Supporting Dynamic Update Scheme for Assured Deletion Based-Multi-Copy Association Tree

Junfeng Tian
Hebei University
Ruxin Bai (rxbai0329@163.com)
Hebei University
Tianfeng Zhang
Hebei University

Research Article

Keywords: Assured deletion, Fog computing, Dynamic update, Blockchain, Public verifiability

Posted Date: April 20th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1548271/v1

Supporting Dynamic Update Scheme for Assured Deletion Based-Multi-Copy Association Tree

Junfeng Tian [1,2], Ruxun Bai [1,2], Tianfeng Zhang [1,2]

1. Hebei University School of Cyber Security and Computer, Baoding 071000, China

2. Hebei Key Laboratory of High Confidence Information Systems (Hebei University), Baoding 071000, China

Corresponding author: Ruxin Bai

E-mail address: rxbai0329@163.com

Telephone number: 008615630109827

Full postal address: Hebei University New Campus, 2666 Qiyi East Road, Lianchi District, Baoding City, Hebei Province, China

Abstract

As one of the most important cloud computing services, cloud storage provides storage resources for resource-constrained users, which reduces their local overhead and computing cost. As an extension of cloud computing, fog computing introduces a fog layer between the cloud and users to deploy computing, storage, and other types of equipment, allowing users to operate outsourced data conveniently. Although cloud storage brings many conveniences to users, assured data deletion is still one of the crucial security challenges. In addition, since cloud servers usually store data in a decentralized manner, it is difficult for users to centrally and securely operate data blocks. This paper proposes an efficient and secure cloud data deletion scheme (SDUS-AD) that supports dynamic data updates in multi-copy scenarios. In this scheme, a new dynamic structure is designed, which improves the traditional Merkle hash tree, thereby realizing the dynamic update of outsourced data efficiently and safely. A cloud-fog-user layer structure is used to meet the needs of resource-constrained users (such as mobile users) to update data, and a secure and trusted fog cluster is constructed using the trusted cloud platform management model based on TPM alliance to ensure the confidentiality of data privacy. Security analysis shows that SDUS-AD meets real-world security requirements. Detailed performance analysis and simulation experiments show that SDUS-AD is efficient, safe, and feasible.

Keywords

Assured deletion; Fog computing; Dynamic update; Blockchain; Public verifiability

1. Introduction

Cloud computing is the fusion, development, and application of the concepts of parallel computing, grid computing, and distributed computing [1, 2]. It can connect large-

scale computing resource pools through the network and provides a series of services for users, such as data sharing service [3, 4], data migration service [5], and data storage service [6, 7]. Through the cloud storage service, individuals and enterprises with limited resources can outsource their sensitive data to the cloud service provider (CSP) [8], thereby enjoying unlimited computing and massive storage resources. However, with the rapid increase in the number of network edge devices and the rapid growth of data volume, centralized cloud service providers have been unable to process the massive data efficiently generated by users. Therefore, fog computing is proposed as an extension of cloud computing [9].

Fog computing is an extension layer between users and the remote CSP, known as the edge network layer. Compared with cloud computing, fog computing is closer to users, and some requests do not need to be processed by the remote CSP but can be processed directly in the fog layer. Fog computing combines cloud services with distributed resources close to the edge of the network, making storage and data processing closer to users, and provides various services such as outsourced computing, resource allocation, and caching [10, 11, 12].

Although cloud storage enables users to reduce storage overhead, it also brings new security problems. Cloud computing is not completely trustworthy, and if the security and privacy issues cannot be properly solved, it will seriously hinder the development of cloud computing [13, 14, 15, 16]. Once the storage and computing tasks are outsourced to the cloud, enterprises or users will lose direct control of the outsourced data. Therefore, ensuring the security of outsourced data is one of the important challenges faced by cloud storage services, such as provable data possession during data storage and assured data deletion during data deletion. The secure deletion of outsourced data has attracted extensive attention in academia and industry.

In cloud storage, the remote CSP performs the deletion of outsourced data. However, the CSP may falsify the results of computations or tamper with data to save computing resources or shorten the response time [17-20]. After users upload the data, there may be some data update requirements, such as data insertion, modification, and deletion. At the same time, the dynamic update of outsourced data may also cause security issues in cloud storage. Traditional schemes require users to re-encrypt and upload the updated data to the CSP, which has a high computational cost. Furthermore, these schemes rely on the CSP to perform dynamic data updates and return update proof for users. However, the CSP is not completely trustworthy, as the cloud may forge update proof to deceive users [21]. In addition, traditional single-copy storage methods may cause data corruption due to equipment failure or human error. The application of multi-copy storage solves the problem of data inaccessibility caused by the corruption of a single data copy. To the best of our knowledge, there is no research work on efficient data deletion schemes that support dynamic updates under multi-copy data. Therefore, it is very meaningful to design a data deletion scheme that can be publicly verifiable and support dynamic updates under multiple copies.

1.1 Our work and contributions

This paper proposes a data deletion scheme that supports the dynamic update, which achieves efficient and secure update operations (such as insertion, deletion, and modification) under multi-copy data. The contributions of this paper are listed as follows:

- In order to realize the fine-grained dynamic update of multi-copy data, a new primitive called multi-copy association tree is proposed, which is mainly used to realize the dynamic update of specific data blocks and their copy data blocks in multi-copy storage (such as insertion, modification, and deletion), and will not affect the normal access of other data. In addition, the specific structure of the MCAT is given, and its security and correctness are analyzed.
- SDUS-AD establishes a cloud-fog-user layer structure and realizes a safe and reliable fog nodes cluster with the help of the trusted cloud platform management model based on TPM alliance [22]. Fog computing is used to solve the problems of insufficient computing power and large communication delay caused by the performance differences of various terminal devices in practical applications, and meets the needs of mobile users for data updates. The user only needs to communicate with local fog to realize data outsourcing, which effectively reduces the local overhead of the user.
- SDUS-AD introduces blockchain technology and builds a blockchain network in the fog layer. The fog provides computing and storage resources for the blockchain. With the help of blockchain technology, the entire process of dynamic data update is recorded to ensure the traceability and verifiability of the system.
- The performance analysis and experimental results show that SDUS-AD is more efficient under the premise of satisfying security.

1.2 Related work

The secure data deletion has received extensive attention from academia and industry for several decades, resulting in many schemes. Existing data deletion schemes are divided into two types: data deletion scheme based on overwriting and data deletion scheme based on key management.

In recent years, researchers have proposed many schemes for data deletion using overwriting technology. Reference [23] proposed using SE commands, block-overwrite programs, and secure destruction of physical devices to achieve secure deletion. Reference [24] designed a new cloud data deletion scheme called “Proof of erasability” (PoE), which uses random patterns to overwrite the disk and returns the same pattern as the deletion proof, and the data owner can verify the data deletion result. Reference [25] proposed a “Proof of Secure Erasure” (PoSE-s) scheme based on reference [24], which aims to erase data in the embedded device. Reference [26] adopted the hourglass function to design a novel data deletion scheme, assuming that the cloud server only maintains the latest version of the data file and all the file backups will be consistent when updated. Therefore, data deletion

can be achieved by overwriting all the file backups with random data. The data owner can verify the deletion result returned by the cloud server through a challenge-response protocol. Reference [27] proposed a provable and traceable data deletion scheme (PTAD). They reference the idea of data integrity checking and construct a consortium chain with Hyperledger Fabric to achieve credible verification and traceable assured deletion.

Since the CSP has control over data in the cloud environment, users should encrypt data to protect data privacy and integrity before the data are outsourced to the cloud. Therefore, researchers have proposed many assured deletion schemes using encryption technology. The secure deletion of cloud data is translated into the secure deletion of the key [28].

Reference [29] first proposed using cryptography technology to achieve secure data deletion. Reference [30] first proposed the assured file deletion system. By deleting the data key of the encrypted file at a specific time, the file cannot be decrypted, thereby achieving assured deletion of the file. Reference [31] proposed to make the data key unrecoverable to achieve secure data deletion. Reference [32] proposed a scheme that achieves verifiable data transfer and deletion, which is the first scheme to the problem of efficient data migration and provable transferred data deletion. Reference [33] proposed a provable data transfer protocol based on provable data possession and deletion for secure cloud storage. Compared with the scheme [32], this scheme can improve the efficiency of the data deletion process. However, there is a security flaw in the scheme. Reference [34] proposed an improved verifiable data transfer scheme, which can solve the problem of verifiable transferred data deletion. Reference [35] proposed a multi-copy data deletion scheme. They outsource data keys encrypted by blind RSA to the third party and propose a multi-copy associated deleting scheme based on pre-deleting sequence and Merkle hash tree. However, the above schemes need to introduce a trusted third party (TTP), but it is difficult to find a completely trusted third party in practical applications. Therefore, the trusted third party would become a bottleneck that impedes the development of the verifiable data deletion system.

To solve the collision problem and system bottleneck problem caused by the trusted third party, reference [36] proposed a new publicly verifiable data deletion scheme, which uses blockchain technology to achieve publicly verifiable data deletion without requiring any trusted third party. Reference [37] proposed a publicly verifiable data transfer and data deletion scheme based on vector commitment. This scheme allows the data owner to migrate outsourced data from the original cloud server to the target cloud server and delete the transferred data from the original server. Reference [38] proposed a new publicly verifiable data deletion scheme. This scheme achieves fine-grained assured deletion of cloud data with the help of the Merkle sum hash tree [39]. Reference [40] proposed a verifiable database scheme for insertion/deletion operations by incorporating vector commitment and designed a new primitive for fine-grained data deletion-Merkle interval hash tree (MIHT). However, the above schemes do not solve the secure deletion of multi-

copy data.

Among the above schemes, only the reference [35] considered the problem of multi-copy association deletion, and only the reference [38] and [40] supported the dynamic operations of outsourced data. There is no scheme to achieve efficient dynamic data update and public verification of deletion results in a multi-copy environment. In addition, these schemes do not consider the problem that users cannot operate data due to limited computing resources. In this paper, we propose such a scheme to solve these problems.

The structure of this article is as follows. Section 2 presents the technical preliminaries required for this paper. Section 3 describes the design goals, the system model, and the adversary model. Section 4 introduces a new primitive called Multi-copy Association Tree, and we give a specific structure and prove its correctness and security. We describe the detailed structure of SDUS-AD in Section 5. The detailed security analysis is given in Section 6. Section 7 is the evaluation of SDUS-AD. It is summarized in Section 8.

2. Related technologies

2.1 Merkle Hash Tree

Merkle hash tree [41] is a binary tree structure that is widely used to check whether the stored elements have changed efficiently. All of its leaf nodes store the hash value of data to be verified, and the internal nodes store the hash value of their child nodes. An MHT is shown in Figure 1, $A = \{a_1, a_2, a_3, a_4\}$ is a data block set to be verified, the leaf node $h(a_1)$, $h(a_2)$, $h(a_3)$, $h(a_4)$ are the hash values of the data block. Each internal node is generated by its two child nodes. For example, $h_1 = h(h(a_1) \| h(a_2))$.

The auxiliary authentication information (AAI) of a leaf node in the MHT is the sibling node of all nodes located on the path from the leaf node to the root. To verify the integrity of the data block a_1, the AAI of $h(a_1)$ is $\Omega = \{h(a_2), h_2\}$, $R' = h(h(a_1) \| h(a_2)) \| h_2)$ can be quickly calculated according to Ω and $h(a_1)$, and then compared with R to determine whether a_1 has changed.

