Hindawi
Mobile Information Systems
Volume 2021, Article ID 9977744, 17 pages
https://doi.org/10.1155/2021/9977744

Research Article

Enabling Decentralized and Dynamic Data Integrity Verification for Secure Cloud Storage via T-Merkle Hash Tree Based Blockchain

Kai He,[1,2] **Chunxiao Huang**,[1,2] **Jiaoli Shi**,[3] **Xinrong Hu**,[1,2] **and Xiying Fan**[4]

[1]*School of Math and Computer Science, Wuhan Textile University, Wuhan, China*
[2]*Hubei Clothing Information Engineering Technology Research Center, China*
[3]*School of Information Science and Technology, Jiujiang University, Jiujiang, China*
[4]*School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China*

Correspondence should be addressed to Jiaoli Shi; shijiaoli@whu.edu.cn

Received 23 March 2021; Revised 14 July 2021; Accepted 22 September 2021; Published 1 November 2021

Academic Editor: L. J. García Villalba

Cloud storage provides elastic storage services for enterprises and individuals remotely. However, security problems such as data integrity are becoming a major obstacle. Recently, blockchain-based verification approaches have been extensively studied to get rid of a centralized third-party auditor. Most of these schemes suffer from poor scalability and low search efficiency and even fail to support data dynamic update operations on blockchain, which limits their large-scale and practical applications. In this work, we propose a blockchain-based dynamic data integrity verification scheme for cloud storage with T-Merkle hash tree. A decentralized scheme is proposed to eliminate the restrictions of previous centralized schemes. The data tags are generated by the technique of ZSS short signature and stored on blockchain. An improved verification method is designed to check the integrity of cloud data by transferring computation from a verifier to cloud server and blockchain. Furthermore, a storage structure called T-Merkle hash tree which is built based on T-tree and Merkle hash tree is designed to improve storage utilization of blockchain and support binary search on chain. Moreover, we achieve efficient and secure dynamic update operations on blockchain by an append-only manner. Besides, we extend our scheme to support batch verification to handle massive tasks simultaneously; thus, the efficiency is improved and communication cost is reduced. Finally, we implemented a prototype system based on Hyperledger Fabric to validate our scheme. Security analysis and performance studies show that the proposed scheme is secure and efficient.

1. Introduction

Nowadays, more and more companies have built their cloud computing services and open them to individuals or other enterprises, for instance, Amazon, Alibaba, Tencent, and Microsoft. As an important service of cloud computing, cloud storage allows clients remotely to store their data in cloud. By data outsourcing, clients enjoy many benefits, such as relieving themselves of heavy storage management, unlimited access at any time and any place, reducing expenditure on hardware/software, and employee maintenances. However, storing data in cloud makes clients lose local control over their data, which may cause potential security risk. One big problem is how to make sure that the integrity of outsourced data is intact. As we know, data loss or corruption with cloud servers often occurs due to malicious attacks, hardware failures, insider attacks, and even human mistakes [1–3].

Public verification has been extensively studied to verify the integrity of cloud data in recent years. A third-party auditor (TPA) is introduced to verify data integrity on behalf of clients periodically without local copies. The key ideal is that each data block is attached with a tag or signature, and the integrity verification depends on the correctness of these tags or signatures. During verification, the TPA sends a query with some random sampled data blocks to cloud server and then the cloud server calculates the proofs using the queried data and tags stored on it and respond them to

the TPA. Finally, the TPA checks these proofs to judge the integrity of cloud data. The benefits and basic requirements of public verification in cloud storage have been discussed in previous scenarios [4, 5].

However, public verification is still subject to a series of restrictions. First, the assumption of complete trust on TPA is impractical because centralized TPA is more vulnerable to internal and external security threats from the Internet. Second, the TPA may cheat clients for profits by conspiring with cloud server to generate fake verification results. Third, when the received tasks exceed its processing capacity, the TPA has to delay the completion of previously agreed tasks. Therefore, the TPA cannot be absolutely trusted and may turn into a bottleneck of the system [6, 7].

Fortunately, blockchain technology provides a new perspective to dispose of the above problems for the properties of decentralized data storage, point-to-point transmission, consensus mechanism, and encryption [8]. Nevertheless, designing a decentralized verification scheme based on blockchain without TPA is a great challenge. If blockchain is used to store clients' data, the data tags or signatures are not required [9]. It may limit its extensive applications because the structure of blockchain has a major obstacle in terms of capacity and scalability. Besides, it is not convenient for future data access and sharing. To overcome this drawback, Wang et al. [10] proposed a private PDP scheme to check remote data integrity by using blockchain technology. They used blockchain to store data tags while data files are still stored in cloud. However, their scheme should iterate through blockchain to obtain the challenged tags during verification, which is inefficient and impractical when blockchain grows large.

Another major concern is that data dynamic operations have not been supported in previous blockchain-based schemes. Clients may not only access but also need to update cloud data, e.g., data modification, deletion, and insertion. Unfortunately, blockchain-based verification schemes mainly pay attention on static data files. Because blockchain is tamper-proof, a block cannot be modified once it is formed. This seems to make data dynamic operations difficult to implement. Thus, how to achieve blockchain-based data integrity verification and support data dynamic operations is necessary and valuable, which needs further exploration.

In consideration of the key points of integrity verification and data dynamics for large-scale cloud storage, we propose a decentralized and dynamic integrity verification scheme with blockchain to check data integrity without requiring TPA and support fast retrieval. The main contributions can be summarized as follows.

First, we present a decentralized cloud storage verification framework. Our scheme uses blockchain to overcome the obstacles brought by TPA and enhances the reliability of verification result. Data tags are calculated by the technique of ZSS short signature [11], and a new verification method is proposed to improve efficiency by transferring computation from a verifier to cloud server and blockchain.

Second, to reduce storage overhead and improve search efficiency of blockchain, we propose a new storage structure called T-Merkle hash tree which is built based on T-tree and Merkle hash tree and design its search algorithm.

Third, we extend the proposed scheme to support data dynamic operations, which is not considered by most existing blockchain-based schemes. In addition, our scheme achieves batch verification, which can handle massive verification tasks from different clients or for different data files at once.

Fourth, we validate the correctness and performance of our scheme by implementing a prototype system. Detailed security deduction shows that our scheme is secure, and experiment results demonstrate the efficiency of our scheme.

1.1. Organization. In the rest of this work, we introduce related work in Section 2 and discuss the system model in Section 3. We design a verification scheme in Section 4, analyze the security of our scheme in Section 5, and further evaluate our scheme by simulations in Section 6. Section 7 concludes the work.

