corresponding to this KS group degrades to that of the original MLE.
- *Users:* The initial uploader may launch duplicate-faking attacks (DFA), which means that he/she may maliciously upload a file tag and an unmatched ciphertext such that the subsequent uploaders owning the same file download a falsified copy from the cloud.
- *Collusion attack:* The threats may also come from the collusion attacks among different entities. First, the CSP may collude with a set of KSs to control the partition of the KS groups such that some malicious KSs can be classified into a same group to recover the group secret key. Second, as claimed in [21], [23], the CSP may also collude with a set of KSs (less than a threshold) to launch offline BFA to the ciphertexts and we make the same assumption. We suppose that the CSP does not collude with any user, which is a basic assumption for an encrypted cloud storage system [26], [27].

C. Design Goals

Our design aims to perform cross-user secure deduplication over large-scale cloud storage. The specific goals of our design are given as follows.

Functionality: Our scheme is designed to mainly support the following functionalities: *cross-user deduplication*, *dynamic change of KSs*, *tolerance of partial KSs leakage*, and *multi-level protection*. (1) Our system should ensure that the identical files from different users can be detected as duplicated files to implement cross-user deduplication. (2) Since some existing KSs may leave and some new KSs may join the system at any time in a large-scale cloud storage system, our scheme should have the scalability to deal with such change. (3) The leakage of a group secret key degrades the security level of only the ciphertexts related to this KS group, rather than all the ciphertexts stored on the cloud. (4) The outsourced files may have different levels of confidentiality. The strictly confidential files should be protected with a higher security level. Our design provides multi-level protection to the outsourced files according to users' willingness. Specifically, a file in our system can be protected by multiple KS groups.

Security: The security goals of our scheme include *data confidentiality* and *data integrity*. (1) The data confidentiality indicates that the ciphertexts are resistant to the security attacks launched by the CSP, KS, malicious user, or their collusion cases explained in our threat model. (2) The data integrity indicates that the scheme can prevent the DFA launched by the initial uploader of a file. Besides, a user can verify the integrity of files downloaded from CSP.

III. Preliminaries

A. Bilinear Pairing

Given a security parameter κ, the bilinear parameter generator $\mathcal{G}en(\kappa)$ outputs a five-tuple $(p, \mathbb{G}, \mathbb{G}_T, g, e)$, where p is a κ-bit prime, $\mathbb{G}$ and $\mathbb{G}_T$ are two multiplicative cyclic groups of prime order p, g is a generator of $\mathbb{G}$, and $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is a bilinear map [28] owning the following properties:

- Bilinear: $e(X^a, Y^b) = e(X, Y)^{ab}$ for all $X, Y \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$.
- Non-degeneracy: $e(g, g)$ is a generator of $\mathbb{G}_T$.
- Computability: For all $X, Y \in \mathbb{G}$, there exists an efficient algorithm to compute $e(X, Y) \in \mathbb{G}_T$.

B. Threshold Cryptography

For a (t, n)-threshold cryptosystem, n players share a secret and each one has a secret share. The secret can only be gained from t or more players, and the collusion of any $t-1$ or fewer players cannot access the secret from their collective shares [29].

C. Verifiable Delay Function

A delay function is a special function that cannot be evaluated in less than a prescribed time period [30]. A verifiable delay function (VDF) [30] is a kind of delay function that enables the evaluator to generate a proof to show the correctness of its computation. A VDF is defined by a three-tuple of algorithms:

- *Setup*$(1^\lambda) \rightarrow pp$*:* It takes the statistical security parameter 1^λ as input and outputs the public parameter pp.
- *Sol*$(pp, x, T) \rightarrow (y, \pi)$*:* It takes the public parameter pp, random parameter $x \in \mathbb{X}$, and time parameter T as inputs and outputs the result $y \in \mathbb{Y}$ and proof π, where $\mathbb{X}$ and $\mathbb{Y}$ are two spaces, y is a deterministic output of x taking T sequential steps to compute, and π can prove that y was computed correctly.
- *Ver*$(pp, x, T, y, \pi) \rightarrow \{1/0\}$*:* It takes the public parameter pp, random parameter x, result y, and proof π as inputs and outputs either 1 to denote an acceptance or 0 to denote a rejection.

IV. The Design of Our Scheme

In this section, we present our secure scheme in detail. Table I describes the key notations used in this paper.

A. Design Overview

Here, we give an overview of how our design can achieve the design goals presented in Section II-C.

TABLE I
Key Notations

Notation	Description
$\mathcal{U}$	User
$\mathcal{KS}$	Key server (KS)
V	Verifiable delay function
F	File
C	Ciphertext
Φ	File-to-KS table
$\Phi[i]$	i-th KS group in Φ
η	Total number of KS groups
p	A large prime number
$\mathbb{Z}_p$	The ring of integers modulo p
$\mathbb{G}, \mathbb{G}_T$	Two multiplicative cyclic groups of order p
g	A generator of $\mathbb{G}$
key_F	Encryption key for F
tag_F	Tag of F
$hash_C$	Hash value of C
$e(\cdot, \cdot)$	A bilinear pairing $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$
$f(\cdot, \cdot)$	A pseudo-random function $f : \mathbb{N} \times \mathbb{Y} \rightarrow \mathbb{N}$
$SH(\cdot)$	Short hash function
$H_1(\cdot)$, $H_2(\cdot)$, $H_3(\cdot)$, $H_4(\cdot)$	Secure Hash functions

Cross-User Deduplication: Our scheme considers large-scale cloud storage and has many KSs involved. To balance the trade-off between efficiency and security, we divide all the KSs into groups and each group works individually. In prior schemes [21], [22], users may choose different KS groups to encrypt the identical files as different ciphertexts. Then extra information should be added to ciphertexts for cross-user deduplication, resulting in security risk and high computation and storage overhead. We design a file-related KS group selection mechanism that uses the file itself to determine the KS group for generating the encryption key. We use a file-to-KS table Φ to record the KS group information and store it on the blockchain such that every entity can access but cannot temper it. Each group of KSs collaborate to generate a shared group secret key for signing users' requests. This design can guarantee that the identical files of different users can find the same target KS group and be encrypted to exactly the same ciphertexts.

Note that the KS group partition should be random and fair. Otherwise, several malicious KSs may be classified into a same group to recover the group secret key. Since the evaluation time of VDF can be controlled by a time parameter and any participant cannot predict its result within the prescribed time period, we first utilize VDF to generate an unpredictable but verifiable seed. Then, we use the seed as the key of a pseudo-random function to divide KSs into different KS groups.

Dynamic Change of KSs: Since the group information is recorded in the file-to-KS table Φ, the system can support the dynamic change of KSs by updating Φ. Specifically, the CSP and all the available KSs update Φ by executing the file-to-KS table generation protocol regularly. Note that, to increase the

flexibility of the system, each KS can belong to more than one KS group. After updating the file-to-KS table, the KS group information is changed and the shared secret key of the previous KS group needs to be migrated to the new KS group to ensure the ability of deduplication. Since our system model does not have a confidential third party, we use a secure group secret migration protocol [23] to migrate the secret key without recovering it.

Tolerance of Partial KSs Leakage: Our scheme divides all the KSs into groups and each KS group shares a randomly generated group secret key. Since the developed file-related KS group selection mechanism can guarantee that the identical files of different users can be mapped to the same target KS group, the identical files can be encrypted to the exactly same ciphertexts such that the cloud can perform deduplication. As a result, each KS group works independently and the leakage at a group of KSs degrades the security level of only the ciphertexts related to the current group, rather than all the ciphertexts stored on the cloud.

Multi-Level Protection: In practice, the outsourced files may have different levels of confidentiality. Users may be willing to protect their strictly confidential files with a higher security level. Our scheme aims to provide selectable levels of protection for different files. Specifically, users can choose multiple KS groups to jointly assist in the generation of the key for encrypting strictly confidential files. Security degrades to MLE only when the group secret keys of all the selected KS groups leak. Note that using more KS groups also cause higher price and this can be determined by the users. Since our file-related KS group selection mechanism uses the information of the file itself to select KS groups, the identical files of different users can determine the same KS groups to generate their encryption keys when the used KS group numbers are the same. As a result, our scheme can provide users the feasibility to use multiple KS groups to encrypt files as well as achieve cross-user deduplication.

Long-Term Security: In server-aided MLE schemes, the deployment of one KS introduces the single-point-of-failure issue. In such a scenario, if an attacker exposes the secret key of the KS, the overall security of the system becomes compromised. To mitigate this vulnerability, the adoption of the threshold signature method involves the deployment of multiple KSs. However, even with this approach, a persistent attacker, given enough time, may successfully compromise a threshold number of KSs and recover the shared secret key.