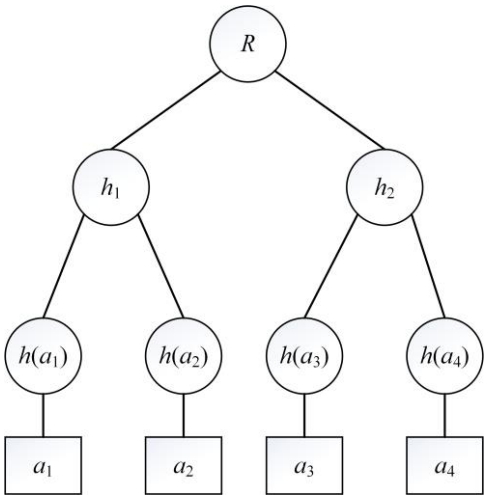

Figure 1. An example of MHT.

2.2 Digital Signature

Digital signature [42] is used to guarantee that the contents of a message have not been altered in transition, and it ensures the non-repudiation of the message by the sender. The digital signature scheme consists of three algorithms: S.KeyGen, S.Sign, and S.Verify. The algorithms are described as follows:

- (SPK, SSK) ← S.KeyGen(1^{κ}). The key generation algorithm takes the security parameter κ as input and outputs a key pair (SPK, SSK).
- σ ← S.Sign(SSK, M). The signature algorithm takes the message M to be signed and the private key SSK as input and outputs the signature σ.
- {1, 0} ← S.Verify(σ, M, SPK). The verification algorithm takes the signature σ, the message M to be verified, and the public key SPK as input and outputs 1 if the signature σ is valid. Otherwise, it outputs 0.

2.3 Hyperledger Fabric

Hyperledger fabric [43] is essentially a distributed shared ledger and database, and the data or information stored in it has the characteristics of unforgeability, traceability, openness, and transparency. Based on these characteristics, it is not feasible for attackers to collude with most nodes in the blockchain to tamper with the ledger.

A smart contract is a computer protocol that disseminates, verifies, or executes contracts in an informative manner and allows for trusted transactions without a third party, which are traceable and unchangeable. Smart contracts can be automatically verified and executed when one or more predefined conditions of the smart contract are triggered.

3. System model and framework

This section describes the design goals, the adversary model, and the system model of SDUS-AD.

3.1 Design goals

The design goals that SDUS-AD wants to achieve are as follows:

Fine-grained deletion: When the user requests to delete data, the CSP should ensure that only the target data is deleted without affecting the user's other data and other users' data.

Timeliness: The operation of data deletion should be timely and promptly, meaning that anyone cannot access the data after the deletion.

The Proof of deletion: The CSP should return a deletion proof to the user after performing a deletion operation. The user can verify the deletion proof to ensure that the data have been deleted.

Traceability: The misbehavior of the user and the CSP can be detected. Neither the user nor the CSP can deny their behaviors and slander the other successfully.

3.2 System model

The system model of SDUS-AD is shown in Figure 2, including the data owner, the fog, the cloud service provider, and Hyperledger fabric.

Data owner (DO): The data owner uploads the data to the cloud service provider to reduce the local storage overhead, but the data owner is not sure whether the cloud service provider performed the deletion operation as required.

Fog: The fog provides the data owner with some computing and storage services, containing two types of nodes. One is the management fog nodes (MFN). With the help of the trusted cloud platform management model based on TPM alliance [22], a secure and reliable fog node cluster is realized, responsible for ensuring data privacy. The other type is service fog nodes (SFN), which are responsible for sending update commands to CSP and verifying update proofs.

Cloud service provider (CSP): The CSP has powerful storage resources and can offer large-scale storage services to the data owner. The CSP is responsible for storing the ciphertext and returning the update proof to the fog.

Hyperledger Fabric: Hyperledger Fabric consists of service fog nodes, which is responsible for storing update transactions in the ledger. Due to the traceability and decentralization of blockchain, service fog nodes can detect misbehaviors of entities within the system and achieve traceability.

3.3 Adversary model

- The CSP is honest and curious, which means that the CSP may collude with other potential attackers to obtain data while storing data. Additionally, the CSP may privately delete infrequently accessed data to reduce storage space.
- The DO will not actively leak data but may frame the CSP for not honestly deleting data for compensation.
- The fog is semi-trusted. With the help of the trusted cloud platform management model based on TPM alliance, the management fog nodes are secure and reliable, and the service fog nodes are semi-trusted, which may collude with the CSP and forge deletion proofs.
- In SDUS-AD, there is a secure channel between entities. The secure channel is established by the shared key obtained by the two parties through the key agreement and can also be established based on the public-private key encryption and decryption between the two parties.

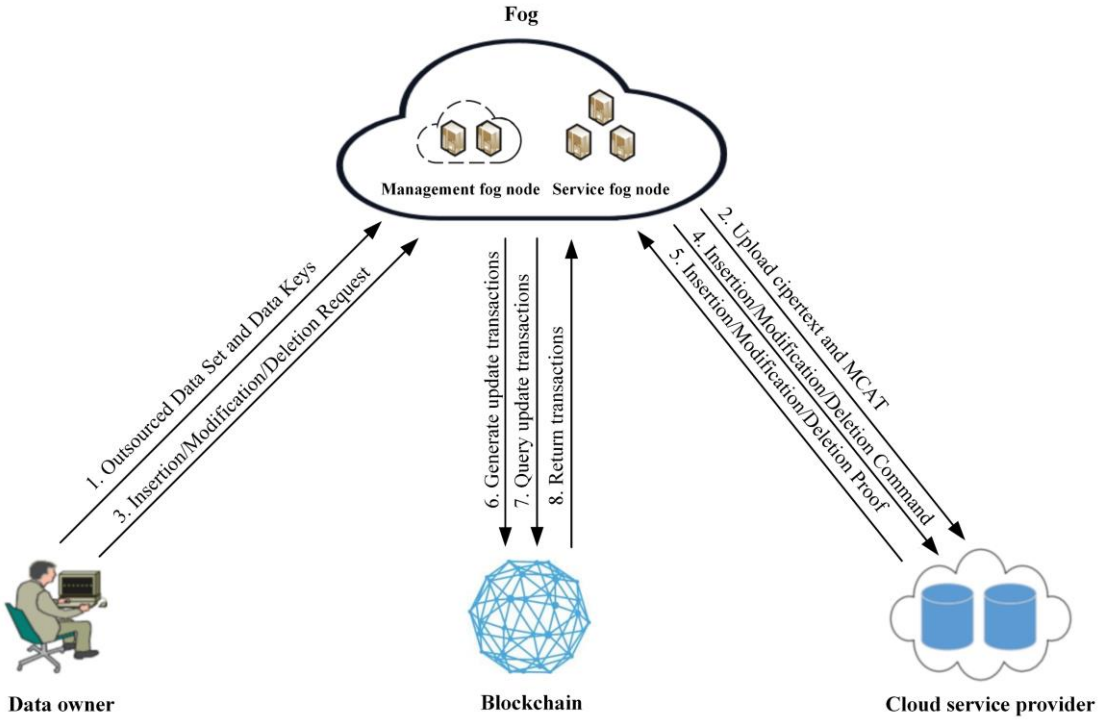

Figure 2. System model.

4. Multi-copy association tree

This section describes Multi-Copy Association Tree (MCAT), gives an example of the tree structure, and proves its correctness and security.

4.1 The structure of MCAT

To support insertion, modification, and deletion operations on data in multi-copy scenarios, we introduce a new primitive called Multi-Copy Association Tree, which can be regarded as an extension of general Merkle hash tree. The main difference between them is that MCAT is mainly used to build a multi-copy data deletion scheme and a leaf node of MCAT contains multiple data blocks, and the horizontal pointers are used to connect the current data block and its next data block, the vertical pointers are used to connect each data block to its copy data blocks. As shown in Figure 3, the data block structure under a leaf node contains p data blocks, the number of copies of each data block is q-1, the previous data block points to the storage address of the next data block, and each copy block is connected by a pointer. Compared with the traditional MHT, the number of leaf nodes is less, so the height of MCAT is lower than MHT. The syntax and security definitions of MCAT are similar to those of MHT. Due to the limitation of space, they will not be repeated here.

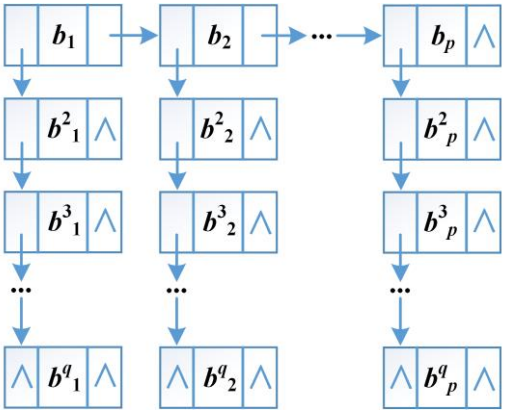

Figure 3. Data block structure.

The detailed construction of MCAT is as follows:

MCAT.KeyGen(1^κ): Given a security parameter κ, the data owner selects a cryptographic hash function H:$\{0, 1\}^* \rightarrow Z_p$, and executes the algorithm S.KeyGen(1^κ) to generate {SSK, SPK}. Let the public key MPK = {H, SPK} and the private key MSK = SSK.

MCAT.Hashing(D, MSK): Given a set of data blocks $D = \{ m_{n_i} \mid i \in \{0, \ldots, n\text{-}1\}\}$ and MSK, parse MSK into SSK. The construction of MCAT is similar to that of MHT. For any leaf node n_i, define the hash value Φ_{n_i} of node n_i by computing:

$$\Phi_{n_i} = H(I_{n_i}, p_{n_i}, p_{n_i} * q_{n_i}, H_{n_i}) \quad (1)$$

where I_{n_i} is the maximum index of data blocks contained in the node; p_{n_i} is the number of data blocks contained in the leaf node; $p_{n_i} * q_{n_i}$ is the total number of data blocks under the node; q_{n_i} is the number of copies; $H_{n_i}=H(m_{n_i})$ is the hash value of the encrypted set of data blocks under the leaf node.

For any internal node n_P, assuming that n_l and n_r are the left and right child nodes of the parent node n_P, respectively, set the hash value Φ_{n_P} of node n_P by computing:

$$\Phi_{n_P} = H(I_{(n_r+n_l)}, p_{(n_r+n_l)}, p_{(n_r+n_l)} * q_{(n_r+n_l)}, H_{n_P}) \quad (2)$$

where $I_{(n_r+n_l)}$ is the maximum value of the data block index contained in the left and right child nodes of the parent node n_P, $p_{(n_r+n_l)}$ is the number of data blocks contained in the left and right child nodes of the parent node n_P; $p_{(n_r+n_l)} * q_{(n_r+n_l)}$ is the total number of data blocks contained in the left and right child nodes of the parent node n_P; $H_{n_P}= H(\Phi_{n_r}, \Phi_{n_l})$, and Φ_{n_r} is the hash value of the right child node; Φ_{n_l} is the hash value of the left child node.

For convenience, we define Φ_{n_R} as the root value of the MCAT. Then it generates the signature of the root value σ = S.Sign(SSK, Φ_{n_R}) and sets the MCAT as authenticated data structure auth(D).

MCAT.Prove(x, auth(D), σ): Given a query request about data block x, the cloud defines an array AAI and executes the algorithm SearchDB (x, k, I_{n_k}, l, L), as shown in Algorithm 1. Then it obtains the index X of the leaf node where the data block x and its copy data blocks are located, and outputs the proof $\Omega_x= (I_{n_X}, p_{n_X}, q_{n_X}, AAI)$. AAI records the nodes in the authentication path of the leaf node where x is located.