2. Related Work

2.1. Centralized Data Integrity Verification. According to the adopted technologies, data integrity verification approaches can be divided into two kinds: Proofs of Retrievability (POR) and Provable Data Possession (PDP). Juels and Kaliski [12] presented a proof of retrievability (POR) scheme to protect the integrity of remote data. They used the technologies of spot-checking and error correcting code to ensure the retrievability and ownership of files on archive service systems. Ateniese et al. [13] defined a framework of provable data possession (PDP) model to prove the possession of data files stored on untrusted servers. Wang et al. [4] utilized a third-party auditor to verify the integrity of cloud data on behalf of clients. Based on these works, lots of works have been conducted in public verification to address the issues of privacy protection, data dynamic, batch processing, and so on. Wang et al. [14] proposed a dynamic PDP scheme based on Merkle hash tree. Yang and Jia [15] encrypted the proofs to preserve the data privacy against TPA during verification and utilized hash index to realize efficient data dynamic update. They also extended their verification method to support batch verification in multicloud scenario.

Afterwards, lots of third-party verification schemes have been studied, mainly taking into account data sharing, deduplication, availability or reliability, pretended or disguised third-party auditors, etc. Wang et al. [16] and Yuan and Yu [17] studied the revocable integrity verification on shared data. Chen and Lee [18] presented a verification scheme for cloud data based on regenerating code, which allows a client to check the integrity of outsourced data and repair the corrupted data. Liu et al. [19] proposed a public verification scheme to eliminate the threat of untrusted auditor to issue unauthorized audit challenges. The dynamic data update of their scheme operates on variable size file blocks and is fine grained rather than block level.

2.2. Blockchain-Based Data Integrity Verification. Recently, many blockchain-based data integrity verification schemes have been put forward to ensure the integrity of

outsourced data or multimedia data [20, 21]. To resolve the defects of uncredible TPA, Xue et al. [22] proposed a public verification scheme against malicious auditors based on blockchain technology. However, it fails to ensure that the audit tasks are performed on time. Zhang et al. [23] presented a certificateless verification scheme using blockchain, which can avoid malicious and delayed auditors, but has high computing cost and does not handle the case of dynamic data update.

By avoiding relying on TPA, Liu et al. [24] presented a data integrity scheme to check based on blockchain for IoT service. However, the basic functions of their scheme are only effective for small-scale scenarios. Liang et al. [25] proposed an architecture for data provenance which involves inserting the provenance information into transactions on blockchain to prove the integrity of cloud data. Wang et al. [26] designed a decentralized scheme to solve the problem of untrustable single point and allow clients to record the trace of their history data. To reduce the computation and communication overhead in large-scale Internet of Things, Wang and Zhang [27] proposed a blockchain-based integrity verification scheme using multiple chains. However, they increased the storage space of blockchain. Yu et al. [28] presented a decentralized verification scheme for big data in smart urban environments; they employed a data auditing blockchain (DAB) to store auditing evidences. Similarly, Huang et al. [29] proposed a blockchain-based cooperative auditing framework for cloud data storage, which recorded auditing requests and results on blockchain in perpetuity. Miao et al. [30] proposed a blockchain-based public verification scheme to achieve decentralization and privacy protection. They employed blockchain to generate challenge request and recorded the process of verification onto the blockchain. However, all of these schemes use blockchain to validate verification results afterwards and cannot check whether the data are integrated in time.

Additionally, data dynamic operations have been widely studied in centralized verification schemes. In the blockchain-based solutions, Wang and Zhang [27] realized block-level dynamic update operations including appending, modification, and deletion. Yu et al. [28] proposed a modified Merkle hash tree to achieve data dynamic update. Yang et al. [31] supported dynamic update by using dynamic hash table and modification record table. These dynamic and blockchain-based verification schemes only use blockchain to record verification process or result, and the dynamic data update is basically independent of blockchain. Nevertheless, our solution makes use of blockchain to store data tags, which makes dynamic update operations more difficult.

In summary, most existing blockchain-based data integrity verification schemes focus on removing centralized TPA and recording verification process or results on blockchain. In our work, we use blockchain to store data tags and deal with the query efficiency of blockchain and together with data dynamic update operations on blockchain.

Part of this work has previously appeared as an extended version in a conference [32]. We revise and extend the work mainly from five aspects compared with [32]. First, we analyze the impact of data dynamic operations on the blockchain-based system and extend our scheme to dispose of secure data dynamic operations. Second, we extend our verification method to support batch processing, by which massive tasks can be handled concurrently and efficiently. Third, we propose a blockchain optimization method which is not discussed in conference paper. Fourth, we add experiments to validate the performance of data dynamic operation and batch verification.

3. Definitions and Preliminaries

3.1. System Model. The blockchain-based storage verification system involves three kinds of participants: Client, Cloud Server (CS), and Blockchain (BC), as presented in Figure 1. Client intends to outsource a large amount of data to the cloud, and it can be either a company or an individual consumer. Cloud Server is built and maintained by cloud service providers, who have sufficient storage space and computational resources to serve Client's demands and should prepare for the integrity check. Blockchain stores Client's data tags or metadata and is used to check the integrity of cloud data.

The framework of our scheme includes two stages: initialization stage and verification stage.

> *Initialization Stage*. (1) Client splits the encrypted data file into a set of equal sized blocks. (2) Client signs the blocks by computing a data tag for each block of data. (3) Client outsources the set of data blocks to CS. (4) Client uploads the data tags to BC rather than CS. (5) CS generates a challenge to BC. (6) BC answers with a tag proof based on the challenge, and then CS checks the integrity of data which will be stored on itself.
>
> *Verification Stage*. (7) Client randomly constructs a challenge, which includes an index subset of data blocks and a series of random values, and sends the challenge to CS and BC, respectively. (8) As proof providers, CS and BC respond to the data proof and tag proof after receiving the challenge. (9) Client checks the received proofs. If the verification passes, it indicates that the cloud data are integrated; otherwise, the cloud data are lost or corrupted.

Because of the security assurance of blockchain, the hash value of root or data tags stored on BC cannot be corrupted, which makes integrity verification results more credible. For convenience of description, we use Client as the verifier. Actually, the verification stage can be launched by any entity, i.e., Client, CS, BC, data sharers, or other third parties. Anyone with the public parameters can perform sampling verification on a regular basis or when accessing cloud data.

3.2. Threat Model and Design Goals. To protect privacy, data file is encrypted before uploading to CS by Client. Following the security model defined in [12–14], we consider two different kinds of threats: semitrusted or untrusted CS and malicious Client. CS may not be in accordance with the contract to store Client's data and hide data loss or even discard the corrupted data to deceive Client for reputation.

