Efficient Certificateless Multi-Copy Integrity Auditing Scheme Supporting Data Dynamics

Lei Zhou, Anmin Fu, *Member, IEEE*, Guomin Yang, *Senior Member, IEEE*, Huaqun Wang, and Yuqing Zhang

Abstract—To improve data availability and durability, cloud users would like to store multiple copies of their original files at servers. The multi-copy auditing technique is proposed to provide users with the assurance that multiple copies are actually stored in the cloud. However, most multi-replica solutions rely on Public Key Infrastructure (PKI), which entails massive overhead of certificate computation and management. In this article, we propose an efficient multi-copy dynamic integrity auditing scheme by employing certificateless signatures (named MDSS), which gets rid of expensive certificate management overhead and avoids the key escrow problem in identity-based signatures. Specifically, we improve the classic Merkle Hash Tree (MHT) to achieve batch updates for multi-copy storage, which allows the communication overhead incurred for dynamics to be independent of the replica number. To meet the flexible storage requirement, we propose a variable replica number storage strategy, allowing users to determine the replica number for each block. Based on the fact that auditors may frame Cloud Storage Servers (CSSs), we use signature verification to prevent malicious auditors from framing honest CSSs. Finally, security analysis proves that our proposal is secure in the random oracle model. Analysis and simulation results show that our proposal is more efficient than current state-of-the-art schemes.

Index Terms—Cloud storage, integrity auditing, multi-copy storage, data dynamics

1 Introduction

In the era of big data, the amount of data is increasing explosively, and users can no longer manage and share data well by relying on traditional computing platforms. The emergence of cloud computing provides a new solution to this dilemma. With the cloud storage model, users can upload their data into the cloud and delete the local copy, enjoying high-quality services on a fee basis [1], [2], [3]. However, users can no longer manage the data as they process it locally [4], [5], [6], [7]. They may be concerned that Cloud Storage Servers (CSSs) do not store their data correctly. Although a Cloud Storage Provider (CSP) supporting CSSs claims that users' data has been stored correctly, it is in their interest to hide data corruption events in order to maintain their reputation. Therefore, an efficient mechanism is necessary to enable users to check the integrity of cloud data.

Integrity auditing technology is regarded as an effective means to allow users to verify whether the cloud data is stored correctly. To liberate users from the heavy computational burden, a Third Party Auditor (TPA) is introduced into the integrity auditing model to interact with the cloud on users' behalf [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], and all of these schemes mainly focus on single copy research. In single-copy proposals, despite the existence of auditing mechanisms, the damaged data is hard to recover due to the deletion of the local copy. Therefore, multi-copy storage has become an inevitable choice to improve data availability and restorability by storing multiple copies of raw data across various CSSs. A notable feature of multi-copy storage is that the damaged data can be correctly restored as long as one copy of the data stored in the cloud remains intact. Therefore, for valuable data, such as financial applications, scientific research materials, and educational documents, it is necessary to provide multi-copy storage to avoid data loss.

Despite existing multi-copy proposals [18], [19], [20], [21] devoting to improving auditing efficiency, all of them have been constructed under Public Key Infrastructure (PKI) technology, which is an uneconomical choice for users, for substantial certificate management overhead is introduced in their model. Although several multi-copy schemes [22], [23] have employed ID-based signatures [24] to reduce certificate overhead, they do not consider the data dynamics problem. Some early multi-replica achievements [19], [21] have implemented dynamic updates for all replicas, but suffer replacement attacks launched by malicious CSSs. Another multi-

- *Lei Zhou and Anmin Fu are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China , and also with the Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing 210023, China. E-mail: 1249709729@qq.com, fuam@njust.edu.cn.*
- *Guomin Yang is with the School of Computing and Information Technology, Institute of Cybersecurity and Cryptology, University of Wollongong, Wollongong, NSW 2522, Australia. E-mail: gyang@uow.edu.au.*
- *Huaqun Wang is with the Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, College of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China , and also with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China. E-mail: whq@njupt.edu.cn.*
- *Yuqing Zhang is with the National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: zhangyq@ucas.ac.cn.*

Manuscript received 19 Sept. 2019; revised 27 July 2020; accepted 30 July 2020. Date of publication 4 Aug. 2020; date of current version 14 Mar. 2022. (Corresponding author: Anmin Fu.)
Recommended for acceptance by S. Nepal.
Digital Object Identifier no. 10.1109/TDSC.2020.3013927

replica dynamic construction [20] is designed to resist replacement attacks, but it is only applicable in the case of one-by-one copy verification to locate the damaged copies. Moreover, the overhead for dynamic updating in dynamic proposals [19], [20], [21] always increases linearly with the replica number, which is impractical in reality. Therefore, reducing the cost of dynamic auditing in multi-replica constructions as much as possible, while resisting replacement attacks, is a major hurdle.

Besides data dynamics, variable copy number storage is another practical requirement worth considering. Existing multi-copy solutions always assume that all blocks of one file are copied into the same replicas for storage. In practice, some blocks in an original file do not contain valid information, while other blocks are of high value. This being the case, users might be allowed to store one or two copies for low-value blocks and store more for valuable ones. Each block in a file will be stored with a different copy number. At present, it is an exciting challenge to provide variable copy number storage for multi-replica auditing.

In addition, all of the aforementioned schemes always assume that CSSs are not entirely trusted in an integrity auditing model. But in fact, users and the TPA might also be dishonest [25], [26]. Even though CSSs have passed the verification, the user/TPA can still claim that the verification is unsuccessful to obtain compensation from the CSP. Additionally, a TPA may collude with CSSs to hide from users the fact that the stored data has been corrupted to obtain bribes from the CSP. Therefore, a fair arbitration service needs to be provided for multiple-copy storage.

In order to address these issues, we propose an efficient multi-copy integrity auditing scheme supporting data dynamics (named MDSS) by employing certificateless signatures [27], which realizes dynamic data support and provides variable copy number storage simultaneously. Also, the communication overhead of our MDSS in the dynamic update procedure is independent of the replica number, which dramatically improves the dynamic efficiency.

Our Contributions. The main contributions of this work can be summarized as follows:

- As far as we know, we are the first to implement multi-copy public auditing construction with certificateless signatures, which avoids the high costs of employing expensive certificates in PKI settings and key escrow threats in ID-based signatures.
- To achieve dynamics for a multi-replica model, we design a novel dynamic structure (called MD-MHT) that supports both block value and serial number validation. The overhead it incurs for dynamic auditing does not increase with the count of replicas. Based on the improved structure, a signature exchange verification is proposed to deal with disputes that dishonest auditors may frame CSSs for compensation.
- In order to benefit users economically, we provide a storage strategy for an uncertain copy number. The strategy allows users to determine the replica count for different data blocks, thus improving the efficiency and feasibility of multi-replica storage.
- We give the provable security analysis for MDSS in random oracle model. Moreover, theoretical and experimental analysis demonstrate that MDSS is efficient in terms of communication and computation costs.

Roadmap. The rest of the paper is organized as follows: we outline the related work in Section 2. We present the system and security model in Section 3. Then we propose the dynamic structure and present detailed algorithms of our MDSS in Section 4. Provable security analysis and performance evaluation are presented in Sections 5 and 6. Finally, we give the conclusion of our paper in Section 7.

2 Related Work

Ateniese *et al.* [8] invented the concept of Provable Data Possession (PDP), where an auditor or the user itself can verify the data stored at untrusted servers. Homomorphic verifiable authenticators were invented to aggregate many proofs into a constant value, realizing batch auditing with an acceptable communication overhead. However, the proposed scheme was designed only for static data. Subsequently, Erway *et al.* [9] extended the model of PDP, and first proposed a fully dynamic PDP construction, where Rank-based Authenticated Skip List (RASL) was designed for supporting full dynamic updating. Wang *et al.* [10] proposed another dynamic solution based on the Merkle Hash Tree (MHT). MHT is an in-depth research verifiable structure [28], which aims to effectively and safely prove that a set of elements is undamaged and unchanged. Unfortunately, only verifying the hash values of the nodes made the proposal vulnerable to replace attacks launched by malicious CSSs. Further, Zhu *et al.* [11] developed another fully dynamic scheme for cloud storage data via a designed structure, Index-Hash Table (IHT). However, any insert and delete operations will cause the tag recalculation of all data blocks located behind the operated block, thus incurring high computation costs. Subsequently, similar solutions are proposed to improve the efficiency of dynamic update [5], [29]. Other research aspects have also been intensively studied, such as privacy protection [30], [31], user revocation [32], [33], group sharing [34], [35], user key updates [14], [15], fog-based clouds [36], etc. In order to relieve the certificate usage, Li *et al.* [37] implemented fuzzy identity-based auditing for cloud data. Shen *et al.* [38] achieved identity-based integrity auditing for secure sharing with hiding sensitive information. Nevertheless, due to the inherent defect of the identity cryptosystem, the above proposals suffer from the key escrow problem. To solve this issue, Li *et al.* [39] utilized the certificateless signature to enable integrity checking of shared data, where certificates are not required and key escrow issue is eliminated simultaneously. Recently, Zhang *et al.* [40] presented another public auditable proposal by combination certificateless signatures with blockchain, which achieves resistance against procrastinating auditors.

To achieve public auditing in multi-copy storage, Curtmola *et al.* [18] proposed the first multi-copy scheme based on RSA signature. Unfortunately, data dynamics could not be supported, for block numbers are involved in calculating validation tags. To support dynamic updates, Barsoum *et al.* [19] further proposed two dynamic proposals by employing BLS signature, TB-DMCPDP and MB-DMCPDP. However, the costs for TB-DMCPDP and MB-DMCPDP increase with the number of replicas. Moreover,

TABLE 1
Notations and Descriptions

Notations	Descriptions
q	A large prime
G_1	Cyclic multiplicative group with order q
G_2	Cyclic multiplicative group with order q
Z_q	$\{1, 2, \ldots, q-1\}$
e	A bilinear pairing $e : G_1 \times G_1 \rightarrow G_2$
m_{max}	Maximum number of stored copies for all blocks
m_i	Number of stored copies for ith block, $i \in [1, m_{max}]$
b_{ij}	The jth replica block of ith block
P	The proof generated based on the challenge $chal$
req	The dynamic update request created by the DO
P_{dy}	The dynamic update proof generated by the CSSs
$Pair$	The time cost of one bilinear pairing operation
Mul	The time cost of one multiplication operation in G_1
Exp	The time cost of one exponentiation operation in G_1
$\|p\|$	The size of an element in G_1 in bits
$\|q\|$	The size of an element in Z_q in bits
n	The block number of the original file F
s	The number of sectors divided for each block
c	The number of challenged blocks for each auditing

TB-DMCPDP is subject to replay attacks launched by a malicious CSS. Liu *et al.* [20] also proposed a multi-copy public auditing scheme called MuR-DPA by improving a classic MHT. MuR-DPR was designed for verifying all replicas one by one; thus it incurs a large amount of computing and communication overhead for users. Zhang *et al.* [21] also put forward a multi-copy dynamic construction by improving the MHT. However, each node of the improved hash tree stores four elements, which introduces additional communication overhead for dynamic updates in public auditing. Peng *et al.* [22] introduced ID-based signatures into multi-copy auditing, which reduces the use of certificates. In the replica generation phase, the replica number and block number are used to generate differentiable multiple replicas, so the model cannot support data dynamics. Recently, Li *et al.* [23] presented another identity-based multi-copy scheme in multi-cloud storage, which stores many copies across several CSSs.

In summary, majority of existing multi-copy proposals incur high overhead costs for certificate management, for they are designed with PKI technique, and the constructions predicated on ID-based signatures suffer from the intrinsic key escrow problem. Besides, all dynamic multi-replica proposals suffer the inefficiency that the costs for updating increase linearly with the replica count. Therefore, it is motivated to design an effective dynamic multi-copy integrity auditing scheme, where the overhead for dynamics is independent of the replica number.

3 System and Security Model

In this section, we will present the system and security model of the proposal. Additional notations and descriptions for this paper are defined in Table 1.

3.1 System Model

Our multi-copy public auditing model, shown in Fig. 1, contains five entities: a Data Owner (DO), a TPA, several CSSs supported by a CSP, a Key Generation Center (KGC), and a Trusted Arbitration Entity (TAE). Their responsibilities and obligations are as follows.

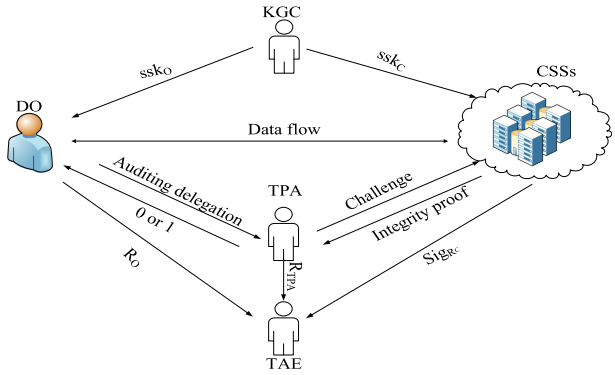


Fig. 1. System model.