SearchDB algorithm

1. **Input:** The queried data block x, the index of the current node k, the maximum value of the index of the data block contained in the current node I_{n_k}, the level of the current node l, the height of the tree L.
2. **Output:** The index of the leaf node where the data block to be queried is X.
3. **BEGIN**
4. **function** searchDB (x, k, I_{n_k}, l, L)
5. **if** $l = L$ and $x \leq I_{n_k}$ **then**
6. // The current leaf node correctly contains the data block to be queried
7. **return** X
8. **end if**
9. $i \leftarrow k.leftchild$
10. $j \leftarrow k.rightchild$
11. **if** $x \leq I_{n_i}$ **then**
12. **//** Continue to query the data block in the left subtree of this node
13. **return** searchDB (x, i, I_{n_i}, l+1, L)
14. **else**
15. // Continue to query the data block in the right subtree of this node
16. **return** searchDB (x, j, I_{n_j}, l+1, L)
17. **end if**
18. **end function**
19. **END**

Algorithm 1. SearchDB algorithm.

MCAT.Verify(X, MPK, Ω_x): Given the queried result X, the proof Ω_x, and the public key MPK, the verifier re-generates the hash value of the root node Φ_{n_R} with the pair $\{\Omega_x, n_X\}$ in a similar way to MHT. Finally, it performs S.Verify(σ, Φ_{n_R}, SPK) to check whether the signature σ is valid. If valid, the Verify algorithm outputs "1". Otherwise, it outputs "0".

MCAT.Update(MSK, I'_{n_i}, p'_{n_i}, H'_{n_i}): Given a new value H'_{n_i} of leaf node n_i, the data owner updates the hash value $\Phi'_{n_i} = H(I'_{n_i}, p'_{n_i}, p'_{n_i} * q_{n_i}, H'_{n_i})$ of the leaf node n_i. Then, it updates the new values and hash values from the leaf node n_i to the root node in MCAT and obtains the new root value Φ'_{n_i} of MCAT. Finally, it calculates the signature σ' = S.Sign(MSK, Φ'_{n_R}) of the root value and outputs the latest authenticated data structure auth(D') and the signature σ'. The process of the MCAT is similar to that in MHT.

Remarks 1. Same as MHT, the update and verification complexity of MCAT with n leaf nodes is $O(log_2 n)$. In addition, the insertion/deletion operations of MHT will increase/decrease the number of leaf nodes, resulting in changes in the MHT structure. Let the index of the newly inserted data block in the MCAT be the average of the indexes of the data blocks before and after it. Therefore, the structure of the MCAT will not change due to the dynamic update of the data. In addition, in the case of multi-copy storage, the

MHT has many leaf nodes, and the number of hash operations is frequent, which is inefficient. Therefore, the MCAT is more efficient in multi-copy scenarios.

Remarks 2. Before giving an example of MCAT, the structure of the multi-copy association tree is introduced. Each node n_i in MCAT is comprised of a quadruple (I_{n_i}, p_{n_i}, $p_{n_i}*q_{n_i}$, H_{n_i}). For convenience, we omit the value and hash value of all nodes in MCAT. Figure 4 depicts the detailed structure of MCAT, where the value in the node is the index.

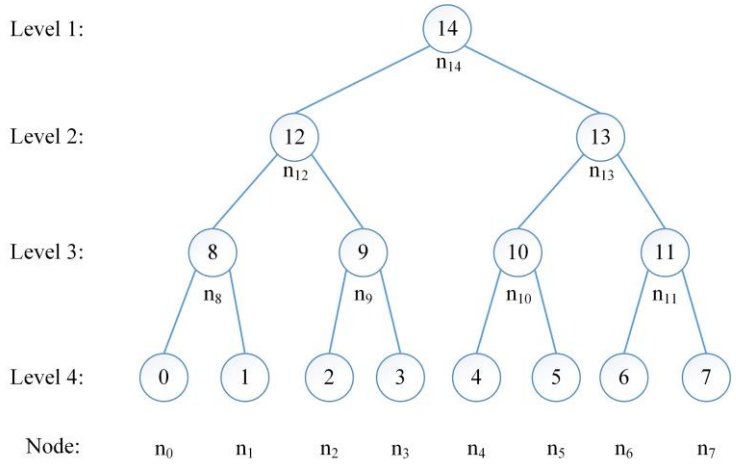

Figure 4. The structure of MCAT.

4.2 Correctness

Suppose the cloud outputs the index X of the leaf node n_X where the data block x is to be queried and its copy data blocks are located according to the algorithm MCAT.Prove, that the root node re-generated by the verifier is the same as the original one.

According to the equation (2), we can get the hash value of the root node as follows:

$$\Phi_{n_R}=H(I_{(n_r+n_l)}, p_{(n_r+n_l)}, p_{(n_r+n_l)}*q_{(n_r+n_l)}, H(\Phi_{n_r} \| \Phi_{n_l})) \quad (3)$$

where the nodes n_l and n_r are the left and right child nodes of the root node n_R, respectively. Let us recursively regard the child node as the root up to the leaf node. Then, the term $\Phi_{n_r} \| \Phi_{n_l}$ in equation (1) can be expressed as follows:

$$\Phi_{n_r} \| \Phi_{n_l}= \Phi_{n_{2^L-1-(2^1)}} \| \Phi_{n_{2^L-1-(2^1+1)}}$$

$$=H(\Phi_{n_{2^L-1-(2^2+3)}} \| \Phi_{n_{2^L-1-(2^2+2)}}) \| H(\Phi_{n_{2^L-1-(2^2+1)}} \| \Phi_{n_{2^L-1-2^2}})$$

$$=H\left(\cdots H\left(\begin{array}{c} H\left(\begin{array}{c} H(\Phi_{n_{2^L-1-(2^l-2^{l-1})}} \| \Phi_{n_{2^L-1-(2^l-2^{l-1}+1)}}) \| \\ H(\Phi_{n_{2^L-1-(2^l-2^{l-1}+2)}} \| \Phi_{n_{2^L-1-(2^l-2^{l-1}+3)}}) \end{array}\right) \\ \| \cdots \| \\ H\left(\begin{array}{c} H(\Phi_{n_{2^L-1-2^l}} \| \Phi_{n_{2^L-1-(2^l+1)}}) \| \\ H(\Phi_{n_{2^L-1-(2^l+2)}} \| \Phi_{n_{2^L-1-(2^l+3)}}) \end{array}\right) \end{array}\right) \cdots\right)$$

…

$$=H\left(\cdots H\left(\begin{array}{c} H\begin{pmatrix} H(\Phi_{n_0}\|\Phi_{n_1})\| \\ H(\Phi_{n_2}\|\Phi_{n_3}) \end{pmatrix} \\ \|\cdots\| \\ H\begin{pmatrix} H(\Phi_{n_{2^{L-1}-4}}\|\Phi_{n_{2^{L-1}-3}}) \\ H(\Phi_{n_{2^{L-1}-2}}\|\Phi_{n_{2^{L-1}-1}}) \end{pmatrix} \end{array}\right)\cdots\right) \tag{4}$$

Then from Equation 3, we can get:

$$p_{n_l}=p_{n_0}+\ldots+p_{n_{2^{L-2}-1}} \tag{5}$$

$$p_{n_r}=p_{n_{2^{L-2}}}+\ldots+p_{n_{2^{L-1}-1}} \tag{6}$$

$$p_{n_l}*q_{n_l}=p_{n_0}*q_{n_0}+\ldots+p_{n_{2^{L-2}-1}}*q_{n_{2^{L-2}-1}} \tag{7}$$

$$p_{n_r}*q_{n_r}=p_{n_{2^{L-2}}}*q_{n_{2^{L-2}}}+\ldots+p_{n_{2^{L-1}-1}}*q_{n_{2^{L-1}-1}} \tag{8}$$

According to the algorithm Prove, the array *AAI* records all the sibling nodes in the path from the leaf node to the root node. *AAI*[*l*] stores sibling nodes at level $l + 2$. For simplicity, we assume that the index of the sibling node of layer $l + 2$ is j_l. We can get:

$AAI[l]$

$$=H\left(\cdots H\left(\begin{array}{c} H\begin{pmatrix} H(\Phi_{n_{2^L-2^{L-l-2}*\left(2^L-1-j_l-1\right)}})\|\Phi_{n_{2^L-2^{L-l-2}*\left(2^L-1-j_l-1\right)+1}})\| \\ H(\Phi_{n_{2^L-2^{L-l-2}*\left(2^L-1-j_l-1\right)+2}}\|\Phi_{n_{2^L-2^{L-l-2}*\left(2^L-1-j_l-1\right)+3}}) \end{pmatrix} \\ \|\cdots\| \\ H\begin{pmatrix} H(\Phi_{n_{2^L-1-2^{L-l-2}*\left(2^L-1-j_l\right)}})\|\Phi_{n_{2^L-1-2^{L-l-2}*\left(2^L-1-j_l\right)-1}})\| \\ H(\Phi_{n_{2^L-1-2^{L-l-2}*\left(2^L-1-j_l\right)-2}}\|\Phi_{n_{2^L-1-2^{L-l-2}*\left(2^L-1-j_l\right)-3}}) \end{pmatrix} \end{array}\right)\cdots\right) \tag{9}$$

Then from Equation 4 and Equation 9, we can get:

$$\Phi_{n_r}\|\Phi_{n_l}=H(AAI[0]\|\ldots\|(H(AAI[L\text{-}2]\|\Phi_{n_X}))) \tag{10}$$

From Equation 5 and Equation 6, it is not hard to see that:

$$p_{(n_r+n_l)}=p_{n_0}+\ldots+p_{n_{2^{L-1}-1}} \tag{11}$$

From Equation 7 and Equation 8, we can get:

$$p_{(n_r+n_l)}*q_{(n_r+n_l)}=p_{n_0}*q_{n_0}+\ldots+p_{n_{2^{L-1}-1}}*q_{n_{2^{L-1}-1}} \tag{12}$$

From Equation 8, Equation 9, and Equation 10, it is not hard to see that:

$$\Phi_{n_R}=H\begin{pmatrix} I_{n_{2^{L-1}-1}}\| \\ p_{n_0}+\ldots+p_{n_{2^{L-1}-1}}\| \\ p_{n_0}*q_{n_0}+\ldots+p_{n_{2^{L-1}-1}}*q_{n_{2^{L-1}-1}}\| \\ H(AAI[0]\|\ldots\|(H(AAI[L\text{-}2]\|\Phi_{n_X}))) \end{pmatrix} \tag{13}$$

It can be seen that Ω_x can always pass the verification of the verifier if all the procedures are performed as describrd above.

4.3 Security

Theorem 1. Our proposed multi-copy association tree is safe if *H* is a collision resistant hash function.

Proof. The adversary A outputs a forgery for the data block x. The forgery is proved by the index n^*_x of the leaf node where x is located and the corresponding proof $\Omega^*_x = (I^*_{n_X}, p^*_{n_X}, q^*_{n_X}, \{AAI^*[0], \ldots, AAI^*[L\text{-}2]\})$. If A destroys the security of multi-copy association tree, The adversary A can use this forgery to construct a method B to destroy the collision-resistant property of the hash function.