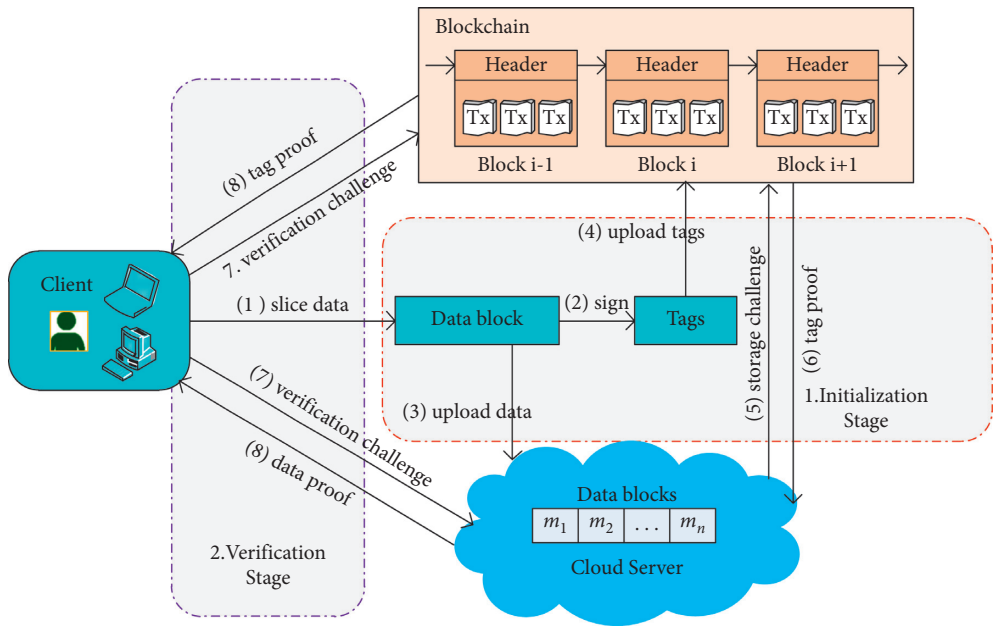

Figure 1: Blockchain-based verification model for cloud storage.

As we known, data loss or corruption can be caused by incidents such as hardware failure, management errors, and internal and external attacks. Client may deliberately outsource corrupted data or tags to defraud CS to obtain compensation. As a result, they may carry out the following types of attacks:

Forgery attack: CS forges data proofs with the intention of cheating verification

Replace attack: CS replaces the challenged data blocks with old versions or other uncorrupted data blocks with the intention of passing the verification

Fraud attack: Client uploads corrupted data blocks to CS or incorrect data tags to BC and then claims the data are intact to defraud CS for compensation

The proposed verification scheme is intended to achieve the following objectives:

Decentralized verification: data blocks and tags should be stored on different parties rather than single point. The integrity of data can be checked by any entity publicly without relying on centralized TPAs

Low cost of blockchain storage: the storage cost of data tags stored on BC should be as minimum as possible

Efficient blockchain query: the proposed scheme enables binary search to quickly find the challenged tags without traversing the entire BC

Data dynamic update: cloud data and tags on BC can be updated without introducing new security threats

Batch verification: the verifier can handle massive verification tasks simultaneously and efficiently

Verification efficiency: the computational and communication overhead of verification method should be as low as possible to meet practical application

3.3. Preliminaries

3.3.1. Blockchain. Blockchain is a point-to-point trading system which achieves decentralized, transparent, tamper-proof, and traceable features through the techniques of distributed consensus, data encryption, Merkle hash tree (MHT), and so on. System nodes do not need to trust each other. Each data or operation is recorded as a transaction. Multiple transactions form a block, and many blocks are linked together to form a blockchain. Each block has two parts: blockchain header and blockchain body. The header consists of version, hash value of the previous block, timestamp (the time block is created), Merkle root value, target, and nonce (a counter). To improve query efficiency, a field which stands for tag index range of current block is inserted into header in our design. All data tags are stored in blocks, and the integrity can be maintained by the Merkle hash root of each block, as shown in Figure 2. Since Merkle hash tree only stores data tags in leaf nodes, when the number of tags is large, the data structure will grow huge and the efficiency of tag query will decrease. To address this problem, we build the blockchain body through the improved Merkle hash tree called T-Merkle hash tree described in the following section.

3.3.2. T-Tree. T-tree is a balanced binary tree, and each node keeps multiple data value items [33]. Figure 3 shows the structure of a T-tree. A T-node contains several data fields, a parent pointer, and two child pointers pointing to its right and left subtrees separately. Generally speaking, the number of data values is kept smaller than the data fields' size of a node for the purpose of efficient data insertion. When data are inserted into a full node, the node will split into two nodes. Both data insertion and deletion may cause tree rotation to achieve balance.

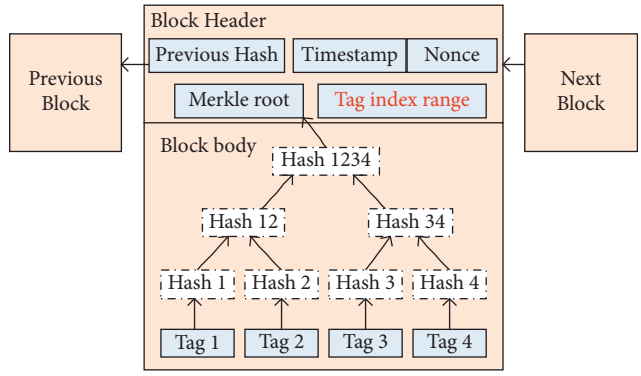

Figure 2: Data tag blockchain.

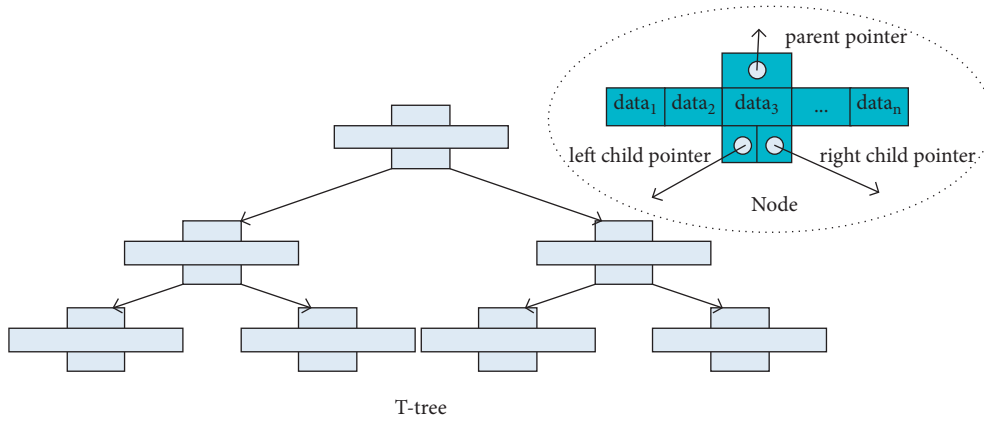

Figure 3: Structure of T-tree.

3.3.3. Bilinear Mapping. Assuming that G_1 is a gap Diffie–Hellman group and G_2 is a multiplicative cyclic group with prime order p, P is a generator of G_1. The bilinear mapping e: $G_1 \times G_1 \longrightarrow G_2$ has some properties. (1) Bilinearity: $e(aP, bQ) = e(P, Q)^{ab}$ and $e(P + R, Q) = e(P, Q) \cdot e(R, Q)$ for all $P, Q, R \in G_1$ and $a, b \in Z_p$. (2) Computability: there is an effective algorithm to calculate $e(P, Q)$ for all $P, Q \in G_1$. (3) Nondegeneracy: $e(P, P) \neq 1$.

For ease of reference, the main symbols we used in this work are summarized in Table 1.

Table 1: Notation.