Our system supports the addition of new KSs and the reorganization of KS groups to ensure long-term security. In the event that an attacker successfully compromises certain KSs, the KS group updating mechanism partitions these compromised KSs into distinct KS groups, simultaneously updating the associated secret shares and making previous attacks invalid.

Data Confidentiality and Data Integrity: Our system aims to protect data confidentiality from three aspects of design. First, we use the server-aided key to encrypt the outsourced files and thus the CSP cannot obtain the file contents by launching offline BFA. Second, we develop a fair method to randomly divide all the KSs into groups, which can prevent the collusion of the CSP and a set of KSs to control the KS group partition. Third, we use a PoW protocol to resist the DFA launched by the initial uploader. For previous secure deduplication schemes [21], [22], the ciphertexts encrypted by different users may need to be converted to support cross-user deduplication, which makes the data integrity verification a challenging problem. However, our solution develops a file-related group selection mechanism that can encrypt the identical files to be the same ciphertexts. Then users can easily verify the data integrity using the hash of the ciphertext after downloading.

Algorithm 1: Group Secret Initialization.

Input: The CSP, a group of KSs $\{\mathcal{KS}_1, \mathcal{KS}_2, \ldots, \mathcal{KS}_n\}$, and threshold parameters (t, n).

Output: The secret shares $\{sk_1, sk_2, \ldots, sk_n\}$, public shares $\{pk_1, pk_2, \ldots, pk_n\}$ of $\{\mathcal{KS}_1, \mathcal{KS}_2, \ldots, \mathcal{KS}_n\}$, and a public key pk.

1: Each $\mathcal{KS}_i$ $(1 \le i \le n)$ generates a $(t-1)$-degree polynomial over $\mathbb{Z}_p$,

$$u_i(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,t-1}x^{t-1}, \quad (1)$$

where $a_{i,k} \in \mathbb{Z}_p$ $(0 \le k \le t-1)$ are randomly selected.

2: $\mathcal{KS}_i$ publishes the verification parameters $\{g^{a_{i,k}}\}$ $(0 \le k \le t-1)$ to $\mathcal{KS}_j$ $(1 \le j \le n, j \ne i)$ and CSP through blockchain, where g is a generator of $\mathbb{G}$.

3: $\mathcal{KS}_i$ computes and sends $u_i(j)$ to $\mathcal{KS}_j$ via a secure channel.

4: When $\mathcal{KS}_i$ obtains $u_j(i)$ from $\mathcal{KS}_j$, it verifies $u_j(i)$ by checking whether the (2) holds.

$$g^{u_j(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} g^{a_{j,k} \cdot i^k} \quad (2)$$

5: If all the $\{u_j(i)\}$ $(1 \le j \le n, j \ne i)$ are valid, $\mathcal{KS}_i$ computes $u_i(i)$. Then $\mathcal{KS}_i$ can compute its secret share $sk_i = \sum_{j=1}^{n} u_j(i)$ and public share $pk_i = g^{sk_i}$. Otherwise, the process terminates.

6: The CSP computes $pk = \prod_{j=1}^{n} g^{a_{j,0}}$.

B. System Setup

The CSP holds four integers $\kappa_0, \kappa_1, \kappa_2, \kappa_3$ as its security parameters. In the phase of system setup, the CSP generates some public parameters using its security parameters as follows:

- Run the bilinear parameter generator $\mathcal{G}en(\kappa_0)$ with the security parameter κ_0 and output the tuple $(p, \mathbb{G}, \mathbb{G}_T, g, e)$.
- Determine the total number of KS groups η and a short hash function $SH : \{0,1\}^* \to \{1, \ldots, \eta\}$.
- Choose a VDF V and run V.**Setup**$(\kappa_1) \to pp$ with the security parameter κ_1. The input and output space of V are $\mathbb{X}$ and $\mathbb{Y}$, respectively.
- Choose a pseudo-random function $f : \mathbb{N} \times \mathbb{Y} \to \mathbb{N}$.
- Choose four cryptographic secure hash functions: $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \mathbb{G} \to \{0,1\}^{\kappa_2}$, $H_3 : \mathbb{G} \to \{0,1\}^{\kappa_3}$, $H_4 : \{0,1\}^* \to \mathbb{X}$.

Then the CSP publishes these public parameters $\{p, \mathbb{G}, \mathbb{G}_T, g, e, \eta, pp, f, V, SH, H_1, H_2, H_3, H_4\}$ on the blockchain

TABLE II
SYSTEM LOGS OF THE SYSTEM SETUP PHASE ON THE BLOCKCHAIN

Type	Value
Public parameter	$\{p, \mathbb{G}, \mathbb{G}_T, g, e, \eta, pp, f, V, SH, H_1, H_2, H_3, H_4\}$
Indicator	$DT\|\|T\|\|t\|\|n$
Random number	$\{(ID_i, x_i)\}_{(1\le i\le N)}$
VDF result	(y, π)
Verification parameter	$\{g^{a_{i,k}^{(j)}}\}_{(1\le j\le\eta, 1\le i\le n, 0\le k\le t-1)}$
KS group public key	$\{pk^{(j)}\}_{(1\le j\le\eta)}$
File-to-KS table	Φ

and all the entities in the system can access them. Each KS is given a unique ID by the CSP when it joins in the system. After that, the CSP and all KSs collaborate to generate a file-to-KS table Φ, which will be described in Section IV-C1. The CSP and each KS group execute group secret initialization shown in Algorithm 1 (following the design of [23]). Each KS keeps its secret share privately and publishes its public share on the blockchain. The collaboration of at least t KSs in a group can recover the group secret key. Finally, the CSP publishes the public keys of all the KS groups $\{pk^{(j)}\}_{(1\le j\le\eta)}$ on the blockchain. Table II lists the data published on the blockchain during this phase.

C. System Modules

Here, we provide a detailed description of some important modules in our system, including file-to-KS table generation, key and tag generation, and group secret migration.

1) File-to-KS Table Generation: We propose a table generation module to generate a file-to-KS table fairly with the participation of the CSP and all the available KSs in the system. The generation process of the file-to-KS table Φ is as follows:

1) The CSP publishes $DT||T||t||n$ on the blockchain, where DT indicates the deadline time for KSs to publish their random numbers, T is the time parameter used in the verifiable delay function V, and (t, n) are the threshold parameters for a KS group to share its group secret key. Meanwhile, we require that the $V.\mathbf{Sol}$'s computation time (controlled by T) is larger than the time interval from the time stamp of publishing information to DT.
2) The blockchain sends the table generation notification to all the KSs in the system.
3) After receiving the notification from the blockchain, the KS randomly chooses $x \in \mathbb{Z}_p$ and publishes it on the blockchain before DT.
4) When DT reaches, the CSP collects all the random numbers from the blockchain. We suppose that there are N random numbers (represented by $\{x_1, x_2, \ldots, x_N\}$ in time order) have been published before DT and their relevant KSs' IDs are $\{ID_1, ID_2, \ldots, ID_N\}$.
5) The CSP aggregates all the random numbers as $x = H_4(x_1||x_2||\cdots||x_N)$.
6) The CSP computes $V.\mathbf{Sol}(pp, x, T) \to (y, \pi)$ and publishes (y, π) on the blockchain. All the entities in the system can compute $V.\mathbf{Ver}(pp, x, T, y, \pi)$ to check if the CSP honestly executed the process. If rejection happens, the process terminates and goes back to step 1).
7) For the i-th $(1 \le i \le \eta)$ KS group, the CSP generates n unique numbers $\{r_{i,1}, \ldots, r_{i,n}\}$ by iteratively executing $f(i \times n + j, y) \mod N$ $(j = 1, 2, \ldots)$, where f is a pseudo-random function. Then the KSs $\{ID_{r_{i,1}}, ID_{r_{i,2}}, \ldots, ID_{r_{i,n}}\}$ are recorded as the i-th KS group in Φ.
8) When the table Φ is generated and published by the CSP, all the entities in the system can check whether the CSP has honestly executed the process or not by performing step 7) using the public f and y. If an exception happens, the process terminates and goes back to step 1).

2) Key and Tag Generation: We propose a key and tag generation module that helps users to generate the encryption key and tag for a file. It contains two parts: *KS group selection* and *blind signature*. Assume that a file-to-KS table Φ has been published on the blockchain and each KS group has shared a secret key among its members with (t, n)-threshold. Our system can provide multi-level protection to a file by using different numbers of KS groups to generate the encryption key.

When a user $\mathcal{U}$ wants to upload a file F to the CSP with the protection of q KS groups, he/she first performs *KS group selection* as follows.