Assuming that A forged the information that passes the verification, the following equation holds:

$$\Phi_{n_R}=H\begin{pmatrix} I^*_{n_{2^{L-1}-1}} \| \\ p^*_{n_0}+\ldots+p^*_{n_{2^{L-1}-1}} \| \\ p^*_{n_0}*q^*_{n_0}+\ldots+p^*_{n_{2^{L-1}-1}}*q^*_{n_{2^{L-1}-1}} \| \\ H(AAI^*[0] \|\ldots\| (H(AAI^*[L\text{-}2] \| \Phi^*_{n_X}))) \end{pmatrix} \tag{14}$$

Similarly, for the correct index X of the leaf node where the data block x is located and the proof $\Omega_x = (I_{n_X}, p_{n_X}, q_{n_X}, \{AAI[0], \ldots, AAI[L\text{-}2]\})$, there are:

$$\Phi_{n_R}=H\begin{pmatrix} I_{n_{2^{L-1}-1}} \| \\ p_{n_0}+\ldots+p_{n_{2^{L-1}-1}} \| \\ p_{n_0}*q_{n_0}+\ldots+p_{n_{2^{L-1}-1}}*q_{n_{2^{L-1}-1}} \| \\ H(AAI[0] \|\ldots\| (H(AAI[L\text{-}2] \| \Phi_{n_X}))) \end{pmatrix} \tag{15}$$

From Equation 14 and Equation 15, it is not difficult to see that:

$$H\begin{pmatrix} I^*_{n_{2^{L-1}-1}} \| \\ p^*_{n_0}+\ldots+p^*_{n_{2^{L-1}-1}} \| \\ p^*_{n_0}*q^*_{n_0}+\ldots+p^*_{n_{2^{L-1}-1}}*q^*_{n_{2^{L-1}-1}} \| \\ H(AAI^*[0] \|\ldots\| (H(AAI^*[L\text{-}2] \| \Phi^*_{n_X}))) \end{pmatrix}$$

$$=$$

$$H\begin{pmatrix} I_{n_{2^{L-1}-1}} \| \\ p_{n_0}+\ldots+p_{n_{2^{L-1}-1}} \| \\ p_{n_0}*q_{n_0}+\ldots+p_{n_{2^{L-1}-1}}*q_{n_{2^{L-1}-1}} \| \\ H(AAI[0] \|\ldots\| (H(AAI[L\text{-}2] \| \Phi_{n_X}))) \end{pmatrix} \tag{16}$$

Therefore, if adversary A successfully forges n^*_X and Ω^*_x, A can use this forgery to construct an effective algorithm B to destroy the collision-resistant property of the hash function (such as $H(a) = H(b) \wedge a \neq b$). Therefore, the security of the multi-copy association tree is proved.

4.4 An illustrative instantiation

This part presents a concrete example of MCAT to illustrate the construction details of MCAT. Without loss of generality, take the MCAT with four leaf nodes as an example. For each leaf node n_i, $p_i = 2$, $q_i = 3$, where $i \in [0,3]$. The value range of the data block under each leaf node is left closed and right open. This is because when the data block under the leaf node is inserted, the index value of the data block to be inserted is the average value of the indexes of the data blocks before and after it. When a new data block is inserted after the last data block of the leaf node, the index of the data block to be inserted is the average value of the index of the last data block of the current leaf node and the index of the first data block of the next leaf node. Therefore, the index value of the data block to be inserted can be infinitely close to the index value of the first data block of the next leaf node, but it will never be equal to it. It is worth noting that the scope of the last leaf node of the multi-copy associative tree is a closed interval. When a new data block is inserted after the last data block, the index value of the data block is obtained by increasing 1 to the index value of the last data block. Figure 5 depicts the *Hashing* phase of the MCAT. Assume that the number of copies of each data block is 3. For each leaf node n_i, the hash value is expressed as $\Phi_{n_i} = H(I_i, 2, 6, H_{n_i})$. For the internal node n_j, p_j is the sum of the values of its left and right children, $q_j = 3$. The hash value of n_j can be calculated from the maximum index of the node, the total number of data block indexes, and the hash values of its left and right children. As shown in Figure 5, $\Phi_{n_4} = H(2+2, 4, 4*3, H_{n_4}) = H(4, 4, 12, H(\Phi_{n_0}, \Phi_{n_1}))$.

When a data block x of the leaf node n_X is updated, MCAT will find the index of the leaf node where the data block to be updated is located through the MCAT.Prove algorithm, and reconstruct the MCAT. Figure 6 shows the update process of the MCAT. At this time, the index of the number of data blocks under the leaf node will not change, and it is only necessary to recalculate the hash values of the leaf node and internal nodes to be updated in sequence. Figure 7 shows the structure change of MCAT when a data block is deleted. At this point, $\Phi_{n_3} = H(7, 1, 3, H'_{n_3})$, p_3 is changed from 2 to 1, and the hash value of the internal nodes of the MCAT is calculated in turn. Figure 8 shows the structure change of MCAT when a data block is inserted. At this time, $\Phi_{n_2} = H(6, 3, 9, H'_{n_2})$, p_2 is changed from 2 to 3, and the hash values of the internal nodes of MCAT are calculated.

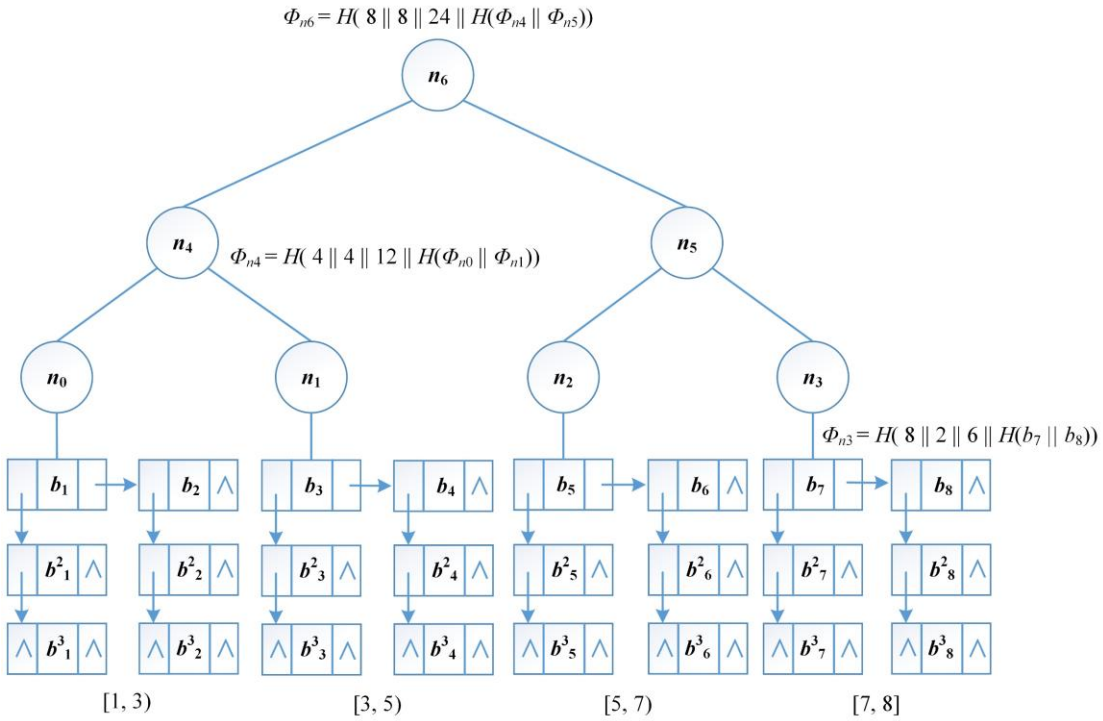

Figure 5. An example of MCAT.

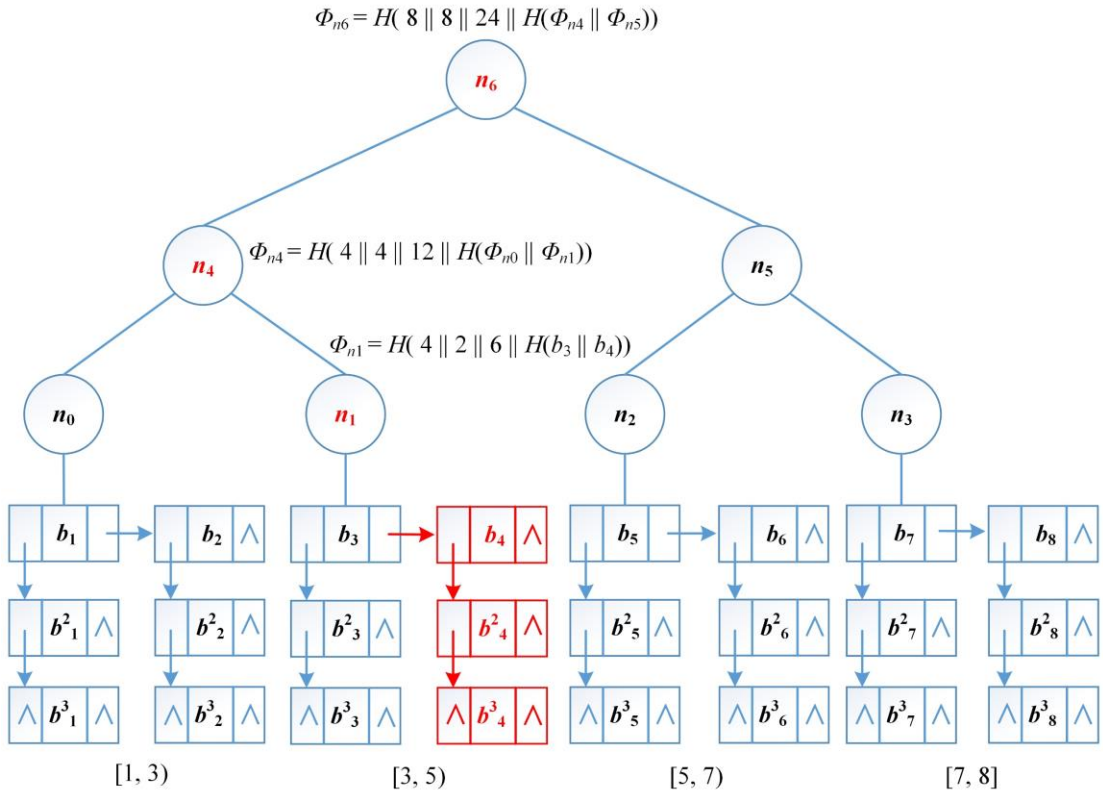

Figure 6. The MCAT during update phase.

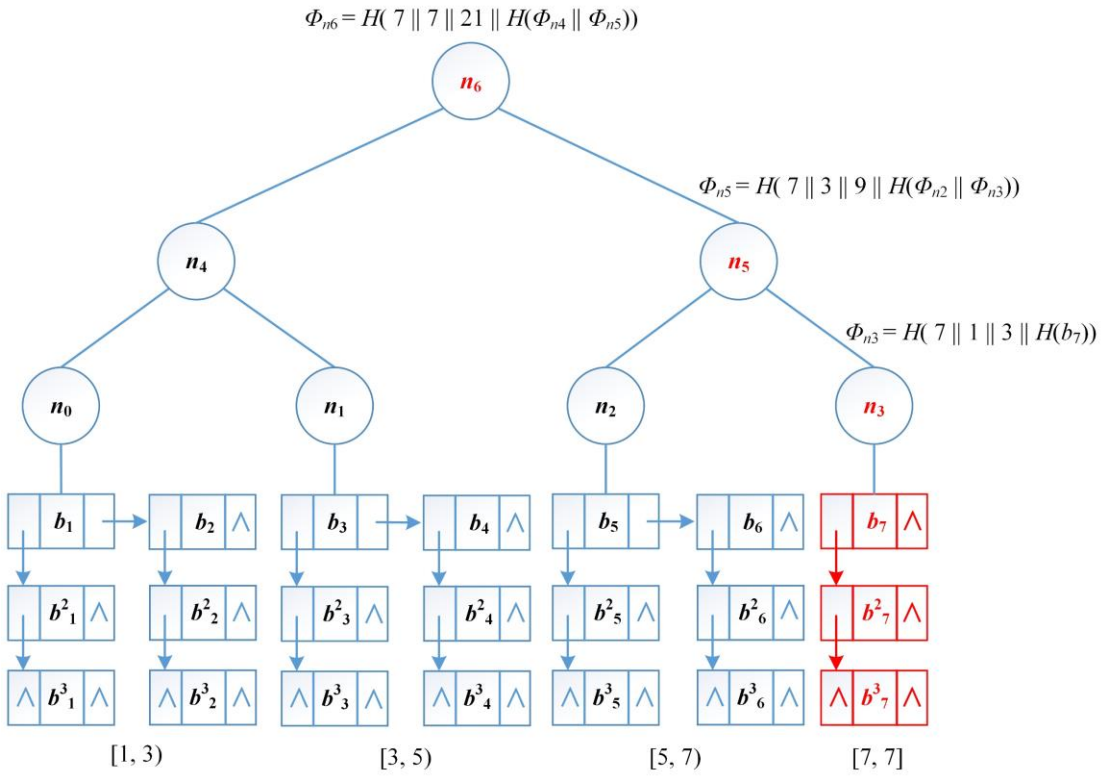

Figure 7. The MCAT during delete phase.