Notation	Description
Client	The uploader or owner of cloud data
CS	Cloud server
BC	Blockchain
v_i	A T-Merkle hash tree node
$H(v_i)$	Hash value of node v_i
k	Number of data tags in a node
$h, \mathcal{H}$	A secure hash function:h: $\{0,1\}^* \longrightarrow Z_q^*$
p	A large prime number
G_1	A gap Diffie–Hellman (GDH) group
G_2	A multiplicative cyclic group
P	A generator of group G_1
e	A bilinear map:e: $G_1 \times G_1 \longrightarrow G_2$
sk	The client's private key
pk	The client's public key
F, m_i	Data file and data block, $i \in [1, n]$
T, Tag_i	Tag set and data tag, $i \in [1, n]$
DP	The data proof generated by CS
TP	The tag proof generated by BC

4. The Proposed Blockchain-Based Data Integrity Verification Scheme

In this section, first, we design the structure of T-Merkle hash tree. Second, we present the decentralized data integrity verification scheme based on blockchain. Then, dynamic verification and batch verification are carried out, respectively. Finally, we analyze our proposal from several aspects.

4.1. Structure of T-Merkle Hash Tree. In order to decrease storage space and improve query efficiency, we modify the Merkle hash tree from three aspects: (1) data tags are stored on each node instead of only on leaf nodes, and each node stores multiple data tag items from small to big; (2) index key is attached to each data tag to support efficient tag query; and (3) the index range field indicating the minimum and maximum index values of the current node will be embedded into the header of each block.

Definition 1. A T-Merkle hash tree is built based on T-tree and Merkle hash tree. Figure 4 shows the node structure. A node v_i contains minimum index key Min_i, maximum index key Max_i, an index key and tag set $\{j, tag_j\}_{j\in[k]}$, and hash value $H(v_i)$. k means the number of data tags in each node.

The hash value $H(v_i)$ of node v_i is calculated by the hash value of data tags in node v_i and the hash values of its children, namely,

$$H(v_i) = \begin{cases} h(v_i), & \text{leaf node,} \\ h(h(v_i)\|H(r\text{child})), & \exists \text{ right child,} \\ h(h(v_i)\|H(l\text{child})), & \exists \text{ left child,} \\ h(h(v_i)\|H(r\text{child})\|H(l\text{child})), & \exists \text{ two children,} \end{cases} \tag{1}$$

where $h(v_i)$ equals to $h(h(\mathrm{tag}_1)\|h(\mathrm{tag}_2)\|\ldots\|h(\mathrm{tag}_k))$, in which h is a secure hash function and $\|$ is a concatenation operation.

To build the T-Merkle hash tree, we firstly generate a basic T-tree with data tags and index keys and then calculate the hash value $H(v_i)$ of each node from bottom to root. Figure 5 gives an instance of constructing a T-Merkle hash tree through 14 data tags with index key from 1 to 14. Each node keeps two data tags, which means $k = 2$. When the verifier wants to authenticate $\{\mathrm{Tag}_5, \mathrm{Tag}_6\}$, the BC sends the verifier with the authentication information $\Omega = <H(c), H(b), h(\mathrm{Tag}_3), h(\mathrm{Tag}_4), h(\mathrm{Tag}_7), h(\mathrm{Tag}_8)>$. The verifier can check $\{\mathrm{Tag}_5, \mathrm{Tag}_6\}$ by calculating $H(a) = h(h(\mathrm{Tag}_5)\|h(\mathrm{Tag}_6)\|H(c)\|h(\mathrm{Tag}_3)|h(\mathrm{Tag}_4))$, $H'(\mathrm{root}) = h(h(\mathrm{Tag}_7)\|h(\mathrm{Tag}_8)\|H(a)\|H(b))$ and then check whether the calculated $H'(\mathrm{root})$ is equal to the Merkle hash root $H(\mathrm{root})$ stored in block header. From the above statement, it can be seen that the authentication feature of Merkle hash tree is retained in T-Merkle hash tree.

When querying a data tag with index key Qkey, we iterate through the blockchain from the last block to the first block so as to compare the query key Qkey with the index range field in the block header. If index key Qkey is in the one block, binary search is used to find the queried data tag in T-Merkle hash tree stored in block body. Algorithm 1 describes the algorithm of searching on T-Merkle hash tree. Figure 5 shows an example; suppose that we are going to look for a tag with index key 5. First, we compare 5 with minimum index key 7 and maximum index key 8 of root node. Because 5 is less than 7, we will go through its left subtree. Second, as 5 is greater than the node $a's$ maximum index key of 4, we then go through its right child. Finally, Tag_5 is obtained in node d. With our careful design, the T-Merkle hash tree based blockchain not only ensures that the tags can be quickly authenticated in the block but also improves the efficiency of query by making full use of binary tree.

4.2. Construction of the Proposed Scheme. In our scheme, data tags are calculated by ZSS short signature [11] and stored on BC. The ZSS scheme uses bilinear pairings and general cryptographic hash functions such as SHA-1 or MD5 instead of requiring special hash functions such as BLS. Besides, it is faster than BLS signature because the verification process requires fewer pairing operations.

The construction of the proposed scheme includes two stages: initialization stage and verification stage.

4.2.1. Initialization Stage. First, Client generates a random value $sk \in Z_p$ as its private tag key and calculates the public key $pk = skP$. Under the assumption of Inv-CDHP, the private key cannot be extracted from the public key.

Second, Client splits the encrypted data file F into n blocks as $F = \{m_1, m_2, \ldots, m_n\}$ and then calculates a data tag Tag_i for each block m_i as

$$\mathrm{Tag}_i = \frac{1}{\mathscr{H}(m_i) + sk} P, \tag{2}$$

where $\mathscr{H}$ is a general hash function such as MD5 or SHA-1. The tag collection of data file F is $T = \{\mathrm{Tag}_1, \mathrm{Tag}_2, \ldots, \mathrm{Tag}_n\}$.

Finally, Client outsources the data file F to CS and uploads the tag collection T to BC. Client removes local data file and tags. Data tags are organized on BC through T-Merkle hash tree. CS will check the integrity of blocks before accepting the outsourced data to prevent malicious Client. The check process is similar to the verification stage described below. The process of initialization stage is shown in Figure 6.

4.2.2. Verification Stage. Client (as verifier) chooses a set of random elements $I = \{s_1, s_2, \ldots, s_c\}$, where c is a subset of $[1, n]$, and then generates a pseudorandom value u_i in Z_p for each s_i. Client sends $\mathrm{Chall} = \{i, u_i\}_{i\in I}$ to CS and BC separately.

After receiving Chall, CS calculates the data proof DP by encrypting it with bilinear map as

$$\mathrm{DP} = e\left(\sum_{i\in I} u_i \mathscr{H}(m_i) P, P\right). \tag{3}$$

Simultaneously, BC finds the challenged tags by using Algorithm 1 and then calculates the encrypted tag proof as

$$\mathrm{TP} = e\left(\sum_{i\in I} \frac{u_i}{\mathrm{Tag}_i} P^2, P\right). \tag{4}$$

CS and BC return $\{DP, TP\}$ as proofs to Client, respectively.