1) The user $\mathcal{U}$ computes the group indexes $\{sf_i = SH(i||F)\}$ $(1 \le i \le q)$.
2) The user $\mathcal{U}$ retrieves the group information $\{\Phi[sf_i]\}$ $(1 \le i \le q)$ from the blockchain. We use $\Phi[sf_i] = \{\mathcal{KS}_{i,j}\}$ $(1 \le j \le n)$ to represent the sf_i-th KS group in Φ.
3) Each $\mathcal{KS}_{i,j}$ owns a secret share $sk_{i,j}$ and the user $\mathcal{U}$ retrieves the corresponding public share $pk_{i,j}$ from the blockchain.

After getting the KS groups for assisting key generation, the user $\mathcal{U}$ performs **blind signature** with each KS group $\Phi[sf_i]$ as follows.

1) The user $\mathcal{U}$ chooses a random number $r \in \mathbb{Z}_p$ to compute the blind hash $bf = H_1(F)^r$ of the outsourced file F and sends $sf_i||bf$ to $\Phi[sf_i]$ for signature.
2) After receiving $sf_i||bf$, the KS $\mathcal{KS}_{i,j}$ responds $\mathcal{U}$ with the signature $\sigma_{i,j} = bf^{sk_{i,j}}$.
3) When receiving a signature $\sigma_{i,j}$ from $\mathcal{KS}_{i,j}$, the user $\mathcal{U}$ verifies it by checking
$$e(\sigma_{i,j}, g) \stackrel{?}{=} e(bf, pk_{i,j}). \tag{3}$$
4) If $\mathcal{U}$ has received t valid signatures from $\Phi[sf_i]$ (assuming $\{\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,t}\}$), it computes the Lagrange coefficients $\{w_{i,k} = \prod_{1\le\gamma\le t, \gamma\ne k} \frac{\gamma}{\gamma - k}\}$ $(1 \le k \le t)$ and aggregates the signature as
$$\sigma_i = \left(\prod_{k=1}^{t} (\sigma_{i,k})^{w_{i,k}}\right)^{r^{-1}}. \tag{4}$$

When the user $\mathcal{U}$ obtains q aggregated signatures, he/she can derive the encryption key key_F and file tag tag_F of the

outsourced file F as

$$key_F = H_2\left(\prod_{i=1}^{q}\sigma_i\right) \quad \text{and} \quad tag_F = H_3\left(\prod_{i=1}^{q}\sigma_i\right). \tag{5}$$

3) Group Secret Migration: We use the group secret migration method introduced in [23] to migrate a secret key between two KS groups. Suppose that $\{\mathcal{KS}_i^{(sr)}\}\ (1 \le i \le n_1)$ is the source KS group sharing a secret key sk generated by Algorithm 1 with (t_1, n_1)-threshold. Each $\mathcal{KS}_i^{(sr)}$ owns a secret share sk_i and its public share pk_i can be retrieved from the blockchain. The public key pk of the source KS group can also be retrieved from the blockchain. The target KS group $\{\mathcal{KS}_j^{(tr)}\}\ (1 \le j \le n_2)$ wants to share sk with (t_2, n_2)-threshold. The migration processes of the secret key sk from the source KS group to the target KS group are as follows.

1) The CSP selects t_1 honest KSs from the source group to perform the secret migration. We assume that $\{\mathcal{KS}_i^{(sr)}\}\ (1 \le i \le t_1)$ are the selected KSs.
2) Each selected $\mathcal{KS}_i^{(sr)}$ generates a (t_2-1)-degree polynomial over $\mathbb{Z}_p$,
$$d_i(x) = b_{i,0} + b_{i,1}x + \cdots + b_{i,t_2-1}x^{t_2-1}, \tag{6}$$
where $b_{i,k} \in \mathbb{Z}_p\ (1 \le k \le t_2-1)$ are randomly selected values and $b_{i,0}$ is set as sk_i.
3) Each selected $\mathcal{KS}_i^{(sr)}$ publishes the verification parameters $\{g^{b_{i,k}}\}\ (0 \le k \le t_2-1)$ on the blockchain.
4) Each selected $\mathcal{KS}_i^{(sr)}$ computes $d_i(j)$ and sends $d_i(j)$ to all the target KSs $\{\mathcal{KS}_j^{(tr)}\}\ (1 \le j \le n_2)$ via security channels.
5) When a target $\mathcal{KS}_j^{(tr)}$ obtains t_1 values, it computes the Lagrange coefficients $\{w_i^* = \prod_{\substack{1\le\gamma\le t_1\\ \gamma\ne i}} \frac{\gamma}{\gamma - i}\}\ (1 \le i \le t_1)$ and verifies
$$g^{d_i(j)} \stackrel{?}{=} \prod_{k=0}^{t_2-1} g^{b_{i,k}\cdot j^k}, \tag{7}$$
$$pk \stackrel{?}{=} \prod_{i=1}^{t_1} pk_i^{w_i^*}. \tag{8}$$
If all the verifications are valid, the current $\mathcal{KS}_j^{(tr)}$ notifies other target KSs. Otherwise, the process stops and goes back to step 1) with other t_1 honest KSs in the source group.
6) Each target $\mathcal{KS}_j^{(tr)}$ rebuilds its secret share and public share using the following equations
$$sk_j^* = \sum_{i=1}^{t_1} w_i^* \cdot d_i(j), \tag{9}$$
$$pk_j^* = g^{sk_j^*}. \tag{10}$$

After secret migration, all the target KSs share the secret key sk with (t_2, n_2)-threshold, and each target $\mathcal{KS}_j^{(tr)}$ holds a new secret share sk_j^* and publishes its public share pk_j^* on the blockchain. The KSs in the source group can delete their stored secret shares.

D. Phases of Our Scheme

In this section, we present the main phases in our scheme, including file uploading, file downloading, and KS group updating.

1) File Uploading: Suppose that a user $\mathcal{U}$ wants to upload a file F with the protection of q KS groups to the cloud. The detailed steps are as follows.

1) The user $\mathcal{U}$ performs key and tag generation module with parameters F and q, obtaining the encryption key key_F and file tag tag_F.
2) The user $\mathcal{U}$ encrypts F using a symmetric encryption algorithm (e.g., AES) and gets the ciphertext $C = E(key_F, F)$.
3) The user $\mathcal{U}$ sends tag_F to the CSP to detect duplication.
4) If the CSP detects that F has been stored, it requires the user $\mathcal{U}$ to execute a PoW protocol (e.g., PoW protocol introduced in [31]) to verify his/her file ownership. If the PoW verification passes, the CSP adds the link of the relevant ciphertext to $\mathcal{U}$. If F has not been stored or the PoW verification fails, the CSP requires $\mathcal{U}$ to upload the complete C.
5) The user $\mathcal{U}$ stores key_F, tag_F and $hash_C = H_1(C)$ locally and then deletes F and C.

2) File Downloading: Suppose that a user $\mathcal{U}$ wants to download a file F from the CSP. The CSP stores the pair (tag_F, C), where tag_F is the tag of F and C is the ciphertext of F. The user $\mathcal{U}$ keeps key_F, tag_F, and $hash_C$ of the file. The detailed steps are described as follows.

1) The user $\mathcal{U}$ sends a file downloading request to the CSP using file tag tag_F.
2) The CSP checks whether $\mathcal{U}$ owns the target file or not through tag_F. If $\mathcal{U}$ owns the ciphertext C identified by tag_F, the CSP sends C to $\mathcal{U}$.
3) After receiving C, the user $\mathcal{U}$ verifies its integrity by checking
$$H_1(C) \stackrel{?}{=} hash_C. \tag{11}$$
4) If the verification passes, the user $\mathcal{U}$ can recover the original file $F = D(key_F, C)$.

3) KS Group Updating: To provide long-term security, our system periodically updates KS groups. Through this process, new KSs can be integrated into the system and the offline KSs can be removed. We introduce α as an epoch identifier and $\Phi^{(\alpha)}$ means the file-to-KS table in α-epoch. Then the KS group updating is described as follows.

1) The CSP and all available KSs execute the file-to-KS table generation module introduced in Section IV-C1 to generate a new table $\Phi^{(\alpha+1)}$. Note that the KS group number η remains unchanged after system setup but threshold parameters can be changed in each epoch during this process.
2) The secret keys are migrated between the KS groups in $\Phi^{(\alpha)}$ and $\Phi^{(\alpha+1)}$. Specifically, each KS group pair $\Phi[i]^{(\alpha)}$ and $\Phi[i]^{(\alpha+1)}\ (1 \le i \le \eta)$ runs the group secret migration module in Section IV-C3 to migrate the group secret key.