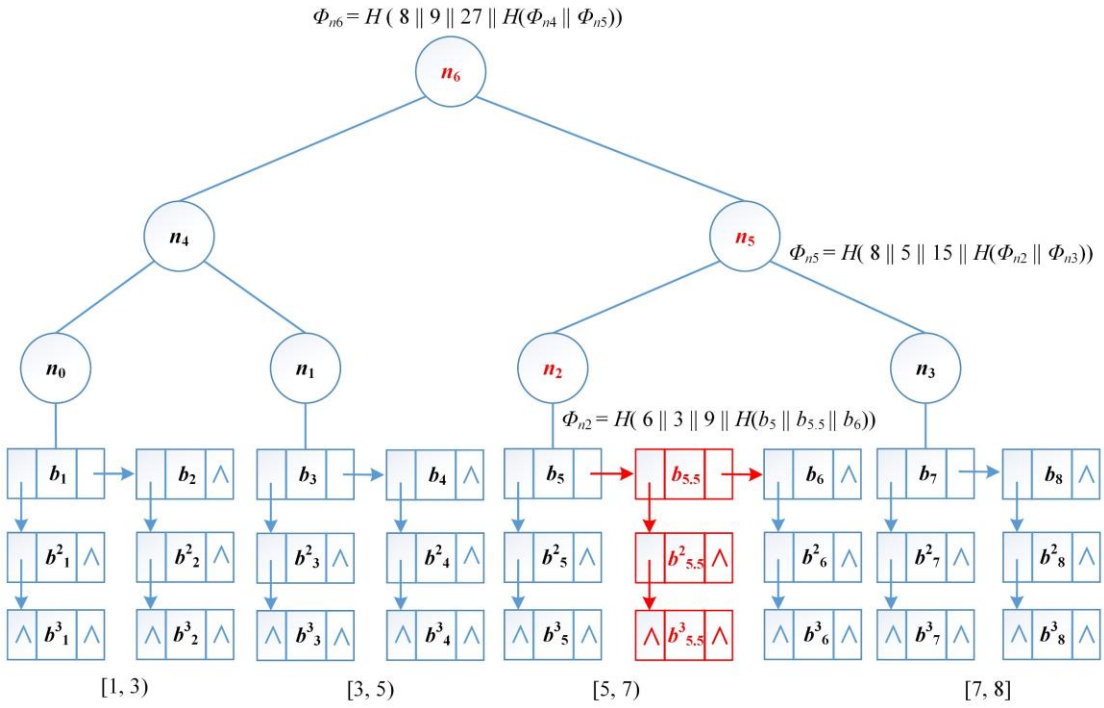

Figure 8. The MCAT during insert phase.

5. Detailed structure of the scheme

5.1 Scheme design

The design idea of SDUS-AD is as follows:

First, to avoid the problem of the same ciphertext of file copies caused by single-key encryption, the data owner generates different data keys for the data file and its copies, and uploads the data blocks into the fog. Then the data blocks are encrypted by the management fog node, the service fog node constructs a multi-copy association tree, and sends the ciphertext to the CSP. Second, when the data owner has update requirements (such as insertion, modification, and deletion), the SFN sends update commands to the CSP. After the CSP performs the update operation, it will return the update proof to the SFN. The SFN needs to verify the correctness of proof to judge whether the CSP has performed the update operation honestly. Finally, the SFN generates the update transaction and records it in the blockchain ledger. When a trust problem occurs in the system, the SFN queries the transactions recorded in the blockchain ledger to achieve arbitration of the total misbehavior of the system.

5.2 Algorithm Description

This part presents the detailed structure of SDUS-AD. Define $Enc_b(f)$ represents using b to encrypt the file f, and $Sig_b(f)$ represents using b to sign f. Fid is the unique id of the file f, which is convenient for the CSP to search the file in the physical disk. $H(\bullet)$: $\{0, 1\}^* \rightarrow Z_q$ represents the collision resistant hash function. The ECDSA key pairs used by the data owner, the cloud service provider, and the service fog node for signing are (pk_o, sk_o), (pk_s, sk_s), and (pk_f, sk_f), respectively.

Setup ($\{sp_i\}$): The DO inputs the security parameters sp_i, and generates the data key $k_i = H(Fid, sk_o, sp_i)$, where $i \in [1, q]$, q is the total number of the file and its copies. DO divides the file f into c data blocks and generates a set of data blocks $S_{DB} = \{b_0, b_1, ..., b_c\}$.

Upload (S_{DB}, q, Fid, $\{k_i\}$, sk_o): The DO sends $info_{up} = (S_{DB}, \{k_i\}, Fid)$ to the fog, and deletes the local file. The management fog node MFN encrypts S_{DB} with the data keys $\{k_i\}$ to obtain a set of ciphertext blocks $S_{CB_i} = Enc_{k_i}(S_{DB})$, where Enc is indistinguishability under chosen-plaintext attack (IND-CPA) advanced encryption standard (AES) algorithm. The service fog node SFN constructs MCAT with S_{CB_i} as the leaf node, and sends the final ciphertext $CT = (Fid, \{S_{CB_i}\})$ to the CSP. After the CSP receives CT, it builds the MCAT in the same way.

Insert (y, b_{new}, T_i, Fid, sk_o, sk_f, sk_c, tag): The DO wants to insert a new data block b_{new} after the data block with block index y, and generates $Sig_{di} = Sig_{sk_o}(T_i, Fid, b_{new}, y, tag)$, where T_i is the current timestamp, tag identifies the identifier of the update operation. When the update operation is insertion, sets the operation identifier tag to 0. When modifying a data block, the tag equals 1, and when a data block is deleted, the tag is equal to 2. The DO sends $IR_{DO} = (Sig_{di}, T_i, Fid, y, b_{new}, tag)$ to the fog. The MFN encrypts the data block b_{new} and its copy blocks to generate ciphertext blocks $EB_{new} = \{e^1_{new}, ..., e^q_{new}\}$. The SFN performs MCAT.Prove(y) to get the leaf node index n_Y where y is located. Then,

increment p_{n_Y} by 1 and run MCAT.Update(sk_f, I_{n_Y}, p'_{n_Y}, H'_{n_Y}) to obtain the new authenticated data structure auth(D') and the new root value Φ_{n_R}. The SFN calculates $Sig_{fi} = Sig_{sk_f}(T_i, EB_{new}, Fid, y, tag)$ and sends $IR_{fog} = (Sig_{fi}, T_i, y, EB_{new}, Fid, tag)$ to the CSP, and the CSP reconstructs the MCAT according to the same requirements and obtains the new root Φ'_{n_R}, then runs $Sig_r \leftarrow$ S.Sign(sk_c, Φ'_{n_R}). The CSP generates the insertion proof $\varphi_i = (\Phi'_{n_R}, Sig_r)$ and returns it to the SFN. The SFN runs Verification (φ_i, pk_s, Φ_{n_R}), as shown in Algorithm 2, and generates the insertion transaction record in the blockchain ledger, the chaincode for generating transaction as shown in Algorithm 3.

Verification algorithm

1. **Input:** The evidence to be verified φ, the public key of the CSP pk_s, the value of the root node of the MCAT generated by the fog Φ_{n_R}.
2. **Output:** The update result *result*, the signature of the fog node on the root value Sig_R.
3. **BEGIN**
4. $Root'_{vd} = H(\Phi'_{n_R})$
5. $Root_{vd} = Compute_{pks}(Sig)$
6. **if** $Root'_{vd} = Root_{vd}$ **then**
7. **if** $\Phi_{n_R} = \Phi'_{n_R}$ **then**
8. $result = Success$
9. **else**
10. $result = Failure$
11. **end if**
12. $Sig_R = Sig_{sk_f}(\Phi_{n_R})$
13. **end if**
14. ***Return*** *result,* Sig_R
15. **END**

Algorithm 2. Verification algorithm.

Chaincode for transaction

1. **Input:** IR_{DO}, Φ_{n_R}, IR_{fog}, MR_{DO}, MR_{fog}, DR_{DO}, DR_{fog}, φ_i, φ_m, φ_d, tag, $result$, Sig_R.
2. **if** $tag = 0$ **then**
3. Transaction $Tx(insertion) = (IR_{DO}, IR_{fog}, result, Sig_{sk_f}(\Phi_{n_R}))$
4. Save $Tx(insertion)$ in ledger
5. **else if** $tag = 1$ **then**
6. Transaction $Tx(modification) = (MR_{DO}, MR_{fog}, result, Sig_{sk_f}(\Phi_{n_R}))$
7. Save $Tx(modification)$ in ledger
8. **else** $tag = 2$ **then**
9. Transaction $Tx(deletion) = (DR_{DO}, DR_{fog}, result, Sig_{sk_f}(\Phi_{n_R}))$
10. Save $Tx(deletion)$ in ledger

Algorithm 3. Chaincode for update.

Modify (z, b_z, T_u, Fid, sk_o, sk_f, sk_c, tag): Similar to the insertion operation, the DO modifies the data block with block index z to b_z. At this point, the DO generates $Sig_{du} = Sig_{sk_o}(T_u, Fid, z, b_z, tag)$. The DO sends $MR_{DO} = (Sig_{du}, T_u, Fid, b_z, tag)$ to the fog. The MFN encrypts the data block b_z to generate the ciphertext block $EB_z = \{e^1{}_z, ..., e^q{}_z\}$, and the SFN runs MCAT.Prove(z) to obtain the leaf node index n_Z where z is located. Then, the SFN runs MCAT.Update(sk_f, I_{n_Z}, p'_{n_Z}, H'_{n_Z}) to obtain the new authenticated data structure auth(D') and the new root value Φ_{n_R}. The SFN calculates $Sig_{fu} = Sig_{sk_f}(T_u, EB_z, Fid, z, tag)$, and sends $MR_{fog} = (Sig_{fu}, T_u, z, EB_z, Fid, tag)$ to the CSP, and the CSP reconstructs the MCAT according to the same requirements and obtains the new root Φ'_{nR}, runs $Sig_u \leftarrow$ S.Sign(sk_c, Φ'_{n_R}). The CSP generates a modification proof $\varphi_m = (\Phi'_{n_R}, Sig_u)$ and returns it to the SFN. The SFN runs Verification (φ_m, pk_s, Φ_{n_R}), generates the modification transaction, and records it in the blockchain ledger.