After receiving data proof DP and tag proof TP, Client firstly computes $R = \sum_{i\in I} u_i pk$ with the issued $\mathrm{Chall} = \{i, u_i\}_{i\in I}$ and public key pk and then verifies the proofs by validating the following equation:

$$\mathrm{TP} = DP \cdot e(R, P). \tag{5}$$

If equation (5) holds, the data file on CS is intact; otherwise, it is corrupted. Figure 7 illustrates the process of verification stage.

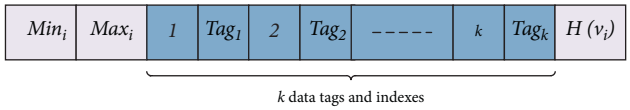

Figure 4: Node structure of T-Merkle hash tree.

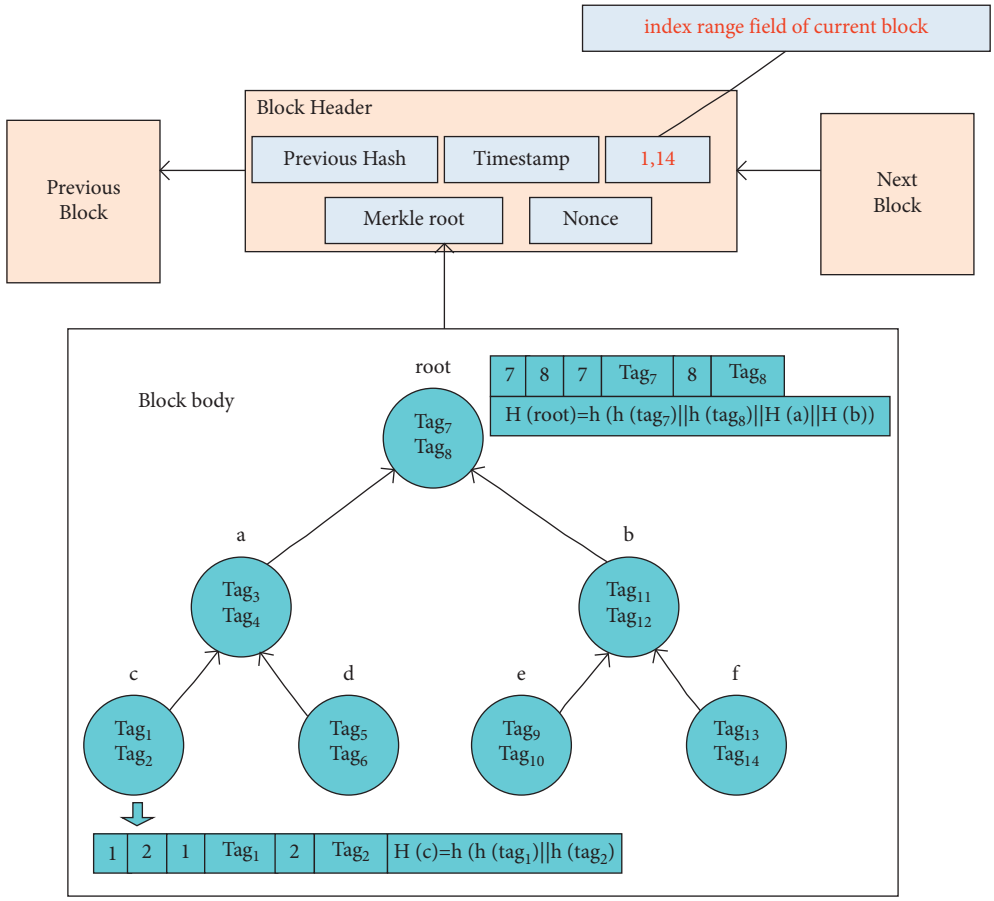

Figure 5: Example of T-Merkle hash tree.

Input: Qkey
Output: data tag Tag_{Qkey}
(1) **for each** block **in** BC
(2) **if** Qkey **in** index range of block **then**
(3) Access T-Merkle hash tree root of current block, set p = root.
(4) **if** p[Min] ≤ Qkey ≤ p[Max] **then**
(5) get Tag_{Qkey} in node p by comparing index key.
(6) **else if** Qkey < p[Min] **then**
(7) set p = p[leftchild].
(8) **else**// Qkey > p[Max]
(9) set p = p[rightchild].
(10) **else**
(11) go through the previous block.
(12) **end**

Algorithm 1: Search algorithm of T-Merkle hash tree.

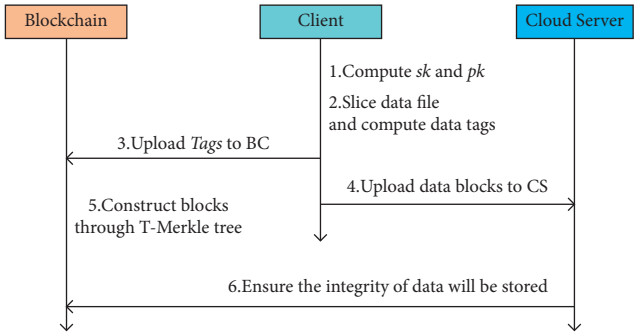

Figure 6: Process of initialization stage.

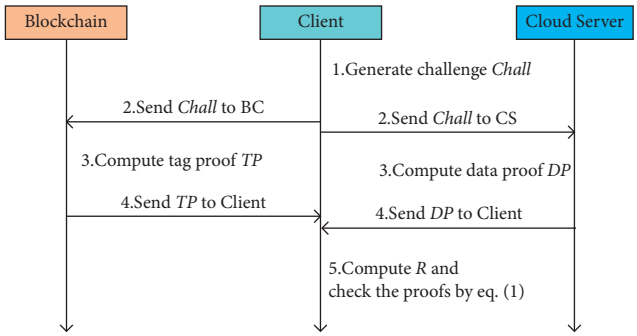

Figure 7: Process of verification stage.

4.3. Support for Data Dynamic Operations. In this work, data dynamic operations refer to block-level operations, including data insertion, data deletion, and data modification. The dynamic operations of data blocks will cause the recalculation of data tags. However, in our design, data tags are stored on blockchain. Because of the nontampering property of blockchain, it seems impossible to update the data tags. To solve this problem, we add the new update tags in the latest block on the blockchain, whether inserting or modifying data block. From this point of view, our data block update can be achieved by data appending.

When preparing to insert a new block m_i before m_j, the index of block m_i can be computed as $((j-1)+j)/2$; for instance, a new data block m_i is going to be inserted between m_1 and m_2, and the index of new inserted data block m_i is $i = (1+2)/2 = 3/2$. Therefore, there is no need to change the block index for subsequent blocks [34]. In addition, each dynamic operation will cause the hash root of current block to be recalculated.

We will start by showing how our solution handles data tag updates. When data tag is updated, it is inserted into the root node of a new block on blockchain. If the inserted key is smaller than or equal to the minimum key of root node, it will be inserted into the left subtree. If it is greater than the maximum key of root node, it is inserted into the right subtree. According to the update rules of T-tree, there are several cases when T-Merkle hash tree is updated.