TABLE III
SYSTEM LOGS OF THE KS GROUP UPDATING PHASE ON THE BLOCKCHAIN

Epoch	Indicator	Random number	VDF result	Verification parameter	File-to-KS table
α	$(DT\|\|T\|\|t\|\|n)^{(\alpha)}$	$\{(ID_i, x_i)\}^{(\alpha)}_{(1\le i\le N^{(\alpha)})}$	$(y,\pi)^{(\alpha)}$	$\{g^{b^{(j)}_{i,k}}\}^{(\alpha)}_{(1\le j\le\eta, 1\le i\le t^{(\alpha-1)}, 0\le k\le t^{(\alpha)}-1)}$	$\Phi^{(\alpha)}$
$\alpha+1$	$(DT\|\|T\|\|t\|\|n)^{(\alpha+1)}$	$\{(ID_i, x_i)\}^{(\alpha+1)}_{(1\le i\le N^{(\alpha+1)})}$	$(y,\pi)^{(\alpha+1)}$	$\{g^{b^{(j)}_{i,k}}\}^{(\alpha+1)}_{(1\le j\le\eta, 1\le i\le t^{(\alpha)}, 0\le k\le t^{(\alpha+1)}-1)}$	$\Phi^{(\alpha+1)}$

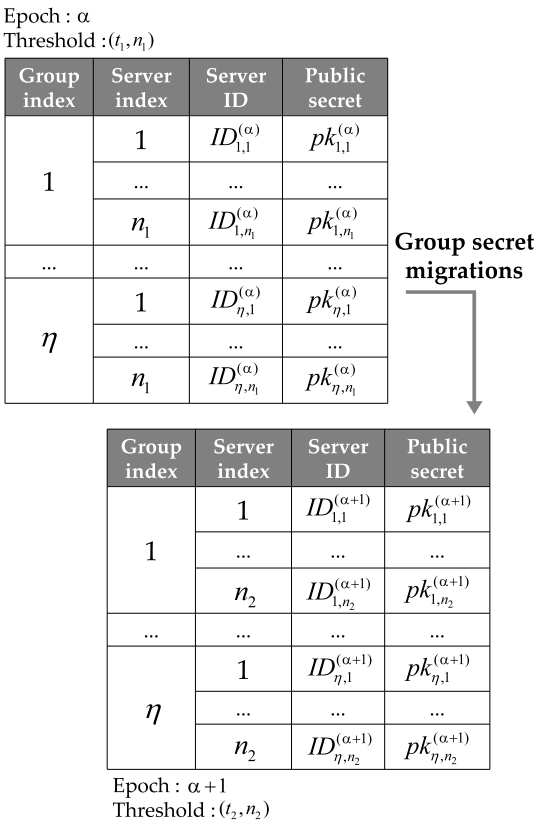

Fig. 2. Illustration of KS group updating. A new file-to-KS table $\Phi^{(\alpha+1)}$ is generated to replace the old one after updating.

After migrating all the group secret keys to the new groups in $\Phi^{(\alpha+1)}$, the system uses the new table to replace the old table $\Phi^{(\alpha)}$ to handle the file outsourcing operations. Fig. 2 shows the illustration of KS group updating. All the η group secret keys can be migrated with parallel processing. Table III lists the data published on the blockchain during this phase.

E. Disscussions

Our scheme utilizes multiple KSs to assist key and tag generation, which can achieve offline BFA resistance as well as avoid the single-point-of-failure issue. By designing a file-related KS group selection mechanism, the identical files of different users can be encrypted with the assistance of the same KS group, which ensures the effectiveness of cross-user deduplication. Our scheme updates KS group regularly to support the dynamic change of KSs and achieve long-term security for the ciphertexts stored on the cloud. Besides, since each KS group holds a different and randomly generated secret key, the leakage at a group of KSs degrades the security level of only the ciphertexts related to the current group, achieving tolerance of partial KSs leakage.

Our scheme can also support block-level deduplication. In our design, we use the short hash value of a file to determine the KS group. In the context of block-level deduplication, users need to use the short hash value of a file block to determine the KS group. Block-level deduplication requires larger communication and computational costs in comparison to file-level deduplication. This is because a user needs to perform server-aided MLE key generation for each file block in block-level deduplication. In contrast, file-level deduplication requires the user to perform server-aided MLE key generation for each file.

Our scheme sets the same threshold parameters for all KS groups within one epoch for simplicity. To enhance the system's adaptability to the dynamic change of KSs, the system allows changing the threshold parameters at the start of each epoch. Note that our system also allows choosing different threshold parameters for different KS groups within one epoch. But this may result in additional blockchain storage costs to record these different threshold parameters.

V. CORRECTNESS AND SECURITY ANALYSIS

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

A. Correctness Guarantee

1) Correctness of KS Group Updating: Here, we prove that the group updating will not change the group secret keys. We consider the migration of secret key sk between two groups and assume that the selected source KSs are $\{\mathcal{KS}_i^{(sr)}\}$ $(1 \le i \le t_1)$, and the target KSs are $\{\mathcal{KS}_j^{(tr)}\}$ $(1 \le j \le t_2)$. The secret share of $\mathcal{KS}_i^{(sr)}$ is sk_i and the that of $\mathcal{KS}_j^{(tr)}$ is sk_j^*. If sk_i and sk_j^* are valid, we can deduce that

$$\begin{aligned} sk &= \prod_{i=1}^{t_1} w_i \cdot sk_i \\ &= \prod_{i=1}^{t_1} w_i \cdot \left[\prod_{j=1}^{t_2} w_j^* \cdot d_i(j)\right] = \prod_{i=1}^{t_1}\prod_{j=1}^{t_2} w_i \cdot w_j^* \cdot d_i(j) \\ &= \prod_{j=1}^{t_2} w_j^* \cdot sk_j^*, \end{aligned} \tag{12}$$

where $\{w_i\}$ and $\{w_j^*\}$ are both Lagrange coefficients. The above equations demonstrate that both the selected source KSs and target KSs share the same group secret key sk, proving the correctness of the secret migration process.

2) Correctness of Deduplication: We prove that the ciphertexts of the same files can be deduplicated. We have proved that the KS group updating will not change the group secret keys. Then we only need to prove the deduplication in the same epoch.

Suppose that users $\mathcal{U}_A$ and $\mathcal{U}_B$ want to outsource their files with the protection of q KS groups. We use $(r_A, F_A, tag_A, key_A, C_A)$ and $(r_B, F_B, tag_B, key_B, C_B)$ to represent the random value, outsourced file, file tag, encryption key, and ciphertext generated by users $\mathcal{U}_A$ and $\mathcal{U}_B$, respectively. Besides, we use $\{sk_{i,j}^A\}$ and $\{sk_{i,j}^B\}$, $\{\sigma_i^A\}$ and $\{\sigma_i^B\}$, $\{w_{i,j}^A\}$ and $\{w_{i,j}^B\}$ $(1 \le i \le q, 1 \le j \le t)$ to indicate group secret keys, signatures and Lagrange coefficients relevant to F_A and F_B, respectively. Since our scheme uses the file-related information to select KS groups, users $\mathcal{U}_A$ and $\mathcal{U}_B$ can determine the same q KS groups when $F_A = F_B$, which indicates that $sk_{i,j}^A = sk_{i,j}^B$. From (4), we can deduce that

$$\begin{aligned}\prod_{i=1}^{q}\sigma_i^A &= \prod_{i=1}^{q}\left[\prod_{j=1}^{t}\left[(H_1(F_A)^{r_A})^{sk_{i,j}^A}\right]^{w_{i,j}^A}\right]^{r_A^{-1}}\\ &= \prod_{i=1}^{q}\prod_{j=1}^{t}H_1(F_A)^{sk_{i,j}^A\cdot w_{i,j}^A}\\ &= \prod_{i=1}^{q}\prod_{j=1}^{t}H_1(F_B)^{sk_{i,j}^B\cdot w_{i,j}^B}\\ &= \prod_{i=1}^{q}\left[\prod_{j=1}^{t}\left[(H_1(F_B)^{r_B})^{sk_{i,j}^B}\right]^{w_{i,j}^B}\right]^{r_B^{-1}}\\ &= \prod_{i=1}^{q}\sigma_i^B. \end{aligned} \tag{13}$$

Then from (5), we can further get that

$$key_A = H_2\left(\prod_{i=1}^{q}\sigma_i^A\right) = H_2\left(\prod_{i=1}^{q}\sigma_i^B\right) = key_B, \tag{14}$$

$$tag_A = H_3\left(\prod_{i=1}^{q}\sigma_i^A\right) = H_3\left(\prod_{i=1}^{q}\sigma_i^B\right) = tag_B, \tag{15}$$

$$C_A = E(key_A, F_A) = E(key_B, F_B) = C_B. \tag{16}$$

The above equations prove that different users can obtain the same ciphertexts when they outsource an identical file with the protection of same number of KS group. Then the CSP can perform deduplication in the ciphertext-domain.