Delete (a, T_d, Fid, sk_o, sk_f, sk_c, tag): When the DO wants to delete the data block whose block index is a, the DO generates $Sig_{dd} = Sig_{sk_o}(T_d, Fid, a, tag)$. The DO sends $DR_{DO} = (Sig_{dd}, T_d, Fid, a, tag)$ to the fog. After the SFN receives the DR_{DO}, it runs MCAT.Prove(a) to get the leaf node index n_A where a is located, runs MCAT.Update(sk_f, I_{n_A}, p'_{n_A}, H'_{n_A}) to obtain the new authenticated data structure auth(D') and the new root value Φ_{n_R}. The SFN calculates $Sig_{fd} = Sig_{sk_f}(T_d, Fid, a, tag)$, and sends $DR_{fog} = (Sig_{fd}, T_d, a, Fid, tag)$ to CSP. The CSP reconstructs MCAT according to the same requirements, obtains the new root Φ'_{n_R}, and runs $Sig_d \leftarrow$ S.Sign(sk_c, Φ'_{n_R}). The CSP generates a deletion proof $\varphi_d =$

(Φ'_{n_R}, Sig_d) and returns it to the SFN. The SFN runs Verification (φ_d, pk_s, Φ_{n_R}), generates the deletion transaction, and records it in the blockchain ledger.

6. Performance Evaluation

6.1 Security Proof

Assuming (M, E, D) is an IND-CPA secure symmetric encryption scheme, H is a pseudo-random function, where $M = \{0, 1\}^z$, E: $M \times M \rightarrow U$, and $M^K = \{0, 1\}^q$. Then (M', E') is IND-CPA safe, and $E'_M(x) = E_{K_M(s)}(x)$.

Proof. If (M', E') is IND-CPA safe, it means that for any message x, y, and any probabilistic polynomial time attacker A, there is always a polynomial $p(n)$ and an integer Z. If $z > Z$, the following equation holds

$$AdvA_{xy}^{E'M'} = | Pr[A^{E'M'}(E_{K_{M'}(x)}(x)) = 1] - Pr[A^{E'M'}(E_{K_{M'}(y)}(y)) = 1] | < \frac{1}{p(n)}$$

Then, we define a set of games:

Game 0:

1. The challenger runs the Setup algorithm Setup (sp) and generates the data key k'.
2. Input sp, attacker A asks the random predictor $Enc_{K_{M'}}(\cdot)$ and outputs two messages m_0 and m_1 of the same length.
3. The challenger randomly selects $b \in \{0, 1\}$, calculates $c = Enc_{K_{M'}}(m_b)$, and sends c to attacker A.
4. Attacker A asks the random predictor $Enc_{K_{M'}}(\cdot)$ and outputs the prediction b' of b.
5. If $b = b'$, output 1; otherwise, output 0.

Game 1:

1. The challenger runs the Setup algorithm Setup (sp) and generates the data key k'.
2. Input sp, attacker A asks the random predictor $Enc_{\emptyset}(\cdot)$ and outputs two messages m_0 and m_1 of the same length.
3. The challenger randomly selects $b \in \{0, 1\}$, calculates $c = Enc_{\emptyset}(m_b)$, and sends c to attacker A.
4. Attacker A asks the random predictor $Enc_{\emptyset}(\cdot)$ and outputs the prediction b' of b.
5. If $b = b'$, output 1; otherwise, output 0.

Game 2:

Game 2 is similar to Game 1. The difference is that the encrypted message has nothing to do with the key.

Based on the indistinguishability of the pseudo-random function and the IND-CPA security of (M', E'), it can be proved that Game 0 and Game 1 are indistinguishable. In other words, for any message x and any probabilistic polynomial time attacker A, there is a

polynomial time algorithm $p(n)$ and an integer Z_1. When $z > Z_1$, the following equation holds

$$AdvA_1 = |Pr[A^{E_{M'}}(E_{K_{M'}(x)}(x)) = 1] - Pr[A^{E_{\emptyset}}(E_{\emptyset(x)}(x)) = 1]| < \frac{1}{4p(n)}$$

Similarly, for any message y and any probabilistic polynomial time attacker A, there is a polynomial time algorithm $p(n)$ and an integer Z_2. When $z > Z_2$, the following equation holds

$$AdvA_2 = |Pr[A^{E_{M'}}(E_{K_{M'}(y)}(y)) = 1] - Pr[A^{E_{\emptyset}}(E_{\emptyset(y)}(y)) = 1]| < \frac{1}{4p(n)}$$

Therefore, Game 1 and Game 0 are indistinguishable.

In addition, similar proofs can be used to prove that Game 1 and Game 2 are indistinguishable. Since the proof is similar, so we do not repeat it here in detail.

6.2 Functional Analysis

This part analyzes the functions implemented by SDUS-AD.

Fine-grained deletion: When the DO needs to delete data, the index of the data block to be deleted is specified in the deletion request. Therefore, only the specified data block and its copy blocks will be deleted. Other data and other users' data will not be deleted and accessed normally.

Timeliness: When the CSP receives the deletion command sent by the SFN, it will immediately execute the deletion operation to ensure data security.

The proof of deletion: The CSP will update the multi-copy association tree and get the hash value of the new root node after performing the deletion operation. After signing the root node with its private key, the CSP will generate a deletion proof and return the proof to the SFN. The SFN can check whether the CSP has completed the deletion operation as required by verifying the deletion proof.

Traceability: The update request sent by the DO and the update proof returned by the CSP will be uploaded to the blockchain ledger. Due to the decentralization and traceability of the blockchain, the data owner and the CSP cannot collude with most nodes in the blockchain to tamper with the proof to obtain compensation or avoid punishment. Therefore, when an entity in the system misbehaves, both the DO and the CSP can be held accountable by querying the evidence held in the Hyperledger fabric ledger.

To better illustrate the functions implemented by SDUS-AD, Table 1 shows the function comparison between SDUS-AD and schemes [35], [39]. It can be seen from the comparison:

- SDUS-AD introduces blockchain technology to realize the traceability of the scheme.
- The new primitive multi-copy association tree is proposed to meet the update operations (such as insertion, deletion, and modification) under the multi-copy data and realizes the public verification of deletion results without introducing any trusted third party.

- Combined with fog computing, it achieves the needs of mobile devices to access and modify the data.

Scheme	ADM [35]	Scheme [39]	SDUS-AD
Accountable traceability	×	√	√
Trusted third party	√	×	×
Dynamic data insertion	×	√	√
Verifiable data deletion	×	√	√
Data multi-copy storage	√	×	√
Dynamic data update	×	×	√
Fog environment	×	×	√

Table 1. Functionality comparison of three schemes.

6.3 Security Analysis

This part analyzes the security of SDUS-AD. The details are as follows:

Theorem 1. The SDUS-AD satisfies data confidentiality.

Proof. SDUS-AD uses the trusted cloud platform management model based on TPM alliance to realize a safe and reliable management fog node cluster, and uses the management fog node to protect data privacy. The DO will not actively disclose their sensitive data, and the DO can hold the fog accountable if data leakage occurs. The Hyperledger fabric is composed of service fog nodes with distributed characteristics, so attackers cannot collude with most service fog nodes to tamper with data. The data is encrypted using the IND-CPA secure symmetric encryption algorithm before outsourcing, and it is worth noting that SDUS-AD is not restricted by encryption technology, any secure encryption method is suitable for this scheme. In addition, SDUS-AD uses multiple keys to encrypt the file copies to avoid the problem of multiple copies leakage caused by a single key.

Theorem 2. The SDUS-AD satisfies verifiability.

Proof. The CSP uses the private key to sign the deletion proof and generates the deletion proof to send to the service fog node, while the private key is kept privately by the CSP. After receiving the proof, the SFN uses the public key of the CSP to verify the deletion proof. The verification process does not require any private information of the DO and the CSP. In addition, any SFN can verify the results of data deletion. Therefore, it can be considered that SDUS-AD satisfies the verifiability of deletion proof.

Theorem 3. The SDUS-AD satisfies traceability.

Case 1: Dishonest data owner.

Proof. After requesting the deletion of a data block, the dishonest DO may deny its deletion request and slander the CSP of privately deleting the data that the DO still needs. In this

case, the CSP can apply to the Hyperledger fabric for arbitration. The Hyperledger fabric ledger contains the deletion request DR_{DO} sent by the DO, and DR_{DO} contains the information Sig_{dd} signed with the DO private key sk_o. Only the DO can generate DR_{DO}, and no one else can forge it. Therefore, DR_{DO} can be viewed as evidence, proving that the DO had required the CSP to delete the data block. Therefore, the dishonest DO cannot deny his data deletion request and slander the CSP successfully.

Case 2: Untrusted Cloud Service Provider.

Proof. The dishonest CSP may arbitrarily delete some data blocks that are rarely accessed to save local storage space. If the dishonest data deletion is discovered, the CSP may slander the deletion operation performed by receiving the deletion command sent by the SFN. In this case, the CSP must be required to provide the received deletion command DR_{fog}, which DR_{fog} contains the information Sig_{fd} signed with the SFN private key. Therefore, the malicious CSP cannot forge DR_{fog} with a non-negligible probability. In addition, the deletion request DR_{DO} generated by the DO cannot be found in the Hyperledger fabric ledger. Therefore, the malicious CSP cannot successfully deny his dishonest deletion operation and frame the DO and the fog.

7. Efficiency analysis

This section experiments with the scheme and analyzes the results. The experiment is deployed on a physical machine. The hardware configuration of the experimental host is an Intel (R) Core (TM) i5-10400F CPU @ 2.90GHz, the memory size is 16 GB, and the operating system is Microsoft Windows 10 (64-bit). All experiments are implemented through the Ubuntu virtual machine built on VMware workstation 16.2.1. The specific configuration of Ubuntu is Intel (R) Core (TM) i5-10400F CPU @ 2.90GHz, the operating system is Ubuntu 16.04 LTS, the memory is 8 GB, and the hard disk is 25 GB. We choose SHA-256 as the hash function. The compilation environment of the experimental program is Pycharm2021.3, the programming language is Python, and the key size is set to 16 bit.

7.1 Update Operation Overhead

Figures 9, 10, and 11 show the changes in the insertion, deletion, and modification overhead under the condition of changing the file size, respectively. The experimental conditions are set as follows: the file size is 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB; the size of the data block to be updated is fixed at 1 MB; the number of copies of SDUS-AD is 4.

It can be seen from Figures 9 and 10 that the insertion and deletion overhead of SDUS-AD and scheme [39] both increase linearly when the file size increases, and SDUS-AD is relatively higher than scheme [39] in the insertion and deletion phases. In SDUS-AD, the main computation cost is dominated by signature and verification calculations in the insertion and deletion phase. Compared with the single-copy scheme [39], SDUS-AD

achieves the insertion operation of four copy files with an average increase of about 90%time costs compared with the scheme [39] and deletes four copy files with an average increase of about 150%time costs. If the scheme [39] implements the insertion and deletion of four copy files, four insertion and deletion operations are required, requiring 400% of the time. Therefore, using the multi-copy association tree to realize the update operation of copy files improves the security of SDUS-AD at the cost of overhead. In addition, the insertion and deletion operations and their verification are performed by the service fog node and do not occupy the local computing resources of the DO. As shown in Figure 11, the modification cost of SDUS-AD and scheme [39] increases linearly with the increase of file size, and scheme [39] has a slightly higher overhead than SDUS-AD. In the scheme [39], the computation cost is dominated by a deletion operation and an insertion operation in the modification operation, namely four pairs of signature/verification operations and a series of hash calculations. Compared with the modification operation of SDUS-AD, there is an additional pair of signature/verification operations. It can be seen that SDUS-AD improves the efficiency of the modification operation through the multi-copy association tree.

To test the changes in the insertion, modification, and deletion overhead under the condition that the number of data blocks to be updated changes, the experimental conditions are changed as follows: the number of file copies of SDUS-AD is 4, the file size is fixed at 64MB, the range of data blocks to be inserted is [20, 100], and the increment is 20; the range of data blocks to be modified is [10, 50], and the increment is 10; the range of the data block to be deleted is [10, 50], and the increment is 10. Figures 12, 13, and 14 show the insertion, modification, and deletion costs with the increased number of data blocks to be updated.