- DO: The DO identified by ID_O generates a few copies for low-value data blocks and multiple replicas for high-value blocks, then all blocks are uploaded into CSSs for storage. To protect data integrity, the DO needs to generate a tag for each data block by using its private key.
- TPA: The TPA is an entity with more computing resources and expertise than users. After the approval of a DO, it launches a random challenge for integrity auditing.
- CSSs: CSSs are resource centers with powerful computing power and sufficient storage space. CSSs are responsible for storing data and responding to challenges from a DO or TPA at any time. Here we assume that all CSSs are supported by a CSP, meaning all CSSs share the same key pair and identity ID_C.
- KGC: The KGC is responsible for generating partial keys for other parties involved in the integrity system based on a given identity.
- TAE: The TAE is a trusted party that provides fair arbitration service in the model. In reality, the TAE can be a trusted government agency.

Definition 1. *Our MDSS system can be implemented by running an MDSS scheme in four stages, illustrated in Fig. 2, where the setup stage will be executed at the system beginning, for only once for one file owned by a DO; the proof stage and update stage can be executed multiple times in an arbitrary order. The arbitration stage will be run after the proof stage or update stage being run at least for once. Specifically, our MDSS scheme is composed of ten algorithms, described below:*

- $KeyGen(1^\lambda)$. *The algorithm is run between the KGC and the DO/CSSs to generate key pairs and system parameters, which will be used in the following algorithms.*
- $CopyGen(F, name, n, m_i, m_{max}, sk_O)$. *The algorithm is run by the DO to generate m_i differentiable replicas for each block.*
- $TagGen(params, b_{ij}, sk_O, ID_O, name, m_i, m_{max})$. *The algorithm is run by the DO to generate verifiable tags, MD-MHT, the root and the signature over the root.*
- $Store(b_{ij}, \sigma_i, sk_O, Sig_{R_O})$. *The algorithm is run between the DO and CSSs to agree on uploading*

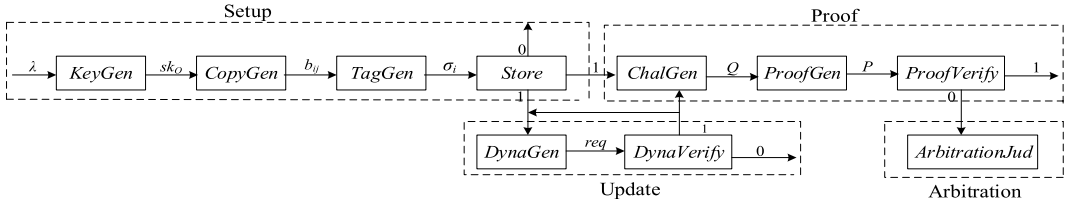


Fig. 2. The procedure of the proposal.

information. It outputs 1 or 0, where $1/0$ *indicates that the DO and CSSs agree/disagree on the uploading data.*

- $ChalGen(params, n, c)$*. The algorithm is run by the DO/TPA to generate a random challenge* Q*.*
- $ProofGen(Q, params, name, c, b_{ij}, \sigma_i, pk_O)$*. The algorithm is run by the CSSs to generate the integrity proof* P*.*
- $ProofVerify(P, pk_O, ID_O, params)$*. The algorithm is run by the DO/TPA to verify the proof* P*. The algorithm outputs a decision bit* $b \in \{0, 1\}$*, where* $1/0$ *represents that the CSSs pass/not pass the DO/TPA's verification.*
- $DynaGen(name, b_{ij}, \sigma_i)$*. The algorithm is run by the DO to generate a dynamic request* req*.*
- $DynaVerify(req, params, sk_O, F, name)$*. The algorithm is run by the DO to output a decision bit* $b_1 \in \{0, 1\}$*, where* $1/0$ *represents that the DO approves/disapproves the CSSs' dynamic updating operation.*
- $ArbitrationJud(Sig_{R_C}, R_O, R_{TPA}, R_{TAE})$*. The algorithm is run by the TAE to output a decision bit* $b_2 \in \{-1, 0, 1, 2\}$*, where* $-1/0/1$ *represents a dishonest CSS/DO/TPA respectively and 2 refers to a scenario where both the DO and TPA are dishonest.*

3.2 Security Model

First, we will give two security assumptions that provide the cornerstone for subsequent provable secure analysis.

Definition 2 (Discrete Logarithm (DL) Problem). *For a unknown value* $a \in Z_q$*, given* $g, g^a \in G_1$*, output* a*. The DL Assumption in* G_1 *holds if it is computationally infeasible to compute* a *with given tuple* (G_1, g, g^a)*.*

Definition 3 (Computing Diffie-Hellman (CDH) Problem). *For unknown* $a, b \in Z_q$*, given* $g, g^a, g^b \in G_1$*, output* g^{ab}*. The CDH Assumption in* G_1 *holds if it is computationally infeasible to compute* g^{ab} *with given tuple* (G_1, g, g^a, g^b)*.*

Since our MDSS is constructed based on certificateless signatures, we consider three types of attackers, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$. $\mathcal{A}_1$ tries to forge the tag for a block with the capability of replacing the DO's public key with any selected value, but cannot access the master secret key. $\mathcal{A}_2$ attempts to forge the tag for a block with the capability of accessing the master secret key, but is unable to replace the DO's public key. $\mathcal{A}_3$ aims to forge the integrity proof to cheat the DO. Here we define our security model through three games between a challenger $\mathcal{C}$ and three types of adversaries respectively.

Game 1. The game is run between $\mathcal{C}$ and $\mathcal{A}_1$.

Setup. $\mathcal{C}$ executes the system initialization to obtain parameters $params$ and the master secret key msk. Then $\mathcal{C}$ sends $params$ to $\mathcal{A}_1$ and keeps msk secret.

Queries. $\mathcal{A}_1$ can pose a series of different queries to C. C responds to $\mathcal{A}_1$'s inquiries as follows.

1) Hash Queries: $\mathcal{A}_1$ adaptively makes a series of hash queries to C. C responds with the hash values to $\mathcal{A}_1$.
2) Partial-Key Queries: $\mathcal{A}_1$ adaptively sends several selected IDs to C. C responds with the partial keys to $\mathcal{A}_1$.
3) Secret-Value Queries: $\mathcal{A}_1$ adaptively sends several selected IDs to C. C sends the secret values to $\mathcal{A}_1$.
4) Public-Key Queries: $\mathcal{A}_1$ adaptively sends several selected IDs to C. C responds with the public keys to $\mathcal{A}_1$.
5) Public-Key Replace: $\mathcal{A}_1$ can replace the public key of a user identified ID with any value.
6) Copy Queries: $\mathcal{A}_1$ adaptively chooses some blocks and sends them to C for getting the replicas of them. C runs $CopyGen$ and sends the replicas to $\mathcal{A}_1$.
7) Tag Queries: $\mathcal{A}_1$ adaptively chooses the tuple (b, ID) and sends it to C in order to obtain the tag on block b, computed by the user ID. C executes $TagGen$ to produce the tag on block b and responds with the tag value to $\mathcal{A}_1$.

Forge. $\mathcal{A}_1$ outputs a signature σ' on all sectors of block b' with the identity ID' and $pk_{ID'}$. $\mathcal{A}_1$ will win the game if the following conditions are achieved:

1) The generated tag σ' forged by $\mathcal{A}_1$ is valid for block b' with the identity ID' and the public key $pk_{ID'}$.
2) $\mathcal{A}_1$ does not query the whole secret key of the user identified by the identity ID'.
3) $\mathcal{A}_1$ does not query the partial key of the user identified by ID' and replaces the public key identified by ID'.
4) $\mathcal{A}_1$ does not query the tag value for (ID', b').

Game 2. The game is run between $\mathcal{C}$ and $\mathcal{A}_2$, which is similar to game 1 with some differences: (1) In setup, $\mathcal{C}$ sends both $params$ and msk to $\mathcal{A}_2$. (2) $\mathcal{C}$ does not make Partial Key Queries and Public Key Replace.

Forge. $\mathcal{A}_2$ outputs a signature σ' on all sectors of block b' with the identity ID' and the public key $pk_{ID'}$. $\mathcal{A}_2$ will win the game if the following conditions are achieved:

1) The generated tag σ' forged by $\mathcal{A}_2$ is valid for block b' with the identity ID' and the public key $pk_{ID'}$.
2) $\mathcal{A}_2$ does not query the secret value of ID'.
3) $\mathcal{A}_2$ does not query the tag value for (ID', b').

Definition 4. *If the probability of an adversary* ($\mathcal{A}_1$ *or* $\mathcal{A}_2$) *winning game 1 or game 2 is negligible, the signature of the block copies of each raw block is unforgettable.*

Game 3. The game is run between $\mathcal{C}$ and $\mathcal{A}_3$. Here $\mathcal{A}_3$ is regarded as untrustworthy CSSs. Game 3 focuses on

whether CSSs can forge the auditing proof without the accurate data. The process of the game is defined as follows.

Setup. C performs $KeyGen$ to obtain msk, $params$, and the private key of the user. Then C keeps msk and the private key of the user secret and sends $params$ to $\mathcal{A}_3$.

CopyGen Query. $\mathcal{A}_3$ adaptively sends the tuple (b, ID, m_b) to C to obtain the replicas $\{b_l\}_{1\le l\le m_b}$, where m_b refers to the copy number of block b. C runs $CopyGen$ to create m_b copies for block b and sends the copies to $\mathcal{A}_3$.

Challenge. C presents a random challenge Q to $\mathcal{A}_3$ to require $\mathcal{A}_3$ to respond to the corresponding proof.

Forge. $\mathcal{A}_3$ generates the proof P according to Q. If P can pass C' verification, we will say that $\mathcal{A}_3$ wins game 3.

Definition 5. *If the probability of an adversary $\mathcal{A}_3$ winning game 3 is negligible, the single signature of each data block is unforgettable.*

4 Our Proposed Scheme

In this section, we describe the dynamic structure for multiple-replica updating, named MD-MHT. Then we explain how our scheme is constructed based on MD-MHT.

4.1 Designed Dynamic Structure MD-MHT

Our MD-MHT is constructed by using a cryptographic hash function H. Each node N in the MD-MHT stores three elements; one is the hash value h_N, and the others are the location information (l_N, p_N) of the node, where l_N refers to the level information of the node and p_N refers to the position information of the node in its layer. In order to give each node unique location information, the MD-MHT is marked with hierarchical information from top to bottom and position information from left to right. Compared with traditional MHT, our MD-MHT has the following advantages. (1) MD-MHT significantly reduces the cost for multi-copy updates. Suppose that the cost of using MHT to support one copy update is t, and the cost of using MHT to support m copies update is $m * t$. In contrast, the cost of employing our MD-MHT for multi-copy updates is independent of m, and it is slightly greater than t for each node needs to store the location information of small size. (2) Only verifying the hash value makes the MHT suffer from replacing attacks, while our MD-MHT supports simultaneous verification of value and location to resist replacing attacks.

In our MD-MHT, we treat the aggregated hash value of all replicas of a block as the value of the leaf node. For each non-leaf node, $\{h_N = H(h_{lchildren}\|h_{rchildren}\|H(l_N\|p_N)), l_N = l_{lchildren} - 1 = l_{rchildren} - 1, p_N = \lceil p_{lchildren}/2\rceil = \lceil p_{rchildren}/2\rceil\}$, where $h_{lchildren}$ represents the hash value stored by N's left child node, and $h_{rchildren}$ represents the hash value stored by N's right child node. We set the location information of the root R as $null$, namely $l_R = null, p_R = null$. According to the bottom-to-top order, it is easy to get the hash value and location information of all nodes to construct our MD-MHT. Meanwhile, the verification path is defined as the sibling nodes on the path upward from the node to the root. For instance, if the fourth block needs to be updated, we provide the verification path $(h_3, 3, 3), (h_a, 2, 1), (h_B, 1, 2)$. The verifier calculates $h_b = H(h_3\|h_4\|H(l_b\|p_b))$, where l_b is $3 - 1 = 2$ and p_b is $\lceil 4/2\rceil = 2$. Then $\{h_A = H(h_a\|h_b\|H(l_A\|p_A)), l_A = 1, p_A = 1\}$ and $\{h_R = H(h_A\|h_B\|H(l_R\|p_R)), l_R = 0, p_R = 1\}$ are calculated. As showed in Fig. 3, assuming the first block has three copies, namely b_{11}, b_{12}, b_{13} and the second has two copies, namely b_{21}, b_{22}. According to our construction, the first leaf node is set to $h_1 = H(H(b_{11})\|H(b_{12})\|H(b_{13}))$ and the second node is set to $h_2 = H(H(b_{21})\|H(b_{22}))$. That is, each leaf node stores the hash value calculated using all copies of the corresponding block. Note that the dynamic overhead in our proposal is independent of the replica number and it can support updates of blocks with different replicas.