B. Security Guarantee

1) Data Confidentiality: The contents of user files should be kept secret from other entities. We analyzed data confidentiality in several attack scenarios depending on the type of adversaries.

Adversarial KS: Since KSs with insufficient security protection are easy to be breached, a KS may have security vulnerabilities from both inside and outside. We now consider the security issues caused by adversarial (compromised) KS. The adversarial KS can launch security attacks using its obtained information during key and tag generation. At this stage, the KS can obtain sf and bf, where sf is a short hash of F and bf is a blind hash value of F. The short hash value exposes little information about the file [32]. Besides, our scheme is based on (t, n)-threshold blind signature, and thus at least t KSs are required to obtain the file's hash $H_1(F)$ from bf, as discussed in [33]. As a result, the adversarial KS cannot get useful information of a file using its accessible information.

Adversarial CSP: The CSP has access to users' ciphertexts and tags. We use σ, tag_F, key_F and C to represent the server-aided signature, file tag, encryption key, and ciphertext of F. Because tag_F is generated by a secure hash operation $H_3(\sigma)$, it is infeasible for CSP to obtain the signature σ from the file tag tag_F. This guarantees that the encryption key key_F cannot be deduced from tag_F. Besides, since the symmetric encryption algorithm used is semantically secure, the ciphertext C is indistinguishable from random data. Therefore, the CSP cannot get any useful information from C directly.

We then consider the offline BFA launched by the adversarial CSP to the ciphertext C. We suppose that the CSP has built a dictionary $\vec{F} = \{F_1, F_2, \cdots\}$ for guessing the plaintext F of C. To launch such an attack, the CSP needs to generate the valid ciphertexts (or tags) of files in $\vec{F}$. If a ciphertext generated from $F_i \in \vec{F}$ is identical to C, the CSP can assume that $F_i = F$ with high probability. Suppose that $pk = g^{sk}$ is the public key corresponding to the secret key of a KS group, the CSP cannot compute the group secret key sk from the public key pk in polynomial time because this is a DLP problem [34]. Without sk, the CSP cannot compute the encryption key $key_{F_i} = H_2(H_1(F_i)^{sk})$ and file tag $tag_{F_i} = H_3(H_1(F_i)^{sk})$. Therefore, the CSP can compare neither the consistency of C and $E(key_{F_i}, F_i)$ nor the consistency of tag_F and tag_{F_i} in polynomial time.

Collusion Attacks: We next consider the collusion attacks launched by the CSP and compromised KS(s), discussed in Section II-B. For simplicity, we assume that each KS group shares a secret key with (t, n)-threshold and the users upload files with the protection of $q = 1$ KS group. Note that t can enlarge with the expansion of the system and the users' data are more secure with greater t and q.

First, we consider that the CSP attempts to control the file-to-KS table generation with the compromised KS(s). The file-to-KS table Φ is generated by a pseudo-random function f with the seed produced by solving the V.**Sol**. The computation of V.**Sol** is time-consuming and the time cost is adjustable by a public time parameter T. We require that T must make the time cost of solving V.**Sol** greater than the random value commitment interval controlled by DT. Meanwhile, the input x of V.**Sol** is aggregated from all KSs' random values through a secure hash function. As a consequence, even the last KS submitting the random value cannot disturb x in line with its interests by real-time computing. In addition, the KSs in the system can use V.**Ver** to check whether the CSP has honestly executed the

pseudo random number generator. From the analysis above, any entity cannot control the generation of Φ as long as one honest KS exists in the system.

Second, we consider that the CSP and less than t compromised KSs of a KS group aim to launch offline BFA to a group-related file. Considering that our scheme executes *KS group updating* to support dynamic change of KSs. Then the worst case is that $t-1$ KS(s) in that KS group are compromised from 0-epoch to τ-epoch. Note that the group secret keys keeps the same during the KS group updating. We denote the group secret key relates to F as sk and the shares of these compromised KSs in all the $\tau+1$ epochs as $\{sk_j^{(\alpha)}\}$ $(0 \le \alpha \le \tau, 1 \le j \le t-1)$. The CSP needs to enumerate the signature σ of each file in an attack dictionary $\vec{F}$. For a file $F_i \in \vec{F}$, the CSP expects to compute $\sigma = H_1(F_i)^{sk}$ but it can build only $\tau+1$ equations denoted as

$$\sigma = \prod_{j=1}^{t} (H_1(F_i)^{sk_j^{(\alpha)}})^{w_j} \ \alpha = 0, 1, \ldots \tau. \tag{17}$$

There are $\tau+2$ unknown variables $\{\sigma, sk_t^{(0)}, \ldots, sk_t^{(\tau)}\}$, but only $\tau+1$ equations. Then the CSP cannot get the unique solution for σ, which means it cannot launch offline BFA.

2) Data Integrity: In our scheme, the data integrity indicates that the scheme can resist the DFA launched by the initial uploader of a file, and a user can verify the integrity of his/her files downloaded from the CSP, discussed in Section II-C.

For the DFA, when a subsequent uploader uploads a file F, the CSP requires he/she to execute the PoW protocol if the cloud detects that tag_F has already existed in the cloud. Since the PoW proofs can determine whether two files are the same or not, the CSP treats the uploaded file as a new file if the PoW verification fails. By this way, the DFA launched by the initial uploader is ineffective.

To verify the integrity of the downloaded file, our scheme requires the user to save the hash value $hash_C$ of the outsourced ciphertext and compare it with the hash value of the downloaded ciphertext C. Thus, the integrity of the file can be achieved in our scheme.

VI. Performance Evaluation

A. Theoretical Analysis

1) Computation Cost: Let $h_e, h_m, h_a, h_{mp}, h_h, h_b, h_p, h_{ec}, h_{dc}$ denote the computation cost of an exponentiation operation over $\mathbb{G}$, a multiplication operation over $\mathbb{G}$, an addition operation over $\mathbb{Z}_p$, a multiplication operation over $\mathbb{Z}_p$, a hash operation, a bilinear map operation, a PoW proof generation operation, a symmetric encryption operation, and a symmetric decryption operation, respectively. We consider the situation that each file is protected by one KS group, which is the most common case. Since the CSP has large computation resources, we consider the cost of only user and KS.

System Setup: The computation cost of KSs in table generation can be ignored and we mainly analyze the computation cost of a KS in one (t, n)-threshold group secret initialization. Each KS first computes the verification parameters with a cost of th_e. We do not count the cost of computing $\{x^i\}$ in polynomial as they can be computed in advance. Then the KS computes the polynomial with a cost of $(nt-n)h_{mp} + (nt-n)h_a$. The KS checks the validity of other $n-1$ KSs' values with a cost of $(nt+n)h_e + (nt-n)h_m$. If all the validation equations hold, the KS obtains its secret share and public share with a cost of $(n-1)h_a + h_e$. Overall, it spends a cost of $(nt-1)h_a + (nt-n)h_{mp} + (nt-n)h_m + (nt+n+t)h_e$ for each KS to generate a shared secret key. Users have no computation cost at this phase.

File Uploading: The user first finds the target KS group by a short hash operation with a cost of h_h. Then the user blinds the message with a cost of $h_h + h_e$ and sends it to the selected KSs. We suppose that the group secret keys are shared with (t, n)-threshold. After receiving the signatures, the user checks their validity with a cost of $2th_b$. Once the threshold parameter n is determined, the Lagrange coefficients $\{w_k\}$ $(1 \le k \le n)$ can be computed in advance. Then if all signatures are valid, the user recovers the final signatures with a cost of $(t-1)h_m + (t+1)h_e$. By performing two hash operations, the user derives the encryption key and file tag with a cost of $2h_h$. The time cost for file encryption and ciphertext hash computation is $h_{ec} + h_h$. If the file has already been uploaded to the CSP, additional PoW verification is performed with a cost of h_p. Overall, it spends a total cost of $5h_h + (t-1)h_m + (t+2)h_e + 2th_b + h_{ec}$ for an initial uploader and a total cost of $5h_h + (t-1)h_m + (t+2)h_e + 2th_b + h_{ec} + h_p$ for a subsequent uploader to upload a file with (t, n)-threshold. It spends a cost of h_e for each selected KS to sign user's request.

File Downloading: During the file downloading, a user needs to perform only a hash operation and a symmetric decryption operation to verify the integrity of the file and obtain the original file with a cost of $h_h + h_{dc}$. KSs have no computation cost at this phase.