As shown in Figure 12, the insertion cost of SDUS-AD and scheme [39] increases linearly with the increase of data blocks to be inserted, and SDUS-AD is relatively higher than the scheme [39]. Analyzing the reason, SDUS-AD and scheme [39] need to encrypt the data block to be inserted before inserting, and SDUS-AD requires a key to encrypt a copy file, so SDUS-AD needs to perform encryption operations more times than the scheme [39]. Compared with scheme [39], SDUS-AD realizes the insertion operation of four copy files with an average increase of about 87% of the insertion overhead (If the scheme [39] performs the insertion operation under the same conditions, that is, under the four copy files, the time overhead is 400%.). This means that in multi-copy data storage, SDUS-AD improves the security at the cost of acceptable overhead. Figure 13 shows that the modification cost of both schemes increases with the increase of data blocks to be modified, and the time cost of the scheme [39] is slightly higher than that of SDUS-AD. Analyzing the reasons, the scheme [39] needs to perform a deletion operation and an insertion operation every time when a modification operation is performed. SDUS-AD only needs to find the index of the leaf node where the data block is to be modified and its copy blocks are located to realize the modification. SDUS-AD improves efficiency through the multi-

copy association tree and a more efficient modification algorithm under the multi-copy data. As shown in Figure 14, the deletion cost of both the two schemes increases linearly with the increase of data blocks to be deleted, and SDUS-AD has a higher time cost than the scheme [39]. SDUS-AD adds a pair of signature and verification operations in the deletion phase. Compared with the scheme [39], SDUS-AD achieves the deletion of four copies at the cost of an average increase of about 127%time cost. In addition, the service fog node completes the verification process of the deletion proof in SDUS-AD. Therefore, under the condition of multiple copies, SDUS-AD improves the security with acceptable overhead.

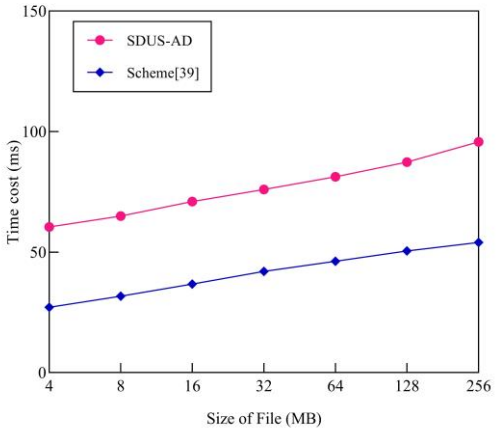

Figure 9. Time cost of insertion with different file sizes.

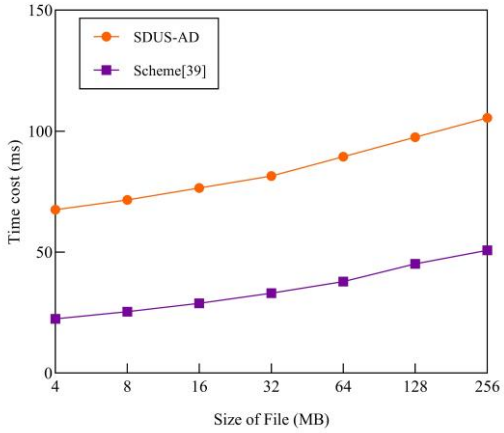

Figure 10. Time cost of deletion with different file sizes.

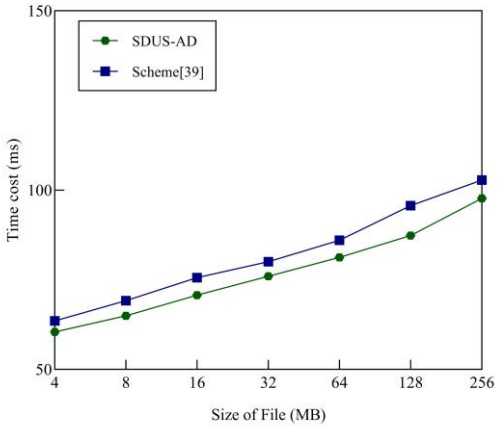

Figure 11. Time cost of modification with different file sizes.

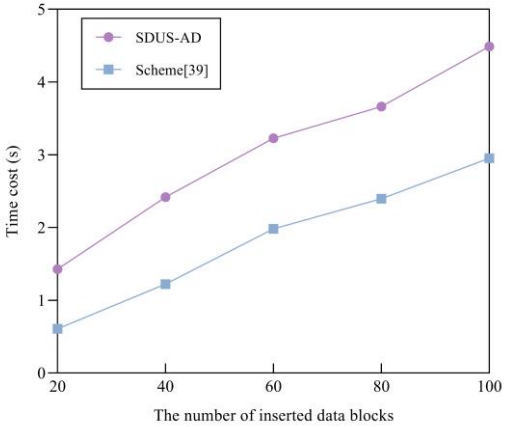

Figure 12. Time cost of insertion with different numbers of data blocks.

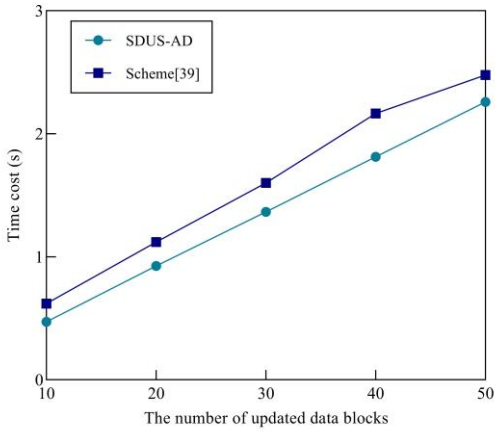

Figure 13. Time cost of modification with different numbers of data blocks.

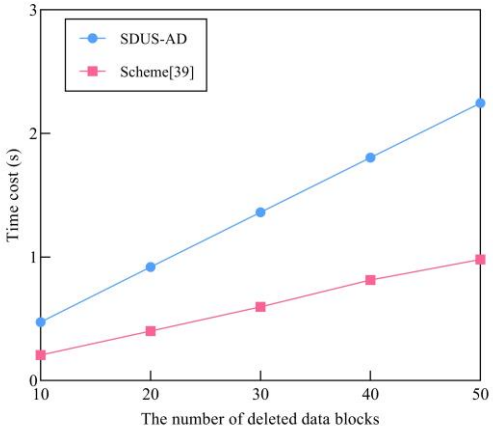

Figure 14. Time cost of deletion with different numbers of data blocks.

7.2 Multi-copy deletion overhead

To describe the multi-copy data deletion overhead of SDUS-AD in more detail, we compare SDUS-AD with the scheme ADM [35]. Figure 15 shows the change in deletion overhead as the size of the data file increases when the number of copy files is 8. It can be seen from Figure 15 that as the file size increases, the cost of both deletion increases, and the cost of SDUS-AD is lower than that of ADM. ADM performs blind RSA encryption, and a multiplication operation in the deletion phase, while SDUS-AD only needs to find the leaf node index where the data block to be deleted is located and performs several hash calculations. This also shows that SDUS-AD is more efficient under multi-copy data. Figure 16 shows the change in the deletion cost of SDUS-AD and ADM as the number of file copies increases when the data file size is fixed at 128 MB. It can be seen from the experimental results that the deletion cost of both two schemes increases linearly when the number of copies increases, and the deletion cost of ADM is higher than that of SDUS-AD. Since ADM needs to delete each copy file in turn and obtain the root of the new Merkle hash tree, SDUS-AD only needs to find the index of the leaf node where the data block to be deleted and its copy block are located to achieve safe data deletion, and the service fog node completes the verification process of the deletion proof, the data owner does not need to pay extra cost. Therefore, compared with ADM, SDUS-AD improves the efficiency of multi-copy data deletion.

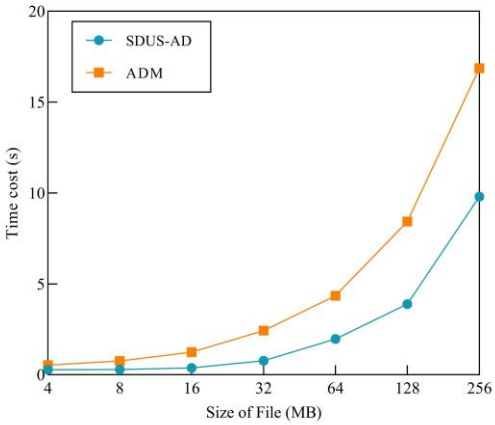

Figure 15. Time cost of deletion with different file sizes.

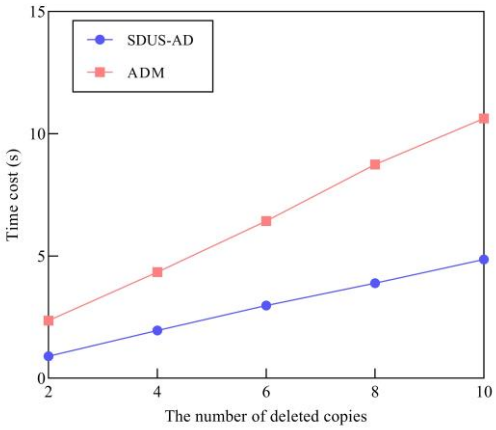

Figure 16. Time cost of deletion with different numbers of copies.

7.3 The overhead of SDUS-AD

Figure 17, 18, and 19 show the cost comparison of each entity in SDUS-AD in insertion, modification, and deletion operations. During the entire interaction process, the main cost of the data owner is the generation of data keys and update requests. The main cost of the fog is to encrypt data blocks and their copy blocks, construct the multi-copy association tree and perform a series of signature and hash calculations. In the traditional scheme, the initiation of update operations and the verification of update proofs are completed by the data owner alone, while in SDUS-AD, the fog bears this part of the cost for the data owner, and the data owner does not need to save the corresponding data locally, which reduces the storage overhead and computational overhead of the data owner. This is more in line with DO's original intention of using cloud storage services to reduce storage overhead and management burden.

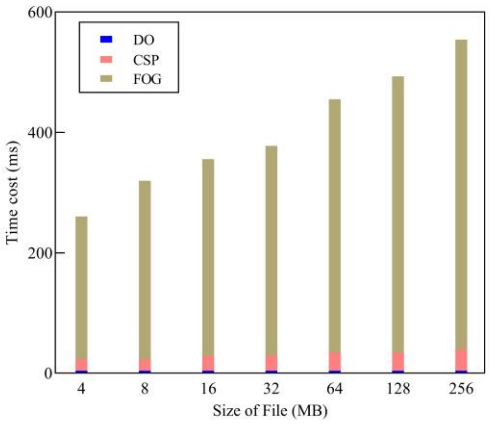

Figure 17. Time cost of insertion in SDUS-AD.

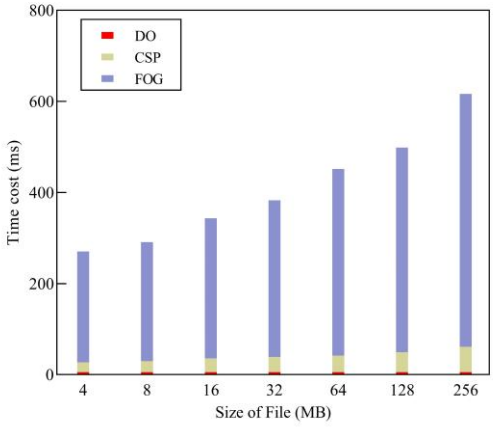

Figure 18. Time cost of modification in SDUS-AD.