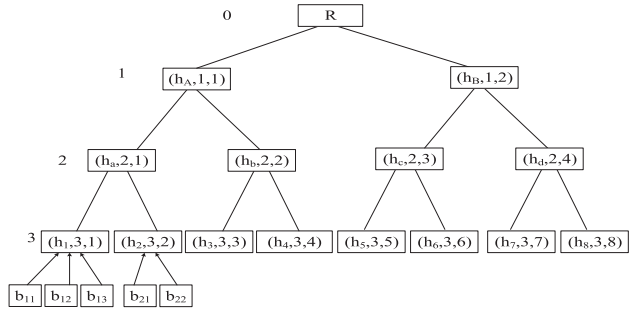


Fig. 3. The structure of the MD-MHT.

4.2 Construction of Our Proposal

Based on the MD-MHT, we have constructed a proposal where all algorithms run as follows.

4.2.1 Setup Stage

DO/CSSs interact with the KGC by running $KeyGen$ to obtain key pairs. For $F = \{b_i\}_{1\le i\le n}$, the DO runs $CopyGen$ to generate $\{b_{ij}\}_{1\le j\le m_i}$ for each block. Then, the DO uses $TagGen$ to generate an aggregation tag σ_i for all replicas of b_i, create a MD-MHT tree, and produce a signature Sig_{R_O}. Finally, the DO uploads $(\{b_{ij}\}, \{\sigma_i\}, \mathrm{MD-MHT}, Sig_{R_O})$ into CSSs, while deleting local storage other than R_O. The CSSs verify the validity of the uploading data, and accept the DO's data if the verification passes by executing $Store$.

$KeyGen(1^\lambda)$. Let G_1 and G_2 be two multiplicative groups of prime order q, where q is a large prime. e is selected as a computable bilinear pairing: $e : G_1 \times G_1 \to G_2$, and g is a generator of G_1. Four cryptographic hash functions, $H : \{0,1\}^* \to G_1$, $H_1 : \{0,1\}^* \to G_1$, $H_2 : G_1 \to Z_q$, $H_3 : Z_q \times \{0,1\}^* \to Z_q$, $H_4 : \{0,1\}^* \to G_1$, are selected. The KGC selects a master key $msk = s \in Z_q$, and computes a system public key $mpk = g^s \in G_1$. The KGC publishes system public parameters $params = \{G_1, G_2, q, e, g, H, H_1, H_2, H_3, H_4, mpk\}$, and keeps the secret key msk private. Given $params$, the DO and CSSs obtain a pair of public and private keys, (sk_O, pk_O) and (sk_C, pk_C), through the following steps, respectively. Note that the DO's secret key is used for three purposes: one is participating in the replica generation; the second is involved in the tag generation; the third is producing a signature over the root during the update process. On the contrary, the secret key obtained by the CSSs is only used to generate a signature over the root in the update phase.

- The DO sends its identity $ID_O \in \{0,1\}^*$ to the KGC to obtain its partial key. The KGC computes $ssk_O = H_1(ID_O)^s$ and returns ssk_O to the DO. After receiving

ssk_O, the DO verifies the formula $e(ssk_O, g) = e(H_1(ID_O), mpk)$. If the validation fails, the DO requests a partial private key again. In this case, the DO selects $x_O \in Z_q$ as a secret value and computes:

$$sk_O = \{sk_{O1} = ssk_O, sk_{O2} = (x_O + H_2(ssk_O)) \bmod q\}, \quad pk_O = g^{sk_{O2}}. \tag{1}$$

- The CSSs send $ID_C \in \{0,1\}^*$ to the KGC to obtain a partial key. The KGC computes $ssk_C = H_1(ID_C)^s$ and returns it to the CSSs. After receiving ssk_C, the CSSs verify the equation $e(ssk_C, g) = e(H_1(ID_C), mpk)$. If the validation fails, the CSSs request a partial private key from the KGC again. In this case, the CSSs select $x_C \in Z_q$ as a secret value, and compute:

$$sk_C = (x_C + H_2(ssk_C)) \bmod q, pk_C = g^{sk_c}. \tag{2}$$

$CopyGen(F, name, n, m_i, m_{max}, sk_O)$. Assume an original file F, and we divide it into n blocks, where each block has the same size. If the size of the last block is less than that of others, then fill it with 0 at the end. Note that the length of the file itself is known, and then the number of blocks will be obtained in the process of file partitioning. Based on the above two factors, although the last block may be filled with 0 to achieve the same length as other blocks, its length is still clear to its owner. Therefore, even if the file ends with 0, the original number of 0 can be easily recovered. The DO selects a random $name \in \{0,1\}^*$ for F and creates m_i copies for ith block, where $m_i \in [1, m_{max}]$, and the value of m_{max} is the max value of replica number m_i determined by the DO. For each block b_i, the DO computes $b_{ij} = b_i + H_3(sk_{O2}, name \| j)$, where $j \in [1, m_i]$. Note that for any block b_{ij}, the DO is able to recover the plaintext $b_i = b_{ij} - H_3(sk_{O2}, name \| j)$ easily. Thus, the DO can get all replica blocks b_{ij} of each original block b_i in F. Then the operated block b_{ij} is further fragmented into s sectors $\{b_{ijk}\}_{1 \le k \le s}$, where each sector belongs to Z_q. The number of the sectors relies on the value of q and the block size, namely $s = \lceil |b_{ij}|/|q| \rceil = \lceil |F|/(n|q|) \rceil$, where $|\cdot|$ represents the bit length.

$TagGen(params, b_{ij}, sk_O, ID_O, name, m_i, m_{max})$. The DO selects s values $\{a_k \in Z_q\}_{1 \le k \le s}$ randomly and computes s public values $A_k = g^{a_k}$. After that, the DO produces the tag for b_{ij} by computing

$$\sigma_{ij} = sk_{O1} \cdot (H_4(bid_{ij}) \cdot g^{\sum_{k=1}^{s} a_k b_{ijk}})^{sk_{O2}}, \tag{3}$$

where $bid_{ij} = name \| ID_O \| m_{max} \| j \| h_i$ and h_i is the hash value stored by the ith leaf node in our MD-MHT, namely $h_i = H(H(b_{i1}) \| \cdots | H(b_{ij}) \| \cdots \| H(b_{im_i}))$. Then the DO produces an aggregated tag $\sigma_i = \prod_{j=1}^{m_i} \sigma_{ij}$ for all the replica blocks of the same indices. Meanwhile, the DO also initializes the MD-MHT, generates a root R_O based on the MD-MHT, and computes a signature over the root $Sig_{R_O} = H_4(R_O)^{sk_{O2}}$.

$Store(b_{ij}, \sigma_i, sk_O, Sig_{R_O})$. Then the DO forwards all blocks, corresponding tags and the MD-MHT along with Sig_{R_O} into CSSs for storage, and deletes all information other than R_O. When receiving these data, the CSSs first verify the consistency between blocks and tags by

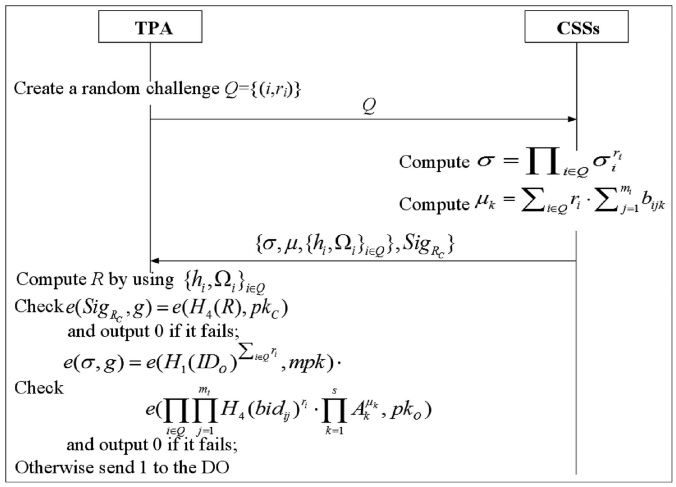


Fig. 4. The description of the proof stage.

$$e(\sigma_i, g) = e(H_1(ID_O), mpk) \cdot e\left(\prod_{j=1}^{m_i} H_4(bid_{ij}) \cdot \prod_{j=1}^{m_i} \prod_{k=1}^{s} A_k^{b_{ijk}}, pk_O\right). \tag{4}$$

If the validation fails, the CSSs refuse to store the DO's data and the algorithm outputs 0. Otherwise, the CSSs compute the root R_C based on the transmitted MD-MHT and verify the validity of Sig_{R_O} by $e(Sig_{R_O}, g) = e(H_4(R_C), pk_O)$. If the validation fails, the algorithm terminates and outputs 0; otherwise the CSSs store all the relevant data, generate $Sig_{R_C} = H_4(R_C)^{sk_C}$ and send Sig_{R_C} to the DO. The DO checks the validity of Sig_{R_C} by comparing $e(Sig_{R_C}, g)$ with $e(H_4(R_O), pk_C)$. If the verification passes, the algorithm outputs 1, meaning that the DO believes that the CSSs store the MD-MHT honestly; otherwise the algorithm terminates and outputs 0. After that, the DO sends the root R_O to the TAE and the TPA, and both the TAE and the TPA will store the root $R_{TAE}/R_{TPA} = R_O$.

4.2.2 Proof Stage

The proof stage involves multiple proof sessions. In each proof session, as shown in Fig. 4, the TPA first computes a random challenge Q by performing $ChalGen$. Based on Q, the CSSs generate the proof P by launching $ProofGen$. The TPA checks the validity of P by launching $ProofVerify$.

$ChalGen(params, n, c)$. In each audit, the TPA creates a random $Q = \{(i, r_i)\}$, where i is randomly selected from I selected subset of $[1, n]$ with c elements and $\{r_i \in Z_q\}_{i \in I}$ are randomly chosen as c coefficients. Afterwards the TPA sends Q to the CSSs for integrity auditing.

$ProofGen(Q, params, name, c, \{b_{ij}\}, \sigma_i, pk_O)$. When receiving the challenge Q, the CSSs generate the integrity proof $P = \{\sigma, \mu\}$ by validating the following formula: $\sigma = \prod_{i \in Q} \sigma_i^{r_i} \in G_1$, $\mu_k = \sum_{i \in Q} r_i \cdot \sum_{j=1}^{m_i} b_{ijk}$. Ultimately, the CSSs return P to the TPA. In addition, the CSSs also return some auxiliary verification information $\{\{h_i, \Omega_i\}_{i \in Q}, Sig_{R_C}\}$, where $\{\Omega_i\}_{i \in Q}$ represents the node siblings on the path from leaf nodes $\{h_i\}_{i \in Q}$ to the root R in our MD-MHT.

$ProofVerify(P, pk_O, ID_O, params)$. Upon receiving the responses from the CSSs, the TPA computes the root R using $\{h_i, \Omega_i\}_{i \in Q}$ and authenticates R by checking $e(Sig_{R_C}, g) = e(H_4(R), pk_C)$. If the verification fails, the TPA rejects

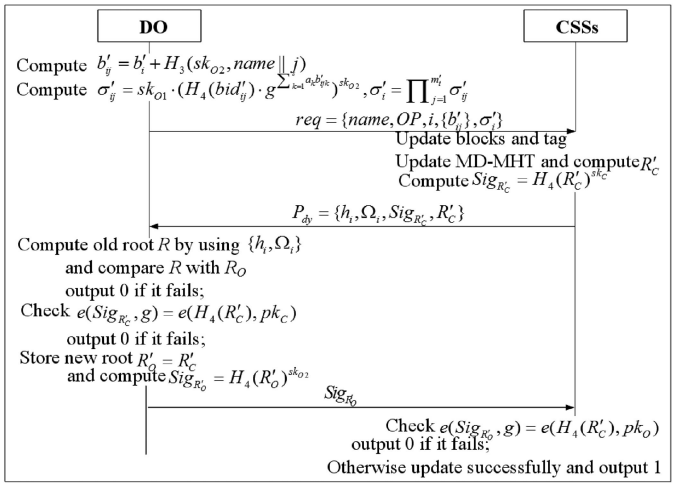


Fig. 5. The description of the update stage.

by emitting 0 to the DO. Otherwise, the TPA verifies the proof P by verifying the following formula:

$$e(\sigma, g) = e(H_1(ID_O)^{\sum_{i \in Q} r_i}, mpk) \cdot e\left(\prod_{i \in Q} \prod_{j=1}^{m_i} H_4(bid_{ij})^{r_i} \cdot \prod_{k=1}^{s} A_k^{\mu_k}, pk_O\right). \quad (5)$$

If the equation is established, the TPA sends 1 to the DO, which means the CSSs have stored replicas correctly as the DO required. Otherwise, the TPA sends 0 to the DO.