(a) When root node is empty, a new node is created to store the update data tag and index key. The root hash value is equal to the node hash value. Figure 8(a) shows this case.

(b) When the update index key is between the minimum and maximum index key of a node and the node is not full, the update tag and index key are inserted into this node in sequence. This is shown in Figure 8(b).

(c) When the update index key is between the minimum and maximum index key of a node and the node is full, it causes the node to split. The tag item with maximum index key is then moved to the right child, while the tag and index key are inserted into the node. This case is shown in Figure 8(c).

(d) If the update index key is smaller than the minimum value of a node, the tag will be stored on the left child. Otherwise, it will be stored on the right child. The result is shown in Figure 8(d).

For tag insertion or modification, it may cause the hash value from the updated node to root node on the path to be recalculated. Since the inserted or modified index keys are unpredictable, it may also cause the nodes to be split. For deletion, we delete the data blocks directly and the tag items in blockchain have no effect. Dynamic updates may also cause the tree to rotate to maintain balance, which is shown in Figure 8(e).

Now, we construct the data dynamic operations in detail, which includes three steps: UpdateRequest, UpdateCommit, and UpdateVerify. Each update operation can be constructed as an update request to CS $\mathrm{CSReq} = \{op, i, m_i^*\}$ and an update request to BC $\mathrm{BCReq} = \{op, i, \mathrm{tag}_i^*\}$, where OP is dynamic update operation type, such as data block insertion (I), deletion (D), and modification (M).

4.3.1. UpdateRequest Step. For data block modification, suppose Client prepares to modify the ith block m_i into m_i^*. Client (1) computes the new tag as $\mathrm{Tag}_i^* = 1/\mathscr{H}(m_i^*) + skP$, (2) uploads $\mathrm{BCReq} = \{M, i, \mathrm{tag}_i^*\}$ to BC, and (3) uploads the request of modified block $\mathrm{CSReq} = \{M, i, m_i^*\}$ to CS.

For data block insertion, suppose Client wants to insert block m_i^* before m_j. Client (1) calculates the new index as $i = (j-1+j)/2$, (2) calculates a new tag as $\mathrm{Tag}_i^* = 1/\mathscr{H}(m_i^*) + skP$, (3) uploads $BCReq = \{I, i, tag_i^*\}$ to BC, and (4) uploads the request of new inserted block $\mathrm{CSReq} = \{I, i, m_i^*\}$ to CS.

For data block deletion, it refers to deleting the specified blocks logically, for which it is not necessary to move all the subsequent blocks forward. Client only sends $\mathrm{CloudReq} = \{D, i\}$ to CS.

4.3.2. UpdateCommit Step. After receiving the request CSReq, CS commits the update operations according to the type of data operation.

For data block modification, CS replaces the block m_i with m_i^*

For data block insertion, CS stores the new block m_i^* in the ith position

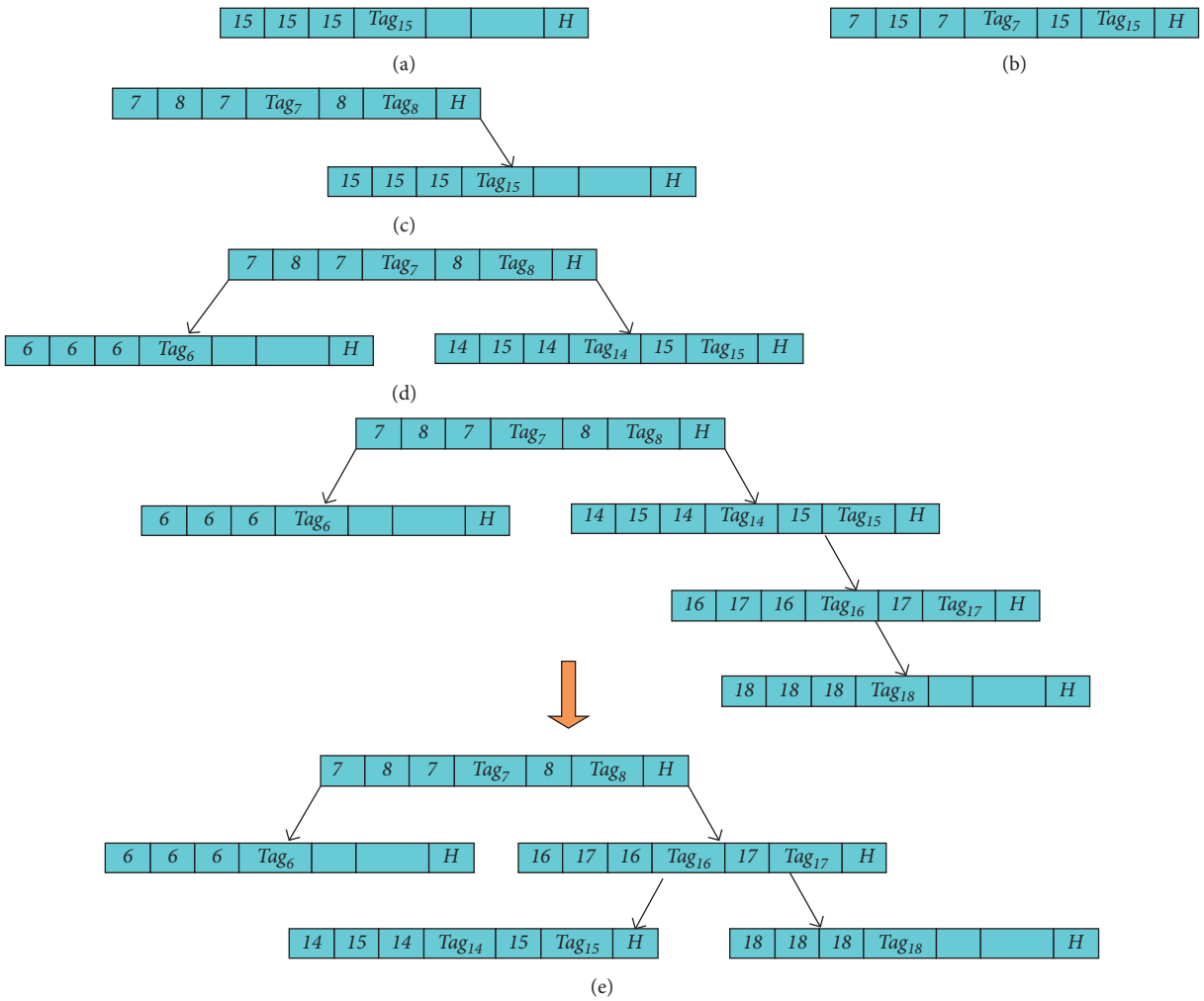

Figure 8: An example of T-Merkle hash tree update stage. (a) Empty node insertion: insert a new tag with index 15. (b) Insertion in not a full node: modify a tag with index 7. (c) Full node insertion: modify a tag with index 8. (d) Left and right insertion: modify tags with indexes 6 and 14. (e) Rotation for balance: insert new tags with indexes 16, 17, and 18.