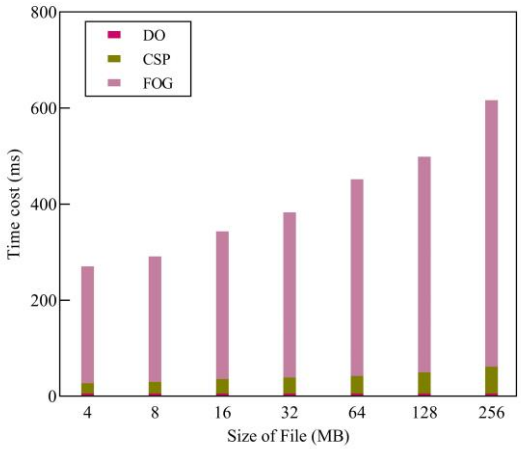

Figure 19. Time cost of deletion in SDUS-AD.

8. Conclusion

Assured deletion of cloud data is a crucial and challenging problem in cloud storage. This paper designs a novel multi-copy data deletion scheme for users with update requirements for some data under the multi-copy data. The update of multi-copy data is realized through a new dynamic tree structure called Multi-Copy Association Tree. In addition, SDUS-AD introduces blockchain technology, and smart contracts make the scheme traceable. Security analysis indicates that the scheme is sufficiently secure, functional analysis indicates that the function of the scheme is extensive, and the experimental results show that the scheme is efficient and feasible. However, the efficiency of searching a data block depends on the height of the multi-copy association tree, and the design of a more efficient and secure search algorithm will become our future work.

Declarations

Ethical Approval and Consent to participate

Not applicable

Human and Animal Ethics

Not applicable

Consent for publication

Not applicable

Availability of supporting data

The data sets supporting the results of this article are included within the article and its additional files.

Competing interests

The authors declare that they have no conflict of interest.

Funding

This work was supported by the Natural Science Foundation of Hebei Province (F2021201049) and the Key Project of Natural Science Foundation of Hebei Province (F2016201244).

Authors' contributions

Junfeng Tian: Conceptualization, Project administration, Funding acquisition, Supervision.
Ruxin Bai: Writing - Original draft preparation, Conceptualization, Methodology, Software, Formal analysis, Investigation.
Tianfeng Zhang: Conceptualization, Methodology, Formal analysis, Investigation.

Acknowledgements

This work was supported by the Natural Science Foundation of Hebei Province (F2021201049) and the Key Project of Natural Science Foundation of Hebei Province (F2016201244).

Authors' information

Junfeng Tian (1965-), male, comes from Baoding of Hebei province, Professor, Deputy Secretary General, Open System Specialized Committee of China Computer Federation; Young and Middle-aged Specialist with Outstanding Contribution in Hebei Province; Young and Middle-aged Teacher with Outstanding Contribution in Hebei Province; He graduated from Hebei University in 1986, received a master's degree in computer science from Xi'an University of Electronic Science and Technology in 1990-1991 and a master's degree in 1995. In 2004, he graduated from the University of Science and Technology of China with a major in computer science and technology. He has been engaged in teaching and research in the fields of distributed computing and network technology for many years.

Ruxin Bai (1998-), female, comes from Fucheng of Hebei province, who is a master student of Hebei University, the main research direction is information security and trusted computing.

Tianfeng Zhang (1998-), male, comes from Chengde of Hebei province, who is a master student of Hebei University, the main research direction is information security and trusted computing.

REFERENCES

[1] Buyya, R., Yeo, CS., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems. 25(6), 599-616 (2009).

[2] Yang, C., Ye, J.: Secure and efficient fine-grained data access control scheme in cloud computing. Journal of High Speed Networks. 21(4), 259-271 (2015).

[3] Shen, J., Zhou, T., Chen, X., Li, J., Susilo, W.: Anonymous and traceable group data sharing in cloud computing. IEEE Transactions on Information Forensics and Security. 13(4), 912-925 (2018).

[4] Han, S., Han, K., Zhang, S.: A data sharing protocol to minimize security and privacy risks of cloud storage in big data era. IEEE Access. 7, 60290-60298 (2019).

[5] Wang, Y., Tao, X., Ni, J., Yu, Y.: Data integrity checking with reliable data transfer for secure cloud storage. International Journal of Web and Grid Services. 14(1), 106-121 (2018).

[6] Dhal, K., Pattnaik, PK., Rai, SC.: RACC: an efficient and revocable fine grained access control model for cloud storage. International Journal of Knowledge-Based and Intelligent Engineering Systems. 23(1), 21-32 (2019).

[7] Zhang, J., Wang, B., He, D., Wang, X.: Improved secure fuzzy auditing protocol for cloud data storage. Soft Computing. 23(10), 3411-3422 (2019).

[8] Yang, T., Li, J., Yu B.: A Secure Ciphertext Self-Destruction Scheme with Attribute-Based Encryption. Mathematical Problems in Engineering. 2015, 1-8 (2015).
[9] Yi, S., Qin, Z., Li, Q.: Security and privacy issues of fog computing: a survey. Wireless Algorithms, Systems, and Applications. WASA 2015. Pp. 685-695 (2015).
[10] Guo, R., Zhuang, C., Shi, H., Zhang, Y., Zheng, D.: A lightweight verifiable outsourced decryption of attribute-based encryption scheme for blockchain-enabled wireless body area network in fog computing. International Journal of Distributed Sensor Networks. 16(2): 155014772090679 (2020).
[11] Jiang, J., Tang, L., Gu, K., Jia, W.: Secure computing resource allocation framework for open fog computing. The Computer Journal. 63(4), 567-592 (2020).
[12] Shahid, M., Hameed, A., Islam, S., Khattak, H., Din, I., Rodrigues, J.: Energy and delay efficient fog computing using caching mechanism. Computer Communications. 154, 534-541 (2020).
[13] Roman, R.，Lopez, J.，Mambo, M.: Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. Future Generation Computer Systems, 78, 680-698 (2018).
[14] Stojmenovic, I., Wen, S., Huang, X., Luan, H.: An overview of fog computing and its security issues. Concurrency and Computation: Practice and Experience. 28(10), 2991-3005 (2016).
[15] Huang, X., Xiang, Y., Bertino, E., Zhou, J., Xu, L.: Robust multi-factor authentication for fragile communications. IEEE Transactions on Dependable and Secure Computing. 11(6), 568-581 (2014).
[16] Choo, K., Domingo-Ferrer, J., Zhang, L.: Cloud cryptography: theory, practice and future research directions. Future Generation Computer Systems. 62, 51-53 (2016).
[17] BBC.: Data on 540 million Facebook users exposed. https://www.bbc.com/news/technology-47812470 (2019). Accessed 5 October 2019.
[18] ISC.: 2019 cloud security report. https://www.isc2.org/Resource-Center/Reports/Cloud-Security-Report (2019). Accessed 7 March 2020.
[19] Wang, Q., Zhou, F., Peng, S., Xu, Z.: Verifiable outsourced computation with full delegation. International Conference on Algorithms and Architectures for Parallel Processing. ICA3PP 2018. Pp. 270-287 (2018).
[20] Xu, J., Wei, L., Zhang, Y., Wang, A., Zhou, F., Gao, C.: Dynamic fully homomorphic encryption-based merkle tree for lightweight streaming authenticated data structures. Journal of Network and Computer Applications. 107, 113-124 (2018).
[21] Miao, M., Wang, J., Ma, J.: New publicly verifiable databases supporting insertion operation. 2015 18th International Conference on Network-Based Information Systems. Taipei, Taiwan. New York: IEEE. Pp. 2-4 (2015).
[22] Tian, J., Chang, F.: Trusted cloud platform management model based on TPM alliance. Journal on Communications. 37(2), 1-10 (2016).
[23] Hughes, G., Coughlin, T., Commins, D.: Disposal of Disk and Tape Data by Secure

Sanitization. IEEE Security & Privacy. 7(4), 29-34 (2009).
[24] Paul, M., Saxena, A.: Proof of erasability for ensuring comprehensive data deletion in cloud computing. Recent Trends in Network Security and Applications. CNSA 2010. Pp. 340-348 (2010).
[25] Perito, D., Tsudik, G.: Secure code update for embedded devices via proofs of secure erasure. The 15th European Symposium on Research in Computer Security. ESORICS 2010. Pp. 643-662 (2010).
[26] Luo, Y., Xu, M., Fu, S., Wang, D.: Enabling assured deletion in the cloud storage by overwriting. The 4th ACM international workshop on security in cloud computing. SCC '16. New York, NY, USA: ACM. Pp.17-23 (2016).
[27] Tang, Y., Lee, P., Liu, J., Perlman, R.: Fade: Secure overlay cloud storage with file assured deletion. Security and Privacy in Communication Networks. SecureComm 2010. Pp. 380-397 (2010).
[28] Li, H., Sun, W. Ll, F., Wang, B.: Secure and Privacy-Preserving Data Storage Service in Public Cloud. Journal of Computer Research and Development. 51(7), 1397-1409 (2014).
[29] Boneh, D., Lipton, R.: A Revocable Backup System. The sixth USENIX Security Symposium. San Jose, CA, USA. Pp. 91-96 (1996).
[30] Perlman, R.: File system design with assured delete. Third IEEE International Security in Storage Workshop. SISW'05. San Francisco, CA, USA. Pp. 83-88 (2005).
[31] Mo, Z., Xiao, Q., Zhou, Y., Chen, S.: On deletion of outsourced data in cloud computing. IEEE 7th International Conference on Cloud Computing. Pp. 344-351 (2014).
[32] Yu, Y., Ni, J., Wu, W., Wang, Y.: Provable data possession supporting secure data transfer for cloud storage. The 10th international conference on broadband and wireless computing, communication and applications. BWCCA. New York, NY, USA: IEEE. Pp.38-42 (2015).
[33] Xue, L., Ni, J., Li, Y., Shen, J.: Provable data transfer from provable data possession and deletion in cloud storage. Computer Standards & Interfaces. 54, 46-54 (2017).
[34] Liu, Y., Xiao, S., Wang, H.: New provable data transfer from provable data possession and deletion for secure cloud storage. International Journal of Distributed Sensor Networks. 15(4), 1-12 (2019).
[35] Dulin., Zhang, Z., Tan, S., Wang, J., Tao, X.: An Associated Deletion Scheme for Multi-copy in Cloud Storage. International Conference on Algorithms and Architectures for Parallel Processing. ICA3PP 2018. Pp. 511-526 (2018).
[36] Yang, C., Chen, X., Xiang, Y.: Blockchain-based publicly verifiable data deletion scheme for cloud storage. Journal of Network and Computer Applications. 103, 185-193 (2018).
[37] Yang, C., Wang, J., Tao, X., Chen, X.: Publicly verifiable data transfer and deletion scheme for cloud storage. The 20th international conference on information and communications security. ICICS 2018. Pp. 445–458 (2018).
[38] Yang, C., Liu, Y., Tao, X.: Assure deletion supporting dynamic insertion for

outsourced data in cloud computing. International Journal of Distributed Sensor Networks. 16(9), (2020).
[39] Miao, M., Ma, J., Huang X., Wang, Q.: Efficient Verifiable Databases With Insertion/Deletion Operations From Delegating Polynomial Functions. IEEE Transactions on Information Forensics and Security. 13(2), 511-520 (2018).
[40] Wang, Q., Zhou, F., Xu, J., Xu, Z.: Efficient verifiable databases with additional insertion and deletion operations in cloud computing, Future Generation Computer Systems. 115, 553-567 (2021).
[41] Merkle, R.: A certified digital signature. Conference on the Theory and Application of Cryptology. CRYPTO 1989. New York, NY. Pp. 218–238 (1989).
[42] ANSI X9.62, FIPS 186-2.: Elliptic Curve Digital Signature Algorithm. (1998).
[43] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. The Thirteenth EuroSys Conference. EuroSys '18. New York, NY, USA: ACM. Pp. 1-15 (2018).