4.2.3 Update Stage

For each update, the DO sends an update request req to the CSSs by running $DynaGen$. Upon receiving req, the CSSs update the stored data and compute the proof P_{dy} as a response. Further, the DO will decide to agree/disagree on the updating operation with the CSSs by running $DynaVerify$. The details of the update stage are illustrated in Fig. 5. In our proposal, three kinds of dynamic operations will be realized via $Modification$, $Insertion$, and $Deletion$. The previous data update constructions only require the DO to verify the updating evidence returned by the CSSs. If the verification is passed, the DO believes that the CSSs have performed the updating operations honestly. As for our construction, both the DO and CSSs are required to agree on each update, which prevents the DO from deliberately framing the CSSs who have performed the update operation honestly.

$DynaGen(name, b_{ij}, \sigma_i)$. When updating data, the DO sends a request $req = \{name, OP, i, \{b^*_{ij}\}_{1 \le j \le m_i}, \sigma^*_i\}$ to the CSSs. We denote OP as M for modification, OP as I for insertion, and OP as D for deletion. b^*_{ij} and σ^*_i represent the new replica values for the operated block and the corresponding tag. The value of b^*_{ij} and σ^*_i is $null$ when OP is D. Finally, the update request is sent to the CSSs.

$DynaVerify(req, params, sk_O, F, name)$. When receiving req, the CSSs interact with the DO as follows.

- $Modification$: Given $F = \{b_i\}_{1 \le i \le n}$, we suppose ith data block b_i is modified as b'_i. The DO creates m_i copy blocks $b'_{ij} = b'_i + H_3(sk_{O2}, name\|j)$, and computes $\sigma'_{ij} = sk_{O1} \cdot (H_4(bid'_{ij}) \cdot g^{\sum_{k=1}^{s} a_k b'_{ijk}})^{sk_{O2}}$ and $\sigma'_i = \prod_{j=1}^{m'_i} \sigma'_{ij}$, where $bid'_{ij} = name\|ID_O\|m_{max}\|j\|h'_i$. Then the DO sends a modification request $req = \{name, M, i, \{b'_{ij}\}_{1 \le i \le m_i}, \sigma'_i\}$ to the CSSs. When receiving req, the CSSs replace $b_{ij} \forall j$ with $b'_{ij} \forall j$, and replace σ_i with σ'_i. Then the CSSs update the MD-MHT and generate the signature $Sig_{R'_C} = H_4(R'_C)^{sk_C}$ on the new root R'_C. Finally, the CSSs respond to the DO with an update proof $P_{dy} = \{h_i, \Omega_i, Sig_{R'_C}, R'_C\}$, where Ω_i represents the verification path of old block b_i. After receiving the evidence, the DO first gets the old root by using $\{h_i, \Omega_i\}$ and compares it with the stored R_O. If the verification is unsuccessful, the algorithm terminates and outputs 0; otherwise the DO checks the validity of $Sig_{R'_C}$ by $e(Sig_{R'_C}, g) = e(H_4(R'_C), pk_C)$. If $Sig_{R'_C}$ is found to be invalid, the algorithm terminates and outputs 0; otherwise, the DO stores the new root $R'_O = R'_C$ and generates a new signature $Sig_{R'_O} = H_4(R'_O)^{sk_{O2}}$, and sends it to the CSSs. the CSSs verify the validity of $Sig_{R'_O}$ by $e(Sig_{R'_O}, g) = e(H_4(R'_C), pk_O)$. If $Sig_{R'_O}$ is found to be invalid, the algorithm terminates and outputs 0. Otherwise the algorithm outputs 1, which means the CSSs believe that the DO has approved the modification operation. Afterwards, the DO sends the root R'_O to the TAE and the TPA, and the CSSs send $Sig_{R'_C}$ to the TAE and the TPA. Then the TAE and the TPA will update $R_{TAE}/R_{TPA} = R'_O$ if $Sig_{R'_C}$ is proved to be valid.
- $Insertion$: Given $F = \{b_i\}_{1 \le i \le n}$, we suppose a block b''_i is inserted after position i. The DO creates m_i copies $b''_{ij} = b''_i + H_3(sk_{O2}, name\|j)$, and computes $\sigma''_{ij} = sk_{O1} \cdot (H_4(bid''_{ij}) \cdot g^{\sum_{k=1}^{s} a_k b''_{ijk}})^{sk_{O2}}$ and $\sigma''_i = \prod_{j=1}^{m''_i} \sigma''_{ij}$, where $bid''_{ij} = name\|ID_O\|m_{max}\|j\|h''_i$. Then the DO sends an insertion request $\{name, I, i, \{b''_{ij}\}_{1 \le i \le m_i}, \sigma''_i\}$ to CSSs. When receiving req from the DO, the CSSs add $b''_{ij} \forall j$ and σ''_i into storage. Then the CSSs update the MD-MHT and generate $Sig_{R'_C}$ on the new root R'_C. Finally, the CSSs respond to the DO with an update proof $P_{dy} = \{\Omega_i, h_i, Sig_{R'_C}, R'_C\}$, where Ω_i represents the verification path of b_i in the old tree. After receiving P_{dy}, the DO first gets the old root by using $\{\Omega_i, h_i\}$ and compares it with the stored R_O. If the verification fails, the algorithm terminates; otherwise the DO continues to check the validity of $Sig_{R'_C}$. An example of inserting block b''_i before b_4 is illustrated in Fig. 6, only the new

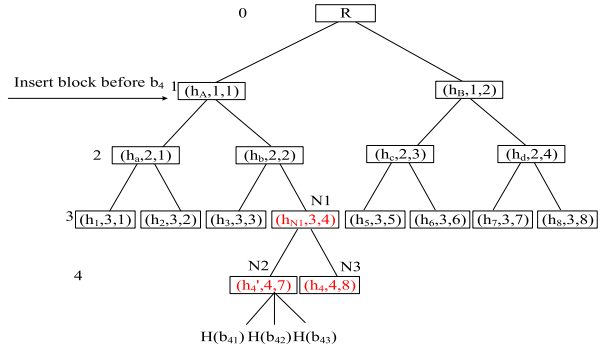


Fig. 6. An example of block insertion in MD-MHT.

node $\{h'_4 = H(H(b_{41})\|H(b_{42})\|H(b_{43})), l'_4 = 4, p'_4 = 2*4-1=7\}$ and an internal node $\{h_{N1} = (H(b'_4)\|h_4\|H(l_{N1}\|p_{N1})), l_{N1} = 3, p_{N1} = 4\}$ are added into the tree and the old node $\{h_4, l_4 = 3, p_4 = 4\}$ is changed to $\{h_4, l_4 = 4, p_4 = 8\}$. The DO further computes the new root R'_O using $\{\Omega_i, h_i, H(b''_i)\}$ and checks $Sig_{R'_C}$ by $e(Sig_{R'_C}, g) = e(H_4(R'_O), pk_C)$. If $Sig_{R'_C}$ is proved to be invalid, the algorithm terminates; otherwise the DO stores the new root R'_O and generates a new signature $Sig_{R'_O} = H_4(R'_O)^{sk_O}$, and sends $Sig_{R'_O}$ to CSSs for storage. The CSSs verify the validity of $Sig_{R'_O}$ by $e(Sig_{R'_O}, g) = e(H_4(R'_O), pk_O)$. If the validation fails, the algorithm terminates and outputs 0; otherwise the algorithm outputs 1, which means the CSSs believe the DO has approved the insertion operation. Afterwards, the DO sends R'_O to the TAE and the TPA, and the CSSs send $Sig_{R'_C}$ to the TAE and the TPA. The TAE and the TPA will update the root $R_{TAE}/R_{TPA} = R'_O$ if $Sig_{R'_C}$ is valid.

- $Deletion$: The deletion request is $\{name, D, i, null, null\}$. Other steps are similar to $Modification$.

4.2.4 Arbitration Stage

When $ProofVerify$ outputs 0 and the CSSs disagree with the auditing result, the CSSs will apply to the TAE to judge the honesty of auditors by running $ArbitrationJud$.

$ArbitrationJud(Sig_{R_C}, R_O, R_{TPA}, R_{TAE})$. The current challenge Q on the TPA's side and the proof P on the CSSs' side are provided to the TAE. If P is found to be invalid, the TAE rejects the CSSs' appeal and the algorithm outputs -1; otherwise the DO and TPA are required to provide R_O and R_{TPA} to the TAE. If $R_O = R_{TAE} \neq R_{TPA}$, the TAE considers the TPA to be dishonest and the algorithm outputs 0. If $R_O \neq R_{TAE} = R_{TPA}$, the TAE considers the DO to be dishonest and the algorithm outputs 1. If $R_O \neq R_{TAE}$ and $R_{TAE} \neq R_{TPA}$ the TAE considers both the DO and the TPA to be dishonest and the algorithm outputs 2. If $R_O = R_{TAE} = R_{TPA}$, the algorithm terminates.

5 Security Analysis

In this section, we provide a provable security analysis of our scheme via the following theorems.

Theorem 1 (Correctness). *If the DO, the CSSs, the KGC, the TPA, and the TAE are honest in obeying the specified procedures, then the proof can pass the TPA's verification.*

Proof. According to the characteristics of bilinear pairings, Equation (5) in the $ProofVerify$ algorithm is proven to be correct according to the deduction from left to right

$$
\begin{aligned}
&e(\sigma, g)\\
&= e\Big(\prod_{i\in Q}\sigma_i^{r_i}, g\Big)\\
&= e\Big(\prod_{i\in Q}\Big(\prod_{j=1}^{m_i}\sigma_{ij}\Big)^{r_i}, g\Big)\\
&= e\Big(\prod_{i\in Q}\Big(\prod_{j=1}^{m_i}(sk_{O1}\cdot(H_4(bid_{ij})\cdot g^{\sum_{k=1}^{s}a_k b_{ijk}})^{sk_{O2}})\Big)^{r_i}, g\Big)\\
&= e\Big(\prod_{i\in Q} sk_{O1}^{r_i}, g\Big)\cdot\\
&\quad e\Big(\prod_{i\in Q}\prod_{j=1}^{m_i} H_4(bid_{ij})^{r_i}\cdot\prod_{i\in Q}\prod_{j=1}^{m_i} g^{\sum_{k=1}^{s}a_k b_{ijk}}, g^{sk_{O2}}\Big)\\
&= e\Big(\prod_{i\in Q} H_1(ID_O)^{r_i}, g^{s}\Big)\cdot\\
&\quad e\Big(\prod_{i\in Q}\prod_{j=1}^{m_i} H_4(bid_{ij})^{r_i}\cdot\prod_{k=1}^{s}\prod_{i\in Q}\prod_{j=1}^{m_i} A_k^{b_{ijk}}, pk_O\Big)\\
&= e\Big(\prod_{i\in Q} H_1(ID_O)^{r_i}, mpk\Big)\cdot\\
&\quad e\Big(\prod_{i\in Q}\prod_{j=1}^{m_i} H_4(bid_{ij})^{r_i}\cdot\prod_{k=1}^{s} A_k^{\sum_{i\in Q}\sum_{j=1}^{m_i} r_i\cdot b_{ijk}}, pk_O\Big)\\
&= e(H_1(ID_O)^{\sum_{i\in Q} r_i}, mpk)\cdot\\
&\quad e\Big(\prod_{i\in Q}\prod_{j=1}^{m_i} H_4(bid_{ij})^{r_i}\cdot\prod_{k=1}^{s} A_k^{\mu_k}, pk_O\Big).
\end{aligned}
$$

□

Theorem 2 (Unforgeability of Tags). *If the probability of an attacker ($\mathcal{A}_1$, $\mathcal{A}_2$) winning the game is negligible, then our MDSS satisfies tag unforgeability under Definition 4.*

Proof. First, we provide validation that if the probability of $\mathcal{A}_1$ winning the game 1 is not negligible, then there exists an extractor $\mathcal{B}$ can calculate g^{ab} through a given instance $(g, G, g^{\alpha}, g^{\beta})$ with the probability of $\mathcal{A}_1$. Therefore, an extractor B can solve the CDH problem. Here B simulates each interaction with $\mathcal{A}_1$ through the following steps.

Setup. $\mathcal{B}$ executes the system initialization to obtain $params$ and sets the master public key $mpk = g^a$.

H_1-*Query*. $\mathcal{A}_1$ adaptively performs H_1-Query for any selected ID'. $\mathcal{B}$ maintains a list $L_{H_1} = (ID, h_1, D, T)$ for the H_1-Query. If the selected ID' belongs to L_{H_1}, $\mathcal{B}$ extracts the corresponding tuple (ID', h_1', D', T') and responds D' to $\mathcal{A}_1$. If not, $\mathcal{B}$ chooses a value $h_1' \in Z_q$ randomly and tosses a coin $T \in \{0, 1\}$. Let's assume that the probability of T taking 0 is γ, then the probability of T taking 1 is $1-\gamma$. When $T = 0$ with the probability of γ, $\mathcal{B}$ computes $D' = g^{h_1'}$. When $T = 1$, $D' = (g^{\beta})^{h_1'}$. Then $\mathcal{B}$ returns D' to $\mathcal{A}_1$ and adds the tuple (ID', h_1', D', T') into L_{H_1}.