KS Group Updating: We mainly focus on the computation cost of one secret migration from (t_1, n_1)-threshold to (t_2, n_2)-threshold. There are two kinds of KSs in the secret migration: the selected source KS and the target KS. A source KS needs a cost of $t_2 h_e$ to compute the verification parameters. We do not count the cost of computing $\{x^i\}$ in polynomial and $\{w_k\}$ $(1 \le k \le n_2)$ as they can be computed in advance. Thus, a source KS takes a cost of $(n_2 t_2 - n_2)h_a + (n_2 t_2 - n_2)h_{mp}$ to compute the polynomial. A target KS spends a cost of $(t_1 t_2 - 1)h_m + (t_1 t_2 + 2t_1)h_e$ to check the validity of these values. If all the validation equations hold, a target KS computes its share with a cost of $(t_1 - 1)h_a + t_1 h_{mp} + h_e$. Overall, it consumes a cost of $(n_2 t_2 - n_2)h_a + (n_2 t_2 - n_2)h_{mp} + t_2 h_e$ for a selected source KS, and a cost of $(t_1 - 1)h_a + t_1 h_{mp} + (t_1 t_2 - 1)h_m + (t_1 t_2 + 2t_1 + 1)h_e$ for a target KS to migrate a shared secret key.

2) Communication Cost: Let $|i|, |\mathbb{G}|, |\mathbb{Z}_p|, |y|, |\pi|, |tag|, |$ PoW$|, |C|$ denote the bit length of an integer, an element in group $\mathbb{G}$, an element in group $\mathbb{Z}_p$, a result of VDF, a proof of VDF, a file tag, a proof of PoW verification, and ciphertext, respectively. We suppose that the KS identifiers, KS group indexes, and notifications in table generation are all integers. Besides, we

also consider the situation that each file is protected by one KS group, because this is the most common case. We consider the communication cost of only user and KS.

System Setup: During the table generation, a KS publishes a random value and gets the result of VDF with $N|\mathbb{Z}_p| + |y| + |\pi| + 4|i|$ bits (N is the KS number). Then we analyze the cost of a KS during a single (t, n)-threshold secret initialization process. It spends $t|\mathbb{G}|$ bits for a KS to publish the verification parameters and $(n-1)|\mathbb{Z}_p|$ bits to send the polynomial values. After receiving other KSs' values and verification parameters with $(tn-t)|\mathbb{G}| + (n-1)|\mathbb{Z}_p|$ bits, the KS generates its public share and publishes it with $|\mathbb{G}|$ bits. Overall, it consumes a KS $(2n-2)|\mathbb{Z}_p| + (tn+1)|\mathbb{G}|$ bits in secret initialization. Users have no communication cost at this phase.

File Uploading: During the file uploading, a user sends the group index of the file to a blockchain node with $|i|$ bits. After obtaining the KS group information with $n|i| + n|\mathbb{G}|$ bits, the user performs blind signature with $t|i| + 2t|\mathbb{G}|$ bits (t and n are the parameters of (t, n)-threshold cryptography). Then a user sends a file tag to the CSP for duplication detection with $|tag|$ bits. If the file has been uploaded, the PoW protocol is executed with |PoW| bits. Otherwise, the user sends the ciphertext with $|C|$ bits to the CSP.

File Downloading: During the file downloading process, a user sends the file tag to the CSP and receives the ciphertext with $|tag| + |C|$ bits.

KS Group Updating: The communication cost of a KS in table generation is discussed in system setup and we analyze the cost of one secret migration process from (t_1, n_1)-threshold to (t_2, n_2)-threshold. There are two kinds of KSs in secret migration: selected source KS and target KS. A selected source KS spends $t_2|\mathbb{G}| + n_2|\mathbb{Z}_p|$ bits to publish the verification parameters and send the polynomial values, while a target KS spends $(t_1t_2 + t_1 + 1)|\mathbb{G}| + t_1|\mathbb{Z}_p|$ bits to receive values and publish public share. Users have no communication cost at this stage.

3) Storage Cost: Let $|tag|, |key|, |hash|, |C|$ denote the bit length of a file tag, a file key, a ciphertext hash, and a ciphertext, respectively. When uploading a non-duplicated file, the storage cost for the CSP is $|C| + |tag|$ bits and for a user is $|hash| + |key| + |tag|$ bits. When uploading an existing file, the CSP only adds a link to the user's account and the storage cost of a user is the same as that for a non-duplicated file.

B. Empirical Evaluation

1) Setup: We implement our scheme using C++ programming language, PBC library 0.5.14 version, and OpenSSL library 1.1.1 version. The blockchain prototype is based on Ethereum Geth 1.10.26 [35]. The experiments are run on a virtual machine with CentOS 7.6 system, two cores of AMD EPYC 7K62 48-Core Processor with 4 GB RAM. We instantiate bilinear pairing with $\kappa_0 = 512$, VDF proposed by Pietrzak [36] with $\kappa_1 = 100$ (the bit length of RSA modulus is 2048), Merkel tree-based PoW [31] with 1 KB block size. We set $\kappa_2 = 128$, $\kappa_3 = 512$ and we use the AES-128 as the encryption algorithm, SHA-512 as the hash function. We set the used KS group number $q = 1$ in key and tag generation.

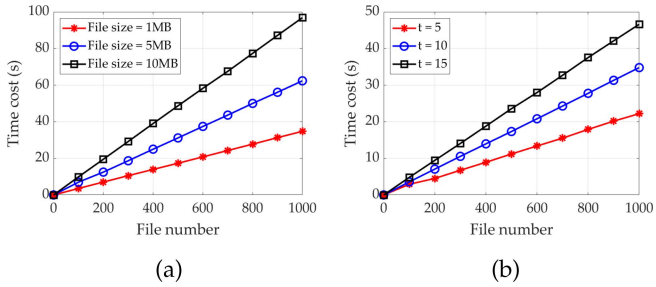

Fig. 3. Computation time cost of a user for uploading different file numbers. Time cost for (a) different file sizes and (b) different threshold parameters t.

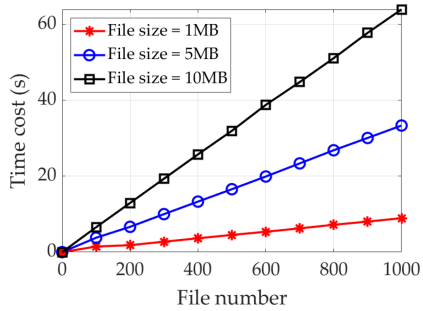

Fig. 4. Computation cost of a user for file downloading.

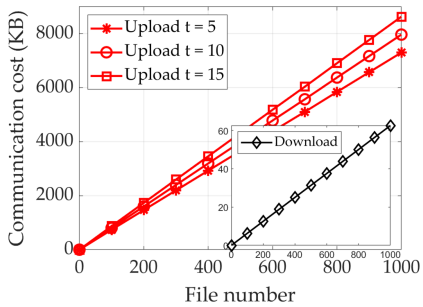

Fig. 5. Communication cost of a user for file uploading and downloading.

2) Evaluation on Off-Chain Cost: We examine the off-chain cost of our scheme by testing computation, communication, and storage cost when performing different phases in the scheme.

Computation Cost: Fig. 3 shows the uploading time cost of a user for different file numbers. Fig. 3(a) shows the cost for different file sizes with threshold parameter $t = 10$. The operations include file read, key and tag generation, and encryption. It shows that the time cost is linear with the file numbers and a larger file needs more time. Fig. 3(b) shows the cost for different t with 1 MB file size. It shows that the time cost is also affected by t because a larger t results in more time for key generation.

Fig. 4 shows downloading time cost of a user for different file numbers and sizes. The operations include integrity check and file decryption. It shows that the time cost is linear with file numbers and a larger file needs more time.

Communication Cost: Fig. 5 shows the communication cost for a user to upload and download files with 1 MB file size, 0.7 deduplication ratio and 1% challenged blocks. The uploading

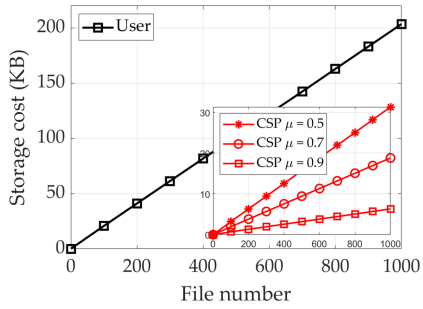

Fig. 6. Storage cost of the CSP and user.

cost includes the key and tag generation, and PoW verification. The communication cost is linear with the file numbers. Larger threshold parameters require the user to cost more for communicating with more KSs.

Storage Cost: Fig. 6 shows the storage cost on the CSP and each user. We count only the extra storage cost of the CSP and do not include the ciphertexts. For each outsourced file, a user stores the key, tag, and ciphertext hash. The storage cost is linear with the file numbers. Obviously, a higher duplication ratio μ can save more space for the CSP.