For data block deletion, CS deletes the ith block directly

After receiving the request BCReq, BC commits the request as follows:

(1) BC inserts the data tag Tag_i^* into T-Merkle hash tree whether the operation is insertion or modification. If an inserted tag Tag_i^* with index i is already in current block, BC just replaces the old Tag_i with the new Tag_i^*.

(2) BC recomputes the node hash values of T-Merkle hash tree.

4.3.3. UpdateVerify Step. Once the dynamic operations have been executed, BC will issue a challenge to check whether CS performs the dynamic operations correctly, which is the same as described in verification stage. The process of data dynamic operations is shown in Figure 9.

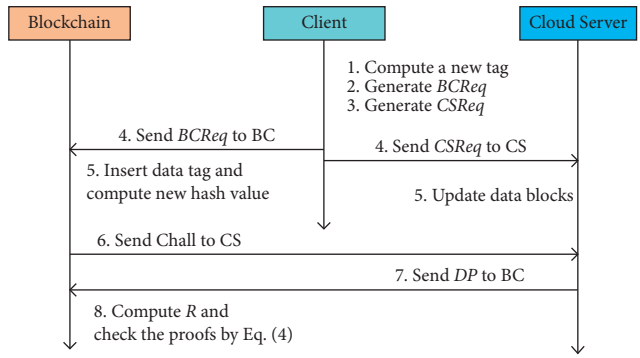

Figure 9: Process of data dynamic operations.

4.4. Support for Batch Verification. In practical applications, there may be a large number of verification tasks that need to be checked in a short time. If these massive tasks can be

processed simultaneously (such as aggregate these tag proofs and data proofs into one separately and verify them at once), the efficiency will be improved and the communication cost will be reduced. To achieve this goal, our key solution is that when generating data proof, CS aggregates different proofs into one instead of sending each proof directly to Client. Thus, the size of proof is constant, regardless of the number of verification tasks. The aggregation of tag proof is the same as that of generating data proof. CS and BC aggregate different proofs into one and can further reduce the verification computation cost of Client because they have more computing power than Client.

Assume the number of verification tasks is L. The data file and tags of each verification task can be expressed as $\{F_l\}_{l\in L}$ and $\{T_l\}_{l\in L}$. The keys are $\{sk_l, pk_l\}_{l\in L}$. Algorithm 2 shows the detailed workflow of batch verification.

According to Algorithm 2, it can be seen that verifying L tasks in batch manner only needs 3 bilinear map pair operations, while checking L tasks one by one requires $3L$ times bilinear map pair operations. Similarly, the commutation cost is $2|p|$, while that of checking L tasks one by one is $2L|p|$. Therefore, batch verification can improve the efficiency in terms of computation and communication cost.

4.5. Discussion on Design Considerations

4.5.1. Decentralized Verification. In the centralized verification scheme, data corruption and tag corruption can lead to verification failure. However, in our decentralized scheme, data tags stored on BC will not be tampered with. If verification fails, the cloud data must be corrupted. In traditional schemes, data and tags are stored on cloud servers. If the verification is performed by cloud servers, the results cannot convince the users, so the third-party audit is introduced. In this work, the data are stored on CS and the tag is stored on BC; CS cannot forge the tag proof, so there is no need of TPA. Although CS and BC encrypt the proofs by using bilinear map, Client can still check the proofs without decrypting them. Since data tags and public key are unknown to CS, it also helps Client remain anonymous to CS. Generally speaking, we take several measures to strengthen the security of the proposed scheme.

4.5.2. T-Merkle Hash Tree. With Merkle hash tree in each block, it can quickly verify whether a data tag exists in the blockchain by using the hash values. However, when Client wants to access a specific data tag, all the tags stored on blockchain need to be traversed. With the growth of blockchain, the efficiency of data tag query on the whole blockchain will become lower and lower. In the structure of T-Merkle hash tree, the index range field in block header helps to quickly determine whether the query key is in block without traversing the whole tree. Furthermore, the T-Merkle hash tree supports binary search in block by adding a minimum index and a maximum index in each node. The time complexity of this process is $O(\log_2(n/k))$. When searching in a node, the time complexity is $O(\log_2(k))$. Thus, the time complexity of Algorithm 1 is $O(\log_2(n/k)) + O(\log_2(k))$.

4.5.3. Detection Probability. Assume that the number of data blocks is n, the sample size of challenge is c, and the number of corrupted blocks is d. X is the number of corrupted blocks detected in the sampled data, and then the detection probability P_X can be expressed as

$$P_X = P\{X \geq 1\} = 1 - P\{X = 0\} = 1 - \left(\frac{n-d}{n}\right)^c. \quad (6)$$

It is not difficult to find that when the corruption rate is 1% and sample size is 300 blocks, the detection probability can reach 95%.

4.5.4. Data Dynamic Operations. Data dynamic operations may cause replace attack, forge attack, and other security problems. On the one hand, CS may fail to update Client's data correctly and cheat the Client by providing fake proof of previously stored version. On the other hand, CS may trick Client by forging the data tag because new versions of the same data blocks reuse the same index and tag private key. If the CS can forge tags, then it can use arbitrary data and forged tags to pass the integrity verification. However, in our design, data tags are stored on BC and CS thus can only forge data proof. Since tags on BC cannot be modified, we insert the new modified tag into BC. When updating a tag in the current block, we only replace it with the new one. When querying tags, we traverse the BC from the latest block to the oldest one, so the first queried tag we find is the latest version. In this way, our scheme can resist replay attack without requiring version number and timestamp information for updates in traditional solutions.

4.5.5. Blockchain Optimization. As we know, data dynamic operations are realized in an append-only way. Once data tags are updated, the old versions are still stored on blockchain, although they will not be accessed again. Hence, storing them on the chain will inescapably cause huge storage overhead. To address this problem, we intend to propose an optimization mechanism. We remove the old block in which data tags are updated, or move the tags not updated to the latest block, so as to guarantee that this block will not be accessed forever, and then delete it. Here, we use Figure 10 to illustrate the key ideal of optimization mechanism. Tag_1 and Tag_2 are updated and stored new ones in Block 3, but Tag_3 is not. We move Tag_3 from Block 1 to Block 3 and then delete Block 1.

4.5.6. Limitations. The proposed work uses blockchain to store data tags and also proposes a new storage structure and optimization method. However, it is still limited by the

Input: challenge$\{\text{Chall}_l\}_{l\in L}$
Output: intact or not
(1) CS calculates the batch data proof as BDP = $e(\sum_{l\in L}\sum_{i\in I}u_{li}\mathscr{H}(m_{li})P, P)$
(2) BC calculates the batch tag proof as BTP = $e(\sum_{l\in L}\sum_{i\in I}u_{li}/\text{Tag}_{li}P^2, P)$
(3) CS and BC send aggregated proofs {BDP, BTP} to Client.
(4) After receiving the proofs, Client checks the correctness of the proofs by calculating BR = $\sum_{l\in L}\sum_{i\in I}u_{li}pk_l$ and checking BTP = BDP $\cdot\, e(BR, P)$
(5) If equation in Algorithm 2 (4) holds, it means all the L tasks are intact.