Partial-Key-Query. $\mathcal{A}_1$ adaptively runs Partial-Key-Query for any selected ID'. $\mathcal{B}$ maintains a list $L_{ssk} = (ID, ssk_{ID}, x_{ID})$ for the Partial-Key-Query. $\mathcal{B}$ checks whether (ID', h_1', D', T') belongs to L_{H_1}. If not, $\mathcal{B}$ performs H_1-Query for ID'. If $T' = 1$, $\mathcal{B}$ terminates.

1) If ID' is in L_{ssk}, $\mathcal{B}$ checks whether $ssk_{ID'} = \perp$. If $ssk_{ID} = \perp$, $\mathcal{B}$ inquires (ID', h_1', D', T') from L_{H_1}, and computes $ssk_{ID'} = D'^{\alpha} = g^{\alpha h_1'}$ for $T' = 0$, and renews $ssk_{ID'}$. When $T' = 1$, $\mathcal{B}$ terminates. If $ssk_{ID'} \neq \perp$, $\mathcal{B}$ extracts $ssk_{ID'}$ directly. Then $\mathcal{B}$ sends $ssk_{ID'}$ to $\mathcal{A}_1$.
2) If ID' is not in L_{ssk}, $\mathcal{B}$ gets (ID', h_1', D', T') from L_{H_1}, and computes $ssk_{ID'} = D'^{\alpha} = g^{\alpha h_1'}$ when $T' = 0$. When $T' = 1$, $\mathcal{B}$ terminates. Then $\mathcal{B}$ sends $ssk_{ID'}$ to $\mathcal{A}_1$ and adds $(ID', ssk_{ID'}, \perp, \perp)$ into L_{ssk}.

Secret-Value-Query. $\mathcal{A}_1$ adaptively runs Secret-Value-Query for any selected ID'. $\mathcal{B}$ checks whether (ID', h_1', D', T') belongs to L_{H_1}. If not, $\mathcal{B}$ performs H_1-Query for ID'. Next, $\mathcal{B}$ checks whether ID' is in L_{ssk}.

1) If ID' is in L_{ssk}, $\mathcal{B}$ checks whether $x_{ID'}$ equals to $\perp$. If $x_{ID'} = \perp$, $\mathcal{B}$ chooses a random value $x \in Z_q$, and sets $x_{ID'} = x$. Then $\mathcal{B}$ renews $x_{ID'}$. When $T' = 1$, $\mathcal{B}$ terminates. If $x_{ID'} \neq \perp$, $\mathcal{B}$ obtains $x_{ID'}$. Then $\mathcal{B}$ sends $x_{ID'}$ to $\mathcal{A}_1$.
2) If ID' is not in L_{ssk}, $\mathcal{B}$ chooses $x \in Z_q$ randomly, and computes $x_{ID'} = x$. Then $\mathcal{B}$ sends $x_{ID'}$ to $\mathcal{A}_1$ and adds $(ID', \perp, x)$ into L_{ssk}.

H_2-*Query*. $\mathcal{A}_1$ adaptively performs H_2-Query for any selected ID'. $\mathcal{B}$ maintains a list $L_{H_2} = (ID, h_2, ssk_{ID}, sk_{ID2}, pk_{ID})$ for the H_2-Query. $\mathcal{B}$ checks whether ID' is in L_{H_1}. If ID' is in L_{H_1}, $\mathcal{B}$ extracts the tuple. If not, $\mathcal{B}$ performs H_1-Query for ID'. Next, $\mathcal{B}$ checks whether ID' is in L_{ssk}. If ID' is in L_{ssk}, $\mathcal{B}$ searches the tuple $(ID', ssk_{ID'}, x_{ID'})$ from L_{ssk}. If not, $\mathcal{B}$ performs H_2-Query for ID'.

1) $\mathcal{B}$ checks whether ID' exists in L_{H_2}. If it does, $\mathcal{B}$ sets $sk_{ID'2} = x_{ID'} + {h_2}' \bmod q$, $pk_{ID'} = g^{sk_{ID'2}}$, updates $sk_{ID'2}$ and $pk_{ID'}$, and sends $sk_{ID'2}$ to $\mathcal{A}_1$. If not, $\mathcal{B}$ selects a random value ${h_2}' \in Z_q$, sets $sk_{ID'2} = x_{ID'} + {h_2}' \bmod q$, $pk_{ID'} = g^{sk_{ID'2}}$, adds $(ID', {h_2}', ssk_{ID'}, sk_{ID'2}, pk_{ID'})$ into L_{H_2}, and sends $sk_{ID'2}$ to $\mathcal{A}_1$ for response.

Public-Key-Query. $\mathcal{A}_1$ adaptively runs Public-Key-Query for any selected ID'.

1) If $(ID', {h_2}', ssk_{ID'}, sk_{ID'2}, pk_{ID'})$ is in L_{H_2}, $\mathcal{B}$ checks whether $pk_{ID'}$ equals to $\perp$. If $pk_{ID'} = \perp$, $\mathcal{B}$ chooses a random value $x \in Z_q$, and sets $x_{ID'} = x$, $sk_{ID'2} = x_{ID'} + {h_2}' \bmod q$, $pk_{ID'} = g^{sk_{ID'2}}$. Then $\mathcal{B}$ renews $\{x_{ID'}, sk_{ID'2}, pk_{ID'}\}$ and sends $pk_{ID'}$ to $\mathcal{A}_1$. If $pk_{ID'} \neq \perp$, $\mathcal{B}$ obtains $pk_{ID'}$ directly. Then $\mathcal{B}$ sends $pk_{ID'}$ to $\mathcal{A}_1$.
2) If $(ID', {h_2}', ssk_{ID'}, sk_{ID'2}, pk_{ID'})$ is not in L_{H_2}, $\mathcal{B}$ chooses $x \in Z_q$ randomly, and computes $x_{ID'} = x, sk_{ID'2} = x_{ID'} + {h_2}' \bmod q$, $pk_{ID'} = g^{sk_{ID'2}}$. Then $\mathcal{B}$ sends $pk_{ID'}$ to $\mathcal{A}_1$ and adds $(ID', {h_2}', \perp, sk_{ID'2}, pk_{ID'})$ into L_{H_2}.

Public-Key-Replace. $\mathcal{A}_1$ adaptively runs Public-Key-Replace with selected $(ID', pk^*_{ID'})$.

1) If $(ID', {h_2}', ssk_{ID'}, sk_{ID'2}, pk_{ID'})$ is contained in L_{H_2}, $\mathcal{B}$ replaces $(ID', {h_2}', ssk_{ID'}, sk_{ID'2}, pk_{ID'})$ with $(ID', {h_2}', ssk_{ID'}, \perp, pk^*_{ID'})$.
2) If $(ID', {h_2}', ssk_{ID'}, sk_{ID'2}, pk_{ID'})$ is not contained in L_{H_2}, $\mathcal{B}$ adds $(ID', \perp, ssk_{ID'}, \perp, pk^*_{ID'})$.

H_3-*Query*. $\mathcal{A}_1$ adaptively performs H_3-Query for any selected $b' \in Z_q$. $\mathcal{B}$ maintains a list $L_{H_3} = (b, \{h_{3,i}\}_{1 \le i \le m_b}, \{b_i\}_{1 \le i \le m_b}, M)$ for the H_2-Query, where b denotes a raw block with replica number m_b. If L_{H_3} contains b', $\mathcal{B}$ sends $\{h'_{3,i}\}_{1 \le i \le m_b}$ to $\mathcal{A}_1$. If not, $\mathcal{B}$ randomly chooses m_b values $h'_{3,i} \in Z_q$ and sends them to $\mathcal{A}_1$.

Copy-Query. $\mathcal{A}_1$ adaptively runs Copy-Query with $((b', ID')$. $\mathcal{B}$ checks whether $T' = 0$ in L_{H_1} for ID'. If $T' = 1$, $\mathcal{B}$ terminates. Otherwise, $\mathcal{B}$ gets $h'_{3,i} \in Z_q$ from L_{H_3}, computes $b'_i = b' + h'_{3,i}$, divides b'_i into $\{b'_{ik}\}_{1 \le k \le s}$ renews b'_{ik} in L_{H_3} and sends $\{b'_{ik}\}_{1 \le i \le m_b}$ to $\mathcal{A}_1$.

H_4-*Query*. $\mathcal{A}_1$ adaptively runs H_4-Query for b'. $\mathcal{B}$ also maintains a list $L_{H_4} = (bid, h_4)$ for the H_4-Query. If L_{H_4} contains bid', $\mathcal{B}$ sends $g^{h'_4}$ to $\mathcal{A}_1$. If not, $\mathcal{B}$ randomly chooses a value $h'_4 \in Z_q$ and sends $g^{h'_4}$ to $\mathcal{A}_1$, then $\mathcal{B}$ adds (bid', h'_4) into L_{H_4}.

Tag-Query. $\mathcal{A}_1$ runs Tag-Query with (bid', b', ID'). $\mathcal{B}$ checks whether $T' = 0$ in L_{H_1} for ID'. If $T' = 1$, $\mathcal{B}$ terminates. Otherwise, $\mathcal{B}$ gets $H_4(bid')$ from L_{H_4}, $ssk_{ID'}$ from L_{H_1}, $sk_{ID'2}$ from L_{H_2}, and $\{b_{ik}\}_{1 \le i \le m_b}$ from L_{H_3}. Then $\mathcal{B}$ generates the tag for (bid', b', ID') by performing $TagGen$ and sends the tag value to $\mathcal{A}_1$.

Forge. $\mathcal{A}_1$ outputs a tuple $O = (\sigma^*, bid^*, b^*, ID^*, pk_{ID^*})$, where σ^* is the forged tag of block b^* divided into $\{b^*_k\}_{1 \le k \le s}$ with the identity ID^* and the public key pk_{ID^*}.

Analysis. When $\mathcal{A}_1$ wins game 1, $\mathcal{B}$ can obtain $e(\sigma^*, g) = e(H_1(ID_O), mpk) \cdot e(H_4(bid^*) \cdot \prod_{k=1}^{s} A_k^{b^*_k}, pk_{ID^*})$. $\mathcal{B}$ extracts the tuple (ID^*, h^*_1, D^*, T^*) from L_{H_1}. If $T^* = 1$, $\mathcal{B}$ terminates. Then $\mathcal{B}$ extracts $H_1(ID^*) = g^{\beta {h_1}^*}$ from L_1 and $H_4(bid^*) = g^{h'_4}$ from L_{H_4}. According to the validation equation, $\mathcal{B}$ can obtain $e(\sigma^*, g) = e(g^{\beta {h_1}^*}, g^{\alpha}) \cdot e(g^{h'_4} \cdot \prod_{k=1}^{s} A_k^{b^*_k}, pk_{ID^*})$. Then $\mathcal{B}$ can derive $g^{\alpha\beta} = (\frac{\sigma^*}{\prod_{k=1}^{s} A_k^{b^*_k} \cdot pk_{ID^*}{}^{h'_4}})^{\frac{1}{{h_1}^*}}$. We evaluate the probability of $\mathcal{B}$ getting the right result. Here we denote that $\mathcal{A}_1$ wins game 1 at the advantage ϵ within time t after querying H_1-Query, Partial-Key-Query, Secret-Value-Query, H_2-Query, Public-Key-Query, Public-Key-Replace, H_3-Query, Copy-Query, H_4-Query, and Tag-Query for up to q_1, q_{ssk}, q_{sv}, q_2, q_{pk}, q_{pkr}, q_3, q_c, q_4, q_t times respectively. If $\mathcal{B}$ and $\mathcal{A}_1$ interact perfectly, that is, $\mathcal{B}$ does not terminate the algorithm in the query process, then H_1-Query, Secret-Value-Query, H_2-Query, Public-Key-Query, Public-Key-Replace, H_3-Query, and H_4-Query are executed successfully without needing additional requirements. $\mathcal{B}$ may terminate the algorithm in Partial-Key-Query, Copy-Query and Tag-Query, and the probability of $\mathcal{B}$ and $\mathcal{A}_1$ interacting perfectly is greater than $(1-\gamma)^{q_{ssk}q_cq_t}$. Thus the probability of $\mathcal{B}$ getting the right result is $\epsilon^* \ge \epsilon\gamma(1-\gamma)^{q_{ssk}q_cq_t} \ge \frac{\epsilon}{(q_{ssk}+q_c+q_t)2e}$. $\mathcal{B}$ can therefore solve the CDH problem with the probability $\epsilon^* \ge \frac{\epsilon}{(q_{ssk}+q_c+q_t)2e}$ in time $t^* \le t + O(q_1 + q_{ssk} + q_{sv} + q_2 + q_{pk} + q_{pkr} + q_3 + q_c + q_4 + q_t)$.