3) Evaluation on On-Chain Cost: We examine the on-chain cost of our scheme by testing the amount of gas cost by performing different phases. The gas represents the computation and storage consumption of EVM in Ethereum's ecosystem.

File-to-KS Table Generation: Fig. 7(a) shows the gas cost for different KS numbers and group numbers η. The gas cost is mainly caused by publishing the random numbers, VDF results, and file-to-KS table. In the simulation, we set the VDF parameter $T = 30$. The gas cost is linear with the KS numbers because each KS publishes a random value on the blockchain. Meanwhile, a larger η produces a larger table and thus causes a higher gas cost.

Group Secret Initialization: We simulated one group secret initialization with (t, n)-threshold. Each KS publishes its verification parameters on the blockchain. Fig. 7(b) shows the gas cost regarding the (t, n)-threshold. The gas cost is linear with n and a bigger t results in a higher gas cost. Note that η secret keys are initialized during the system setup.

Group Secret Migration: We simulated one group secret migration from (t_1, n_1)-threshold to (t_2, n_2)-threshold. Fig. 7(c) shows the gas cost regarding t_1 and t_2. The gas cost is linear with t_1 and a bigger t_2 results in a higher gas cost. There are η secret keys are migrated during the KS group updating.

VII. Relate Work

Data deduplication has been widely used in cloud storage to save storage space. However, considering the privacy issues, a user may encrypt his/her data and outsources the ciphertexts to the CSP. The traditional symmetric encryption algorithms cannot achieve cross-user deduplication over encrypted data, because each user holds an encryption key and different users encrypt an identical files into different ciphertexts. CE technique [9] can well solve this issue by deriving the encryption key from the data itself such that different users can encrypt an identical files to be the same ciphertexts and then the CSP can perform cross-user deduplication. Bellare et al. [10] formalized this kind of encryption algorithm as a new cryptographic primitive called MLE with formal security definition. Subsequently, many secure deduplication schemes based on MLE [14], [15], [37], [38], [39], [40] have been proposed for specific application scenarios. Because the encryption key in MLE is determined by the file itself, the MLE loses semantic security and the ciphertext has security risks under offline BFA [17].

There have been two kinds of research efforts aiming at resisting the private threats caused by offline BFA in secure deduplication. The first kind introduces additional KS to assist in generating the encryption key. The DupLESS proposed by Keelveedhi et al. [17] is the first such scheme and it deploys a KS with a secret key to assist in generating the encryption key. Then the encryption key depends on both the data itself and the secret key kept by the KS, achieving a high ability to resist offline BFA. The second kind does not use additional KSs and the first such scheme was proposed by Liu et al. [41]. To achieve ciphertext deduplication, the system requires that at least one previous data uploader should be online to provide assistance when a new uploader uploads a file that already exists on the cloud. This kind of secure deduplication scheme has a very strict assumption that the previous uploader needs to assist the new uploader, which is difficult to deploy on existing cloud storage systems.

Due to the high compatibility with existing cloud storage systems, many research efforts are devoted to designing secure deduplication schemes with the assistance of KS [19], [21], [22], [23]. Duan et al. [19] improved DupLESS by introducing a distributed key generation protocol to avoid the single-point-of-failure issue. The secret key is shared by a group of KSs using threshold cryptography. However, it lacks scalability to deploy more KSs for handling the possible increasing storage requirements in a large-scale cloud system. Shin et al. [21] proposed a decentralized secure deduplication scheme where multiple KS groups exist in the system. Each KS group shares a different secret key. This scheme allows CSP to detect the duplicated ciphertexts encrypted with the assistance of different KS groups. But it cannot support the dynamic change of KSs and the ciphertext size is linearly related to the number of KS groups. Yang et al. [22] solved these issues but brought the single-point-of-failure issue because a single KS is deployed in each KS group. Zhang et al. [23] developed a proactivization mechanism to support the dynamic change of KSs to provide long-term security.

We compare the functionality and security of different server-aided deduplication schemes over encrypted data in Table IV. Note that all the schemes are based on server-aided MLE and have the ability to resist offline BFA. The schemes in [17], [22] cannot resist the single-point-of-failure issue because a single KS is deployed in each group. The schemes in [17], [19], [21] do not support the dynamic change of KSs. The schemes in [17], [19], [21], [22] cannot sustain long-term security for ciphertexts since the KS group settings are fixed and a sophisticated adversary may corrupt some fixed KSs with long enough time [24]. The schemes in [17], [19], [21], [22], [23] do not have tolerance

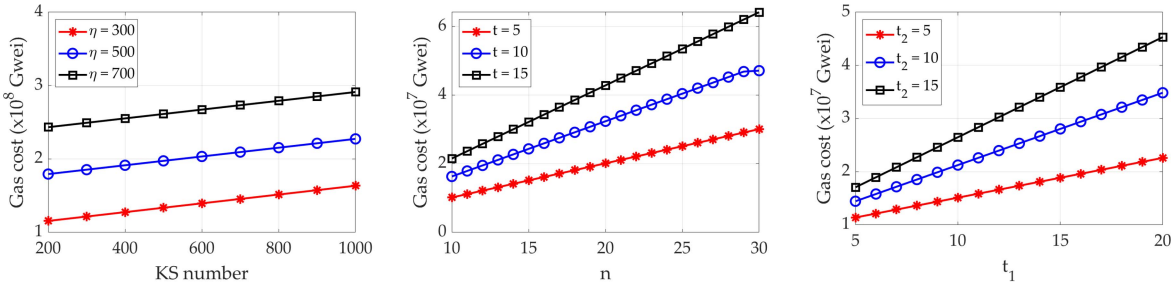

Fig. 7. Gas cost of a client for file uploading. (a) File-to-KS table generation; (b) group secret initialization; (c) group secret migration.

TABLE IV
FUNCTIONALITY AND SECURITY COMPARISONS OF DIFFERENT SERVER-AIDED DEDUPLICATION SCHEMES OVER ENCRYPTED DATA

Schemes	Offline BFA resistance	Single-point failure resistance	Dynamic change of KSs	Long-term security	Tolerance of partial KSs leakage	Multi-level protection
[17]	√	×	×	×	×	×
[19]	√	√	×	×	×	×
[21]	√	√	×	×	×	×
[22]	√	×	√	×	×	×
[23]	√	√	√	√	×	×
Ours	√	√	√	√	√	√

of partial KSs leakage and the secret key leak from a KS group will cause insecure of all the stored ciphertexts. Specifically, the schemes in [17], [19] deploy one KS or a fixed KS group and the leakage of the KS (or KS group) secret key affects all ciphertexts. For the schemes in [21], [22], special detection mechanisms have been designed to detect the duplication of encrypted files from different KS groups, however, also expose the connections between the secret keys of different KS groups. Thus, the leakage of one KS group's secret key may also result in insecure of the ciphertexts encrypted with the assistance of other KS groups. The scheme in [23] also faces the problem of partial KSs leakage, as it treated all the KSs as one group. Besides, none of these schemes support selectable multi-level protection for different files. Our design is the first scheme that considers and solves all the above issues.

VIII. CONCLUSION

In this paper, we design, analyze, and evaluate a new server-aided encrypted data deduplication scheme over large-scale cloud storage, which is the first solution to support both tolerance of partial KSs leakage and dynamic change of KSs. Our scheme divides all the KSs into groups and then develops a file-related group selection mechanism to select the KS group for assisting encryption key generation such that the identical files can be encrypted to the exactly same ciphertexts. To support the dynamic change of KSs, our scheme leverages the blockchain to design a fair KS group updating strategy and uses a migration mechanism to securely migrate the secret key from the old KS group to the new one. As a result, the identical files uploaded before and after the KS group updating can be detected as duplicated files. Meanwhile, the secret keys of different KS groups are completely irrelevant. Then the leakage at a group of KSs can affect only the ciphertexts related to that group. Besides, our design provides users the feasibility to use multiple KS groups to encrypt files as well as achieving cross-user deduplication. We formally verify the correctness of our scheme and justify its security. Both theoretical and experimental evaluations are carried out to demonstrate its modest performance overhead.