We now provide validation for a case in which the probability of $\mathcal{A}_2$ winning game 2 is not negligible. Here an extractor $\mathcal{B}$ can calculate g^{ab} through a given instance $(g, G, g^{\alpha}, g^{\beta})$ with the probability of $\mathcal{A}_2$. Therefore B can solve the CDH problem defined in Definition 2. B simulates each interaction with $\mathcal{A}_2$ through the following steps.

Setup. $\mathcal{B}$ chooses a value $s \in Z_q$ randomly as msk and generates $params$. $\mathcal{B}$ sends msk and $params$ to $\mathcal{A}_2$.

H_1-*Query*. $\mathcal{A}_2$ adaptively performs H_1-Query for any selected ID'. B maintains a list $L_{H_1} = (ID, h_1)$ for the H_1-Query. If the selected ID' belongs to L_{H_1}, $\mathcal{B}$ extracts the tuple $(ID', {h_1}')$ and responds $g^{{h_1}'}$ to $\mathcal{A}_1$. If not, $\mathcal{B}$ chooses a value ${h_1}' \in Z_q$ randomly and returns $g^{{h_1}'}$ to $\mathcal{A}_2$ and adds the tuple $(ID', {h_1}')$ into L_{H_1}.

H_2-*Query*. $\mathcal{A}_1$ adaptively performs H_2-Query for any selected ID'. $\mathcal{B}$ maintains a list $L_{H_2} = (ID, h_2)$ for the H_2-Query. If ID' belongs to L_{H_1}, $\mathcal{B}$ searches the tuple $(ID', {h_1}')$. If not, $\mathcal{B}$ performs H_1-Query for ID'. Next, $\mathcal{B}$ checks whether ID' is in L_{H_2}. If ID' is in L_{H_2}, $\mathcal{B}$ searches the tuple $(ID', {h_2}')$ from L_{H_2}. If not, $\mathcal{B}$ chooses a value ${h_2}' \in Z_q$ randomly and returns ${h_2}'$ to $\mathcal{A}_2$ and adds the tuple $(ID', {h_2}')$ into L_{H_2}.

Secret-Value-Query. B does not run Partial-Key-Query, as it already possesses the master key. $\mathcal{A}_2$ adaptively runs Secret-Value-Query for any selected ID'. $\mathcal{B}$ maintains a list $L_{sv} = (ID, x_{ID}, pk_{ID}, T)$ for the Secret-Value-Query. If

ID' belongs to L_{H_2}. If not, $\mathcal{B}$ performs H_2-Query for ID'. $\mathcal{B}$ checks whether ID' belongs to L_{sv}.

1) If not, $\mathcal{B}$ chooses a value $x' \in Z_q$ randomly and tosses a coin $T' \in \{0,1\}$. Let's assume that the probability of T' taking 0 is γ; then the probability of T' taking 1 is $1-\gamma$. When $T'=0$ with the probability of γ, $\mathcal{B}$ computes $pk_{ID'} = g^{x+h_2{}' \bmod q}$. When $T=1$, $pk_{ID'} = (g^{\alpha})^{x+h_2{}' \bmod q}$. Then $\mathcal{B}$ returns $x_{ID'}$ to $\mathcal{A}_2$ and adds the tuple $(ID', x_{ID}{}', pk_{ID'}, T')$ into L_{sv}.
2) Otherwise, $\mathcal{B}$ checks T'. When $T'=1$, $\mathcal{B}$ terminates. When $T'=0$, $\mathcal{B}$ extracts $x_{ID}{}'$ and returns it to $\mathcal{A}_2$.

Public-Key-Query. $\mathcal{A}_1$ adaptively runs Public-Key-Query for any selected ID'.

1) If $(ID', x_{ID'}, pk_{ID'}, T')$ is in L_{sv}, $\mathcal{B}$ sends $pk_{ID'}$ to $\mathcal{A}_2$.
2) If $(ID', x_{ID'}, pk_{ID'}, T')$ is not in L_{sv}, $\mathcal{B}$ chooses $x' \in Z_q$ randomly, and tosses a coin $T' \in \{0,1\}$. Let's assume that the probability of T' taking 0 is γ, then the probability of T' taking 1 is $1-\gamma$. When $T'=0$ with the probability of γ, $\mathcal{B}$ computes $pk_{ID'} = g^{x+h_2{}' \bmod q}$. When $T=1$, $pk_{ID'} = (g^{\alpha})^{x+h_2{}' \bmod q}$. Then $\mathcal{B}$ returns $pk_{ID'}$ to $\mathcal{A}_2$ and adds the tuple $(ID', x_{ID}{}', pk_{ID'}, T')$ into L_{sv}.

H_3-*Query*. $\mathcal{A}_1$ adaptively performs H_3-Query for any selected $b' \in Z_q$. $\mathcal{B}$ maintains a list $L_{H_3} = (b, \{h_{3,i}\}_{1\le i\le m_b}, \{b_i\}_{1\le i\le m_b}, M)$ for the H_3-Query. If L_{H_3} contains b', $\mathcal{B}$ sends $\{h_{3,i}\}_{1\le i\le m_b}$ to $\mathcal{A}_2$. If not, $\mathcal{B}$ randomly chooses m_b values $h'_{3,i} \in Z_q$ and sends them to $\mathcal{A}_2$.

Copy-Query. $\mathcal{A}_1$ adaptively runs Copy-Query with (b', ID'). $\mathcal{B}$ checks whether $T'=0$ in L_{H_1} for ID'. If $T'=1$, $\mathcal{B}$ terminates. Otherwise, $\mathcal{B}$ gets $h'_{3,i} \in Z_q$ from L_{H_3}, computes $b'_i = b' + h'_{3,i}$, divides b'_i into $\{b'_{ik}\}_{1\le k\le s}$ renews b'_{ik} in L_{H_3} and sends $\{b'_{ik}\}_{1\le i\le m_b}$ to $\mathcal{A}_2$.

H_4-*Query*. $\mathcal{A}_1$ adaptively runs H_4-Query for b'. $\mathcal{B}$ also maintains a list $L_{H_4} = (bid, h_4)$ for the H_4-Query. If L_{H_4} contains the value bid', $\mathcal{B}$ sends $g^{h'_4}$ to $\mathcal{A}_2$. If not, $\mathcal{B}$ randomly chooses a value $h'_4 \in Z_q$ and sends $g^{h'_4}$ to $\mathcal{A}_2$, then $\mathcal{B}$ adds (bid', h'_4) into L_{H_4}.

Tag-Query. $\mathcal{A}_2$ runs Tag-Query with (bid', b', ID'). $\mathcal{B}$ checks whether $T'=0$ in L_{H_1} for ID'. If $T'=1$, $\mathcal{B}$ terminates. Otherwise, $\mathcal{B}$ gets $H_4(bid')$ from L_{H_4}, $sk_{ID'2}$ from L_{H_2}, and $\{b_{ik}\}_{1\le i\le m_b}$ from L_{H_3}. Then $\mathcal{B}$ generates the tag for (bid', b'_k, ID') by performing $TagGen$ and sends the tag value to $\mathcal{A}_2$.

Forge. $\mathcal{A}_1$ outputs a tuple $O = (\sigma^*, bid^*, b^*, ID^*)$, where σ^* is the forged tag of block b^* divided into $\{b^*_k\}_{1\le k\le s}$ with the identity ID^*.

Analysis. When $\mathcal{A}_2$ wins game 2, $\mathcal{B}$ can obtain $e(\sigma^*, g) = e(H_1(ID_O), mpk) \cdot e(H_4(bid^*) \cdot \prod_{k=1}^{s} A_k^{b^*_k}, pk_{ID^*})$. $\mathcal{B}$ extracts the tuple (ID^*, h_1^*) from L_{H_1}. If $T^*=0$, $\mathcal{B}$ terminates. Otherwise $\mathcal{B}$ extracts $H_1(ID^*) = g^{h_1{}^*}$ from L_{H_1}, $H_4(bid^*) = g^{\beta h_4^*}$ from L_{H_4} and $pk_{ID^*} = (g^{\alpha})^{x+h_2{}^* \bmod q}$ from L_{sv}. According to the validation equation, $\mathcal{B}$ can obtain $e(\sigma^*, g) = e(g^{h_1{}^*}, g^s) \cdot e(g^{\beta h_4{}^*} \prod_{k=1}^{s} A_k^{b^*_k}, g^{\alpha(x+h_2{}^* \bmod q)})$. Then $\mathcal{B}$ can derive $g^{\alpha\beta} = \frac{1}{(\sigma^*)^{sh_1{}^* h_4{}^* \prod_{k=1}^{s} A_k^{b^*_k (x+h_2{}^* \bmod q)}}}$. Here we evaluate the probability of $\mathcal{B}$ getting the right result. We denote that $\mathcal{A}_1$ wins game 2 at the advantage ϵ within time t after querying H_1-Query, Secret-Value-Query, H_2-Query, Public-Key-Query, H_3-Query, Copy-Query, H_4-Query, and Tag-Query for up to q_1, q_{sv}, q_2, q_{pk}, q_3, q_c, q_4, q_t times respectively. Then Secret-Value-Query, H_2-Query, Public-Key-Query, H_3-Query, Copy-Query, and H_4-Query are executed successfully without needing additional requirements. $\mathcal{B}$ may terminate the algorithm in H_1-Query, Copy-Query and Tag-Query, and the probability of $\mathcal{B}$ and $\mathcal{A}_2$ interacting perfectly is greater than $(1-\gamma)^{q_1 q_c q_t}$. Thus the probability of $\mathcal{B}$ getting the right result is $\epsilon^* \ge \epsilon\gamma(1-\gamma)^{q_1 q_c q_t} \ge \frac{\epsilon}{(q_1+q_c+q_t)2e}$. Thus $\mathcal{B}$ can solve the CDH problem with the probability $\epsilon^* \ge \frac{\epsilon}{(q_1+q_c+q_t)2e}$ in time $t^* \le t + O(q_1 + q_{sv} + q_2 + q_{pk} + q_3 + q_c + q_4 + q_t)$. □

Theorem 3 (Unforgeability of Proof). *If the probability of an attacker ($\mathcal{A}_3$) winning game 3 is negligible without possessing the correct data, then our MDSS scheme satisfies the unforgeability of proof under Definition 5.*

Proof. To make the Equation (5) hold, the CSSs should generate a valid proof $P = \{\sigma, u\}$ based on the undamaged data. However, the CSSs must generate a proof $P' = \{\sigma', u'\}$ based on the corrupted data to win game 3. If the Equation (5) is validated by using the proof, then the CSSs win the game; otherwise, they lose. Since a valid proof can pass the verification, we can get the following formula:

$$e(\sigma, g) = e(H_1(ID_O)^{\sum_{i\in Q} r_i}, mpk) \cdot e\left(\prod_{i\in Q}\prod_{j=1}^{m_i} H_4(bid_{ij})^{r_i} \cdot \prod_{k=1}^{s} A_k^{\mu_k}, pk_O\right).$$

Supposing the CSSs win the game, we also can get the following formula:

$$e(\sigma', g) = e(H_1(ID_O)^{\sum_{i\in Q} r_i}, mpk) \cdot e\left(\prod_{i\in Q}\prod_{j=1}^{m_i} H_4(bid_{ij})^{r_i} \cdot \prod_{k=1}^{s} A_k^{\mu'_k}, pk_O\right).$$

Then, we denote $\Delta\mu_k = \mu_k - \mu'_k$, where $\sigma = \sigma'$. With the properties of bilinear pairings, we derive $\sum_{k=1}^{s} a_k \Delta\mu' = 0$. If we let G be a multiplicative cyclic group of a large prime q, and we suppose there are λ different $\Delta\mu_k = \mu_k - \mu'_k$, where $1 \le \lambda \le s$. We know that the tuple $(a_1, a_2, \ldots, a_s)$ are randomly chosen and kept secret from the CSSs. Thus the probability of making $\sum_{i\in Q} \Delta\mu\beta$ hold is less than $q^{\lambda-1}/q^s$, where $q^{\lambda-1}/q^s \le q^{\lambda-1}/q^{\lambda} = 1/q$. As q is selected as a large prime, $1/q$ is negligible. Therefore, the CSSs cannot pass the data integrity verification by forging a integrity auditing proof without storing the correct data. □

6 Performance Analysis

In this section, we will demonstrate the efficiency of our MDSS by comparing it with other the-state-of-art schemes, namely TB-DMCPDP [19], MuR-DPR [20], and ID-MRPDP [23]. To our knowledge, TB-DMCPDP and MuR-DPR are the two classic BLS-based schemes to provide multi-replica dynamics, saving more costs than RSA-based multi-replica

TABLE 2
Computational Overhead Comparison

	TB-DMCPDP [19]	MuR-DPR [20]	ID-MRPDP [23]	MDSS
Tag-Gen(DO)	$mn(s+1)Exp + n(ms+m-1)Mul$	$2mnsExp + mnsMul$	$(2mns+1)Exp + mnsMul$	$2mnExp + (2mn+m-1)Mul$
Proof-Gen(CSS)	$cExp + (c-1)Mul$	$mcExp + m(c-1)Mul$	$Pair + (c+2)Exp + cMul$	$cExp + m(c-1)Mul$
Proof-Verify(TPA)	$2Pair + (mc+s)Exp$ $+(mc+s-1)Mul$	$2mPair + m(c+1)Exp$ $+mcMul$	$3Pair + (mc+4)Exp$ $+(mc+1)Mul$	$3Pair + (mc+s+1)Exp$ $+(mc+s)Mul$
Total	$T_{CSS} + T_{TPA}$	$T_{CSS} + T_{TPA}$	$T_{CSS} + T_{TPA}$	$T_{CSS} + T_{TPA}$

TABLE 3
Communication Cost Comparison

		TB-DMCPDP [19]	MuR-DPR [20]	ID-MRPDP [23]	MDSS
Proof stage	The Challenge	$\|q\| + N$	$\|q\| + N$	$\|q\| + N$	$\|q\| + N$
	The proof	$\|p\| + ms\|q\|$	$m(\|p\| + \|q\|)$	$3\|p\| + \|q\|$	$\|p\| + s\|q\|$
	Total	$\|p\| + (ms+1)\|q\| + N$	$m\|p\| + (m+1)\|q\| + N$	$3\|p\| + 2\|q\| + N$	$\|p\| + (s+1)\|q\| + N$
Update stage	Update proof	$mw(N+1)\|p\| + \|p\|$	$w(N\lceil\log_2 m\rceil + 1)(\|p\| + 2N + 1) + 2\|p\|$	$-$	$wN(\|p\| + 2N) + 2\|p\|$

schemes. ID-MRPDP is a quite advanced proposal designed with ID-based signatures. Thus we select the above three proposals as comparison options to highlight the efficiency of our MDSS from different aspects.