REFERENCES

[1] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu, "Cloud storage as the infrastructure of cloud computing," in *Proc. Int. Conf. Intell. Comput. Cogn. Informat.*, 2010, pp. 380–383.
[2] J. Gants, "Digital universe decade-are you ready?," 2010. [Online]. Available: http://idcdocserv.com/925
[3] H. Biggar, "Experiencing data de-duplication: Improving efficiency and reducing capacity requirements," 2007. [Online]. Available: http://media.zones.com/images/pdf/ss_backup_wp_09.pdf
[4] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. S. Shen, "Providing task allocation and secure deduplication for mobile crowdsensing via fog computing," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 3, pp. 581–594, May/Jun. 2020.
[5] Y. Zhang et al., "Improving restore performance for in-line backup system combining deduplication and delta compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, pp. 2302–2314, Oct. 2020.
[6] D. Quick and K.-K. R. Choo, "Google drive: Forensic analysis of data remnants," *J. Netw. Comput. Appl.*, vol. 40, pp. 179–193, 2014.
[7] A. Dropbox, "A file-storage and sharing service," 2016. [Online]. Available: https://www.dropbox.com
[8] Y. Zhang, Y. Mao, M. Xu, F. Xu, and S. Zhong, "Towards thwarting template side-channel attacks in secure cloud deduplications," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1008–1018, May/Jun. 2021.

[9] 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.*, 2002, pp. 617–624.
[10] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. 32nd Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2013, pp. 296–312.
[11] M. Li, C. Qin, J. Li, and P. P. Lee, "CDStore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," *IEEE Internet Comput.*, vol. 20, no. 3, pp. 45–53, May/Jun. 2016.
[12] Y. Zhang, C. Xu, H. Li, K. Yang, J. Zhou, and X. Lin, "HealthDep: An efficient and secure deduplication scheme for cloud-assisted ehealth systems," *IEEE Trans. Ind. Informat.*, vol. 14, no. 9, pp. 4101–4112, Sep. 2018.
[13] J. Li, P. P. Lee, Y. Ren, and X. Zhang, "Metadedup: Deduplicating metadata in encrypted deduplication via indirection," in *Proc. 35th Symp. Mass Storage Syst. Technol.*, 2019, pp. 269–281.
[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] 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.
[16] B. Zhang, H. Cui, Y. Chen, X. Liu, Z. Yu, and B. Guo, "Enabling secure deduplication in encrypted decentralized storage," in *Proc. 16th Int. Conf. Netw. Syst. Secur.*, 2022, pp. 459–475.
[17] S. Keelveedhi, M. Bellare, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 179–194.
[18] D. Chaum, "Blind signature system," in *Proc. Adv. Cryptol.*, 1984, pp. 153–153.
[19] Y. Duan, "Distributed key generation for encrypted deduplication: Achieving the strongest privacy," in *Proc. 6th Ed. ACM Workshop Cloud Comput. Secur.*, 2014, pp. 57–68.
[20] Y. G. Desmedt, "Threshold cryptography," *Eur. Trans. Telecommun.*, vol. 5, no. 4, pp. 449–458, 1994.
[21] Y. Shin, D. Koo, J. Yun, and J. Hur, "Decentralized server-aided encryption for secure deduplication in cloud storage," *IEEE Trans. Services Comput.*, vol. 13, no. 6, pp. 1021–1033, Nov./Dec. 2020.
[22] 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.
[23] 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.
[24] Y. Zhang, C. Xu, J. Ni, H. Li, and X. S. Shen, "Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1335–1348, Fourth Quarter, 2021.
[25] D. Puthal, N. Malik, S. P. Mohanty, E. Kougianos, and C. Yang, "The blockchain as a decentralized security framework [future directions]," *IEEE Consum. Electron. Mag.*, vol. 7, no. 2, pp. 18–21, Mar. 2018.
[26] X. Yang, R. Lu, J. Shao, X. Tang, and A. A. Ghorbani, "Achieving efficient secure deduplication with user-defined access control in cloud," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 591–606, Jan./Feb. 2022.
[27] S. Zhang, S. Ray, R. Lu, Y. Guan, Y. Zheng, and J. Shao, "Efficient and privacy-preserving spatial keyword similarity query over encrypted data," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 5, pp. 3770–3786, Sep./Oct. 2023, doi: 10.1109/TDSC.2022.3227141.
[28] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proc. 7th Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2001, pp. 514–532.
[29] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[30] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Proc. 38th Annu. Int. Cryptol. Conf.*, 2018, pp. 757–788.
[31] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 491–500.
[32] Z. Pooranian, M. Shojafar, S. Garg, R. Taheri, and R. Tafazolli, "LEVER: Secure deduplicated cloud storage with encrypted two-party interactions in cyber–physical systems," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5759–5768, Aug. 2021.
[33] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme," in *Proc. 6th Int. Workshop Public Key Cryptogr.*, 2003, pp. 31–46.
[34] D. R. Stinson, *Cryptography - Theory and Practice*. Boca Raton, FL, USA: CRC Press, 1995.
[35] G. Wood et al., "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.
[36] K. Pietrzak, "Simple verifiable delay functions," in *Proc. 10th Innovations Theor. Comput. Sci. Conf.*, 2019, pp. 60:1–60:15.
[37] S. Li, C. Xu, Y. Zhang, Y. Du, and K. Chen, "Blockchain-based transparent integrity auditing and encrypted deduplication for cloud storage," *IEEE Trans. Services Comput.*, vol. 16, no. 1, pp. 134–146, Jan./Feb. 2023.
[38] M. Song, Z. Hua, Y. Zheng, H. Huang, and X. Jia, "Blockchain-based deduplication and integrity auditing over encrypted cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 6, pp. 4928–4945, Nov./Dec. 2023, doi: 10.1109/TDSC.2023.3237221.
[39] H. Yuan, X. Chen, J. Li, T. Jiang, J. Wang, and R. H. Deng, "Secure cloud data deduplication with efficient re-encryption," *IEEE Trans. Services Comput.*, vol. 15, no. 1, pp. 442–456, Jan./Feb. 2022.
[40] H. Kwon, C. Hahn, K. Kang, and J. Hur, "Secure deduplication with reliable and revocable key management in fog computing," *Peer-to-Peer Netw. Appl.*, vol. 12, pp. 850–864, 2019.
[41] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 874–885.

Zhongyun Hua (Senior Member, IEEE) received the BS degree in software engineering from Chongqing University, Chongqing, China, in 2011, and the MS and PhD 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 works have appeared in prestigious venues, such as the *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Image Processing*, *IEEE Transactions on Signal Processing*, and *ACM Multimedia*. He has been recognized as 'Highly Cited researcher 2023' and 'Highly Cited researcher 2022'. His current research interests include focused on chaotic system, multimedia security, and secure cloud computing. He has published about eighty papers on the subject, receiving more than 5,900 citations.

Yufei Yao received the BE degree in computer science from the Harbin Institute of Technology, Shenzhen, in 2022. He is currently working toward the ME 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.

Mingyang Song received the BE and ME degrees in software engineering from Sun Yat-sen University, Guangzhou, China, in 2019 and 2021, respectively. He is currently working toward the EngD 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.

Yifeng Zheng received the PhD degree in computer science from the City University of Hong Kong, Hong Kong, in 2019. He is an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. 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, the *IEEE Transactions on Dependable and Secure Computing*, and *IEEE Transactions on Information Forensics and Security*. 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.

Yushu Zhang received the BS degree from the School of Science, North University of China, Taiyuan, China, in 2010, and the PhD degree from the College of Computer Science, Chongqing University, Chongqing, China, in 2014. He held various research positions with the City University of Hong Kong, Southwest University, the University of Macau, and Deakin University. He is currently a professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. He has authored or coauthored more than 100 research peer-reviewed journal and conference papers. His research interests include multimedia security, artificial intelligence, and blockchain. He is an associate editors for the *Information Sciences and Signal Processing*.

Cong Wang (Fellow, IEEE) is currently a professor with the Department of Computer Science, City University of Hong Kong. His research interests include data and network security, blockchain and decentralized applications, and privacy-enhancing technologies. He has been one of the founding members of the Young Academy of Sciences of Hong Kong since 2017, and has been conferred the RGC research fellow in 2021. He received the Outstanding Researcher Award (junior faculty) in 2019, the Outstanding Supervisor Award in 2017 and the President's awards in 2019 and 2016, all from the City University of Hong Kong. He is a co-recipient of the Best Paper Award of IEEE ICDCS 2020, ICPADS 2018, MSN 2015, Best Student Paper Award of IEEE ICDCS 2017, and the IEEE INFOCOM Test of Time Paper Award 2020. His research has been supported by multiple government research fund agencies, including National Natural Science Foundation of China, Hong Kong Research Grants Council, and Hong Kong Innovation and Technology Commission. He has served as the editor-in-chief of the *IEEE Transactions on Dependable and Secure Computing* (TDSC), and associate editor of the *IEEE Transactions on Services Computing* (TSC), *IEEE Internet of Things Journal* (IoT-J), *IEEE Networking Letters*, and the *Journal of Blockchain Research*, and TPC co-chairs for a number of IEEE conferences and workshops. He is a member of the ACM.