6.1 Theoretical Analysis

We analyze the computation and communication cost of our scheme in tag generation of setup stage, proof stage, and update stage, for they are the most resource-consuming phases in our proposal. Some definitions used for theoretical analysis are given in Table 1. In the table, $Pair$ denotes the time cost of performing one bilinear pairing operation, Mul denotes the time cost of running one multiplication operation in G_1, and Exp denotes the time cost of executing one exponentiation operation in G_1. Other operations, such as addition and multiplication operations in Z_q and hash, are neglected here because their overheads are nearly negligible. In our assessment, the experiment file F is divided into n blocks and m replicas are generated for the file (mn blocks in total). Each block is further segmented into s sectors. The TPA challenges c blocks for each challenge, and the DO updates w data blocks for each updating. $|p|$ represents the size of an element in G_1/G_2 and $|q|$ represents the size of an element in Z_q. Note that both TB-DMCPDP and our MDSS take advantage of the fragment structure [41] to maximize the storage efficiency and audit performance, while MuR-DPR and ID-MRPDP only divide the file into blocks. When the same file F is segmented into n blocks and each block is divided into s sectors by employing TB-DMCPDP and our MDSS, there will be $n \times s$ blocks by applying MuR-DPR and MuR-DPR. In this case, our MDSS only consists of n block-tag pairs rather than $n \times s$ block-tag pairs in the settings without employing the fragment structure. In other words, the overhead for storing and transferring signature tags will decrease with the increase of s for s sectors making up each block only corresponds to one tag. Thus, this structure can reduce the additional storage space and communication overhead for tags, improving the audit performance. In addition, the unit measurement unit of communication cost is $bits$, and the unit of measurement of computation cost is seconds. For readability, we omit them in subsequent analysis.

6.1.1 Computational Overhead

We evaluate the computational cost of our scheme against three other schemes in tag generation of the setup stage, proof generation and proof verification of the proof stage, for they are the most resource-consuming phases. Table 2 shows the comparison of the four proposals.

In the tag generation, we can derive that the computation overhead in MDSS is independent of the sector number, while ID-MRPDP, TB-DMCPDP and MuR-DPR have increased linearly with the count of sectors. Moreover, we can see that our MDSS costs $2mnExp + (2mn + m - 1)Mul$ for computing all tags for the DO's file.

In the proof stage, the computation overhead of the four proposals are mainly affected by the replica count and the number of challenged blocks. In our MDSS, the CSSs are required to cost $cExp + m(c-1)Mul$ to generate the auditing proof and the TPA needs to cost $3Pair + (mc + s + 1)Exp + (mc + s)Mul$ for verifying the proof.

6.1.2 Communication Cost

The communication cost of integrity schemes consists of several parts: transmitting a random challenge and relevant integrity proof in the proof stage, and transmitting the update proof in the update stage. In the data update phase, the size of the update request is ignored here, for it accounts for a small proportion of the whole overhead. Here we give the comparison of communication overhead as shown in Table 3, where N represents $\lceil \log_2 n \rceil$ for readability.

In the proof stage, the random challenges in four schemes have the same size, and the proof size in MDSS is $(|p| + s|q|)$, which is not affected by the replica number. In short, the total communication cost of our MDSS in the proof stage is $|p| + (s+1)|q| + N$.

For the update stage, the size of the update proof P_{dy} in our MDSS is $w\lceil \log_2 n \rceil(|p| + 2\lceil \log_2 n \rceil) + 2|p|$, which is independent of the number of replicas, while the communication cost of TB-DMCPDP and MuR-DPR increases with the growth of the replica number.

6.1.3 Storage Cost

In our scheme, the storage overhead is the extra space for storing some information except for the copies of outsourcing files, in which the tag storage overhead accounts for an important proportion. For a data file divided into n blocks with m copies, our MDSS is required to produce n aggregation tags, where each tag is an element in G_1. Then we can obtain that the storage cost for tag storing is $n|G_1|$, which has

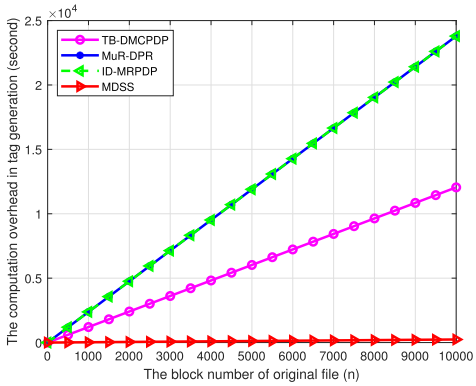


Fig. 7. Computation overhead in tag generation (m=5,s=100).

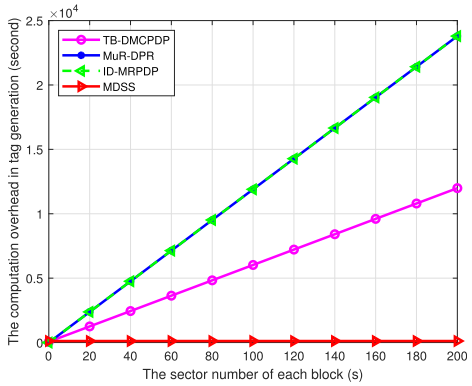


Fig. 9. Computation overhead in tag generation (m=5,n=5000).

nothing to do with the number of replicas or sectors. Here we illustrate that the storage overhead is relatively small compared with that used to store the uploaded copies. Without losing generality, we select the security level as 80-bit. For a $16MB$ file with five replicas, we can divide it into 8389 blocks and each block is segmented into 100 sectors, we choose the elliptic curve group with $|q| = 512$ bits. Under the conditions, our proposal costs 8389×512 bits ≈ 525 KB to store all tags into the cloud, where $525 \text{ KB}/(16 \text{ MB} \times 5) \approx 0.65\%$ is relatively small compared to the size of the outsourced file copies.

6.2 Experimental Result

To performed our experiments, we have implemented MDSS and related comparison schemes by using the Pair Based Cryptography (PBC) library [42]. In the implementation, we conduct CSS computations on Alibaba Elastic Compute Cloud (ECS. G5. xlarge) with 16 GB RAM while TPA computations are executed on a laptop with dual Intel Core CPU running at 2.40 GHz with 8 GB RAM, and DO computations are simulated on a laptop with dual Intel Core CPU running Ubuntukylin-15.10-desktop-i386 with an Intel 2.4 GHz CPU and 4 GB memory. Without loss of generality, we choose the elliptic curve group of Type A defined over 512-bit base field with 160-bit group order. Note that all the experiments are carried out for 100 rounds for pursuing more precise results.

6.2.1 *Computational Overhead*

In this section, we evaluate the computational overhead of all four schemes for tag generation, and the total cost of proof generation and verification in the proof stage.

We first measure the computation overhead for tag generation under different parameters, such as block number n, replica number m, and sector number s. Without loss of generality, we increase the block count from 0 to 10000 at intervals of 500 when creating 5 copies for each block divided into 100 sectors. As illustrated in Fig. 7, the computation overhead of all proposals increases linearly with n. To generate tags for 10000 blocks copied into 5 replicas and segmented into 100 sectors, it only needs about 119 seconds by employing our MDSS. When creating 5000 blocks that contain 100 sectors for each, the comparison result is illustrated in Fig. 8. As observed, the computational cost of our MDSS is much smaller than that of the other schemes. Besides, we create 5 copies for 5000 blocks that contain 0 to 200 sectors, and the comparison result is shown in Fig. 9. It can be concluded that our MDSS always has better efficiency with respect to tag generation, and the computation cost is not affected by the sector number.

Then we test how m and c can influence the computational cost for proof stage. For comparison, we used 5000 blocks of 160 bits each. The experimental results are depicted in Figs. 10 and 11. For proof generation and verification, the other three schemes have the same cost except MuR-DPR since all copies need to be verified one by one in MuR-DPR. Our MDSS, TB-DMCPDP and ID-MRPDP need to spend about the same amount of pairing and exponent operations of group G_1, so their computation cost in the proof stage is almost the same.

We conclude that our MDSS performs well in tag generation and proof stage, especially since when the sector number increases, the advantage of our MDSS for tag generation is more obvious.

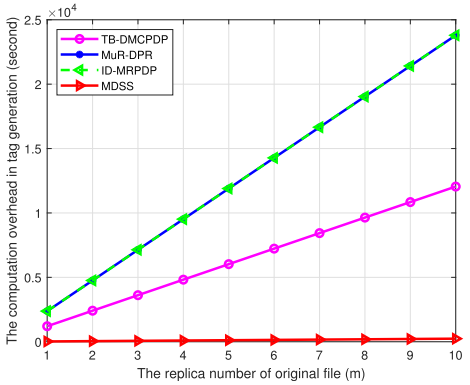


Fig. 8. Computation overhead in tag generation (n=5000,s=100).

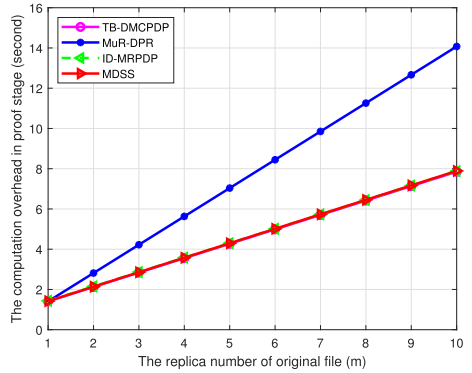


Fig. 10. Computation overhead in proof stage (c=300).

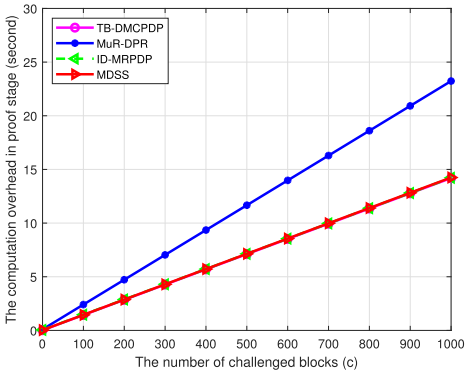


Fig. 11. Computation overhead in proof stage (m=5).

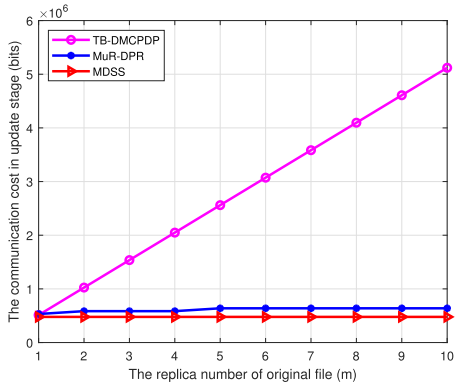


Fig. 14. Communication cost in update stage (w=100,n=5000).

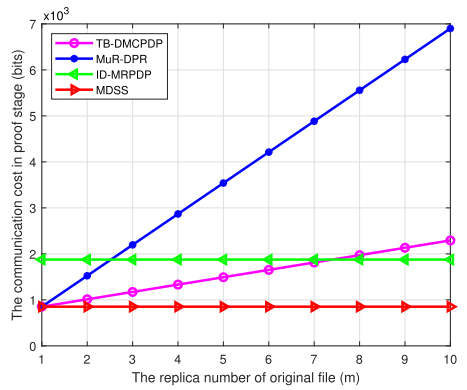


Fig. 12. Communication cost in proof stage (s=1).

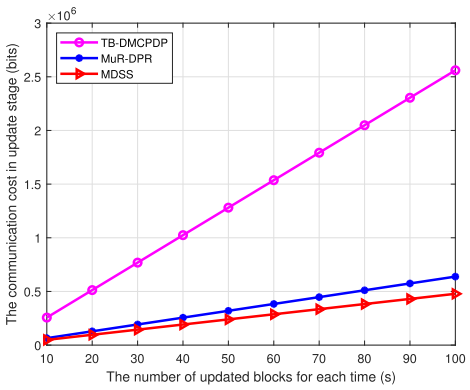


Fig. 15. Communication cost in update stage (m=5,n=5000).

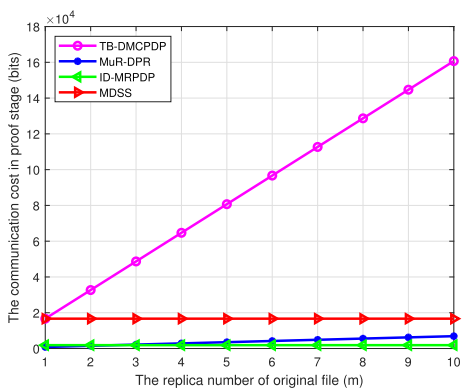


Fig. 13. Communication cost in proof stage (s=100).

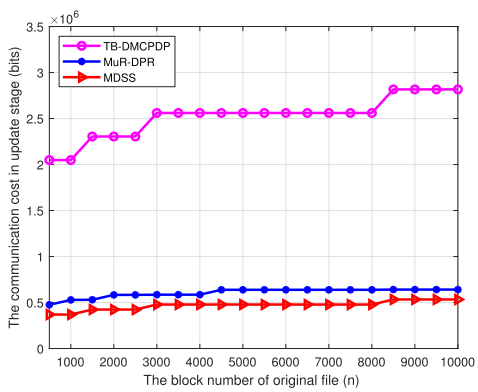


Fig. 16. Communication cost in update stage (m=5,w=100).

6.2.2 *Communication Cost*

In this section, we evaluate the communication cost of the four schemes for transmitting the total information in the proof stage and update stage, respectively.

We first assess the communication overhead in proof phase. The testing file is 16 MB with randomly selected data. When fixing $s = 1$, we vary the count of generated replicas for the testing file from 0 to 10, as illustrated in Fig. 12. Then we divide each block into 100 sectors, and the results are in Fig. 13. Although our MDSS need a little more communication overhead than MuR-DPR and ID-MRPDP, the storing and computing of tags will soar without taking advantage of the fragment structure in MuR-DPR and ID-MRPDP. In a word, the advantage of our MDSS will become obvious when m increases, and our communication cost is only affected by the sector count.

Then, we evaluate the communication overhead for the update stage. Fig. 14 shows the comparison of communication cost between our MDSS and TB-DMCPDP and MuR-DPR with m varies from 1 to 10. Obviously, our scheme has the lowest overhead and does not vary with m. When the number of updated blocks grows from 0 to 100 at the interval of 10 when setting $m = 5$, the comparison of the three schemes is shown in Fig. 15. We can observe that our proposal still has the lowest communication overhead. In addition, the phenomenon that the communication overhead of the three schemes varies with the block number is shown in Fig. 16. Again we conclude that our scheme has the smallest communication overhead.

Whether in proof stage or update stage, our MDSS shows good performance in contrast to other proposals. Moreover, the communication overhead of our scheme is independent of the replica number in both the proof and update stage.

7 Conclusion

In this paper, we propose an effective multi-copy auditing scheme by using certificateless signatures. The inherent characteristics of the certificateless cryptosystem make our system more efficient due to certificate removal. To realize dynamics, we design a dynamic structure to update all replicas through one interaction. Meanwhile, we also provide users with an optional replica number storage strategy, allowing users to decide the storage copy number. Finally, based on the dynamic structure, we employ signature authentication to provide arbitration services between auditors and servers. Security analysis and experimental results confirm that our proposal is provably secure and efficient.

Acknowledgment

This work was supported by National Natural Science Foundation of China (61572255, U1836210, 61702266, 61941116), the Fundamental Research Funds for the Central Universities (30920021129), Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, NJUPT (BDSIP1909), Open Foundation of the State Key Laboratory of Information Security of China (No. 2020-MS-01), Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX19_0343).

References

[1] A. Fu, Z. Chen, Y. Mu, W. Susilo, Y. Sun, and J. Wu, "Cloud-based outsourcing for enabling privacy-preserving large-scale non-negative matrix factorization," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2019.2937484.

[2] H. Wang, D. He, A. Fu, Q. Li, and Q. Wang, "Provable data possession with outsourced data transfer," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2019.2892095.

[3] Y. Yu *et al.*, "Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 4, pp. 767–778, Apr. 2017.

[4] L. Zhou, A. Fu, S. Yu, M. Su, and B. Kuang, "Data integrity verification of the outsourced big data in the cloud environment: A survey," *J. Netw. Comput. Appl.*, vol. 122, pp. 1–15, 2018.

[5] K. He, J. Chen, Q. Yuan, S. Ji, D. He, and R. Du, "Dynamic group-oriented provable data possession in the cloud," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2019.2925800.

[6] A. Fu, S. Li, S. Yu, Y. Zhang, and Y. Sun, "Privacy-preserving composite modular exponentiation outsourcing with optimal checkability in single untrusted cloud server," *J. Netw. Comput. Appl.*, vol. 118, pp. 102–112, 2018.

[7] H. Wang, D. He, J. Yu, and Z. Wang, "Incentive and unconditionally anonymous identity-based public provable data possession," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 824–835, Sep./Oct. 2019.

[8] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 598–609.

[9] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 17, no. 4, pp. 1–29, 2015.

[10] 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.

[11] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Dynamic audit services for integrity verification of outsourced storages in clouds," in *Proc. ACM Symp. Appl. Comput.*, 2011, pp. 1550–1557.

[12] C. Liu *et al.*, "Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2234–2244, Sep. 2014.

[13] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[14] J. Yu, K. Ren, C. Wang, and V. Varadharajan, "Enabling cloud storage auditing with key-exposure resistance," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 6, pp. 1167–1179, Jun. 2015.

[15] J. Yu, K. Ren, and C. Wang, "Enabling cloud storage auditing with verifiable outsourcing of key updates," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1362–1375, Jun. 2016.

[16] Y. Hao, J. Li, J. Han, and Y. Zhang, "A novel efficient remote data possession checking protocol in cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 1, pp. 78–88, Jan. 2017.

[17] H. Yan, J. Li, and Y. Zhang, "Remote data checking with a designated verifier in cloud storage," *IEEE Syst. J.*, vol. 14, no. 2, pp. 1788–1797, Jun. 2020.

[18] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Proc. 28th Int. Conf. Distrib. Comput. Syst.*, 2008, pp. 411–420.

[19] A. F. Barsoum and M. A. Hasan, "On verifying dynamic multiple data copies over cloud servers," *IACR Cryptol. ePrint Archive*, vol. 2011, 2011, Art. no. 447.

[20] C. Liu, R. Ranjan, C. Yang, X. Zhang, L. Wang, and J. Chen, "MuR-DPA: Top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2609–2622, Sep. 2015.

[21] Y. Zhang, J. Ni, X. Tao, Y. Wang, and Y. Yu, "Provable multiple replication data possession with full dynamics for secure cloud storage," *Concurrency Comput.: Practice Experience*, vol. 28, no. 4, pp. 1161–1173, 2016.

[22] S. Peng, F. Zhou, Q. Wang, Z. Xu, and J. Xu, "Identity-based public multi-replica provable data possession," *IEEE Access*, vol. 5, pp. 26 990–27 001, 2017.

[23] J. Li, H. Yan, and Y. Zhang, "Efficient identity-based provable multi-copy data possession in multi-cloud storage," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2929045.

[24] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," *SIAM J. Comput.*, vol. 32, no. 3, pp. 586–615, 2003.

[25] A. Barsoum and A. Hasan, "Enabling dynamic data and indirect mutual trust for cloud computing storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 12, pp. 2375–2385, Dec. 2013.

[26] H. Jin, H. Jiang, and K. Zhou, "Dynamic and public auditing with fair arbitration for cloud data," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 680–693, Third Quarter 2018.

[27] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2003, pp. 452–473.

[28] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. IEEE Symp. Security Privacy*, 1980, pp. 122–122.

[29] M. Sookhak, F. R. Yu, and A. Y. Zomaya, "Auditing big data storage in cloud computing using divide and conquer tables," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 5, pp. 999–1012, May 2018.

[30] A. Fu, S. Yu, Y. Zhang, H. Wang, and C. Huang, "NPP: A new privacy-aware public auditing scheme for cloud data sharing with group users," *IEEE Trans. Big Data*, to be published, doi: 10.1109/TBDATA.2017.2701347.

[31] T. Wu, G. Yang, Y. Mu, F. Guo, and R. Deng, "Privacy-preserving proof of storage for the pay-as-you-go business model," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2019.2931193.

[32] B. Wang, B. Li, and H. Li, "Public auditing for shared data with efficient user revocation in the cloud," *IEEE Trans. Services Comput.*, vol. 8, no. 1, pp. 92–106, Jan./Feb. 2015.

[33] M. Thangavel and P. Varalakshmi, "Enabling ternary hash tree based integrity verification for secure cloud data storage," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2019.2922357.

[34] B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 43–56, First Quarter 2014.

[35] J. Yuan and S. Yu, "Efficient public integrity checking for cloud data sharing with multi-user modification," in *Proc. IEEE INFOCOM*, 2014, pp. 2121–2129.

[36] H. Wang, Z. Wang, and J. Domingo-Ferrer, "Anonymous and secure aggregation scheme in fog-based public cloud computing," *Future Gener. Comput. Syst.*, vol. 78, pp. 712–719, 2018.

[37] Y. Li, Y. Yu, G. Min, W. Susilo, J. Ni, and K. R. Choo, "Fuzzy identity-based data integrity auditing for reliable cloud storage systems," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 1, pp. 72–83, Jan./Feb. 2019.

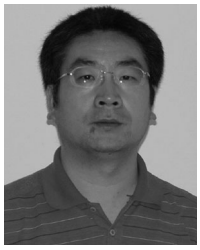
[38] W. Shen, J. Qin, J. Yu, R. Hao, and J. Hu, "Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 2, pp. 331–346, Feb. 2019.
[39] J. Li, H. Yan, and Y. Zhang, "Certificateless public integrity checking of group shared data on cloud storage," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2018.2789893.
[40] Y. Zhang, C. Xu, X. Lin, and X. S. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2908400.
[41] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2008, pp. 90–107.
[42] The pairing-based cryptography (PBC) library. Accessed: Sep. 16, 2016. [Online]. Available: https://crpto.stanford.edu/pbc/download.html

Lei Zhou received the BS degree in network engineering from the Nanjing University of Science and Technology, Nanjing, China, in 2015. She is currently working toward the PhD degree with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. Her research interest include cloud computing security.

Anmin Fu (Member, IEEE) received the PhD degree in information security from Xidian University, Xi'an, China, in 2011. From 2017 to 2018, he was a visiting research fellow with the University of Wollongong, Australia. He is currently a professor at the Nanjing University of Science and Technology, China. His research interests include cloud computing security and applied cryptography. He has published more than 50 technical papers, including international journals and conferences, such as the *IEEE Transactions on Big Data*, *IEEE Transactions on Services Computing*, the *IEEE Transactions on Vehicular Technology*, the *IEEE Internet of Things Journal*, *Information Sciences*, *Computers & Security*, and ACISP.

Guomin Yang (Senior Member, IEEE) received the PhD degree from the Computer Science Department, City University of Hong Kong, Hong Kong, in 2009. He worked as a research scientist with the Temasek Laboratories, National University of Singapore before joining the University of Wollongong (UOW), Australia, in 2012. He is currently an associate professor with the School of Computing and Information Technology, UOW. In 2015, he received the Australian Research Council Discovery Early Career Researcher Award. His research interests include cryptography and network security.

Huaqun Wang received the BS degree in mathematics education from the Shandong Normal University, Jinan, China, in 1997, the MS degree in applied mathematics from the East China Normal University, Shanghai, China, in 2000, and the PhD degree in cryptography from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2006. Currently, he is a professor at the Nanjing University of Posts and Telecommunications. His research interests include applied cryptography, network security, and cloud computing security.



Yuqing Zhang received the PhD degree in cryptography from Xidian University, Xi'an, China. He is currently a professor and the director of the National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, Beijing, China. He has authored or coauthored more than 100 research papers in international journals and conferences, such as the ACM CCS, the *IEEE Transactions on Parallel and Distributed Systems*, and *IEEE Transactions on Dependable and Secure Computing*. His research has been sponsored by NSFC, Huawei, Qihu360, and Google. His current research interests include network and system security and applied cryptography.