Algorithm 2: Batch verification.

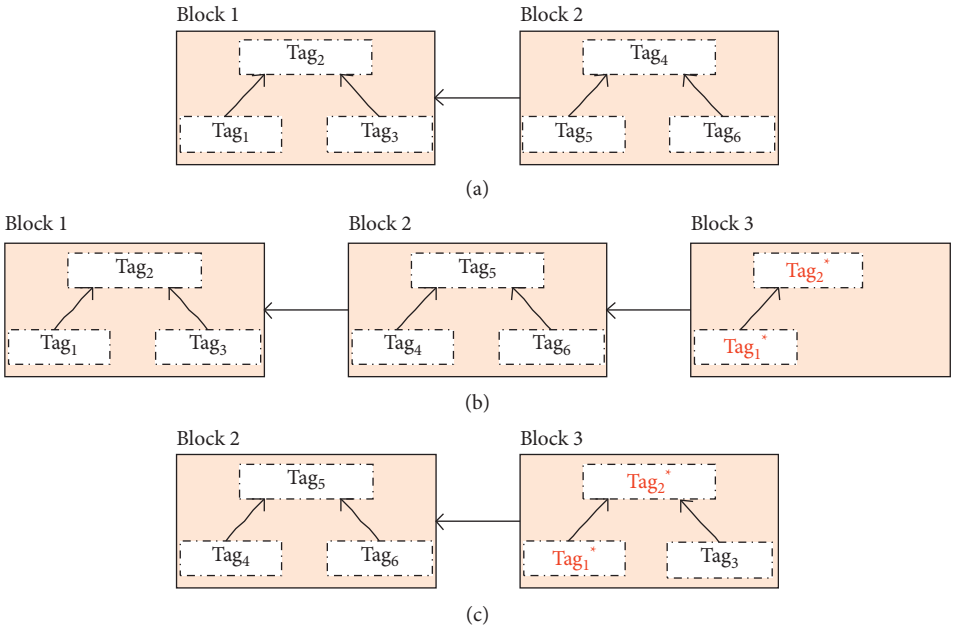

Figure 10: Example of blockchain optimization. (a) Initialization of blockchain. (b) Update Tag_1 and Tag_2. (c) Move Tag_3 to block 3 and delete Block 1.

storage scalability of blockchain. At the same time, the verification method is based on ZSS signature technology. The calculation method makes it difficult to extend our work to support the case of multiuser sharing or deduplication.

5. Analysis

We analyze the correctness and security of our scheme based on four aspects: correctness, forgery attack resistant, replace attack resistant, and fraud attack resistant.

5.1. Correctness Analysis

Theorem 1. *The proposed scheme is correct and feasible to check the integrity of cloud data.*

Proof. The correctness of equation (5) can be derived as follows:

$$\begin{aligned}\text{TP} &= e\left(\sum_{i\in I}\frac{u_i}{\text{Tag}_i}P^2, P\right),\\ &= e\left(\sum_{i\in I}u_i(\mathscr{H}(m_i)+sk)P, P\right),\\ &= e\left(\sum_{i\in I}u_i\mathscr{H}(m_i)P, P\right)\cdot e\left(\sum_{i\in I}u_i skP, P\right),\\ &= \text{DP}\cdot e(R, P).\end{aligned} \tag{7}$$

Similarly, the correctness of batch verification can be calculated as follows:

$$\begin{aligned}\mathrm{BTP} &= e\left(\sum_{l\in L}\sum_{i\in I}\frac{u_{li}}{\mathrm{Tag}_{li}}P^2, P\right),\\ &= e\left(\sum_{l\in L}\sum_{i\in I}u_{li}(\mathscr{H}(m_{li})+sk_l)P, P\right),\\ &= e\left(\sum_{l\in L}\sum_{i\in I}u_{li}\mathscr{H}(m_{li})P, P\right)\cdot e\left(\sum_{l\in L}\sum_{i\in I}u_i sk_l P, P\right),\\ &= \mathrm{BDP}\cdot e\left(\sum_{l\in L}\sum_{i\in I}u_{li}pk_l, P\right),\\ &= \mathrm{DP}\cdot e(BR, P).\end{aligned} \tag{8}$$

From the above calculation process, if the data proof provided by CS passes the verification, the data stored on cloud are uncorrupted. Thus, the proposed scheme is feasible. □

5.2. Security Analysis

5.2.1. Forgery Attack Resistant

Theorem 2. *It is computationally infeasible for CS and BC to falsify data proofs and tag proofs with our scheme.*

Proof. Following the definition of adversary model and security game in [12, 13], the unforgeability of our scheme can be derived as follows.

Suppose the challenge is $\mathrm{Chall} = \{i, u_i\}_{i\in I}$, the correct proof should be DP and TP. We assume that CS and BC forge incorrect proofs DP' and TP', and these incorrect proofs passed the verification. Then, according to equation (5), we have

$$\mathrm{TP}' = \mathrm{DP}'\cdot e(R, P). \tag{9}$$

Since DP and TP are correct proofs, we can get

$$\mathrm{TP} = \mathrm{DP}\cdot e(R, P). \tag{10}$$

By dividing equation (9) by equation (10), we can get

$$\mathrm{TP}'\cdot \mathrm{TP}^{-1} = \mathrm{DP}'\cdot \mathrm{DP}^{-1}. \tag{11}$$

We firstly consider that there are some attackers or malicious CS. The cloud data are tampered with. That is, $\mathrm{DP}' \neq \mathrm{DP}$ and $\mathrm{TP}' = \mathrm{TP}$. With the properties of bilinear maps, we have

$$e\left(\sum_{i\in I}u_i\mathscr{H}(m_i')P, P\right)\cdot e\left(\sum_{i\in I}u_i\mathscr{H}(m_i)P, P\right)^{-1} = 1. \tag{12}$$

By setting $\Delta m = \sum_{i\in I}u_i\mathscr{H}(m_i') - \sum_{i\in I}u_i\mathscr{H}(m_i)$, we have

$$e(\Delta mP, P) = e(P, P)^{\Delta m} = 1. \tag{13}$$

As $\Delta m \neq 0$, we have $e(P, P) = 1$. This conflicts with the hypothesis that P is nondegenerate if $e(P, P) \neq 1$.

We then suppose that BC is under attack and generates a fake tag proof, namely, $TP' \neq TP$. Similar to the above analysis, without the private key sk, it is computationally infeasible to generate a fake m_i' and make equation $1/\mathscr{H}(m_i') + skP = 1/\mathscr{H}(m_i) + skP$ work.

From the above proof, we can know that our verification scheme can correctly check the integrity of cloud data. □ □

5.2.2. Replace Attack Resistant

Theorem 3. *Based on the nonmodifiability of BC, the proposed verification scheme can resist the replace attack from CS.*

Proof. When the challenged data block m_t is broken, CS may try to deceive the verifier by using another data block m_k to replace it. Thus, the data proof DP^* becomes

$$\mathrm{DP}^* = e\left(\sum_{i\in I, i\neq t}u_i\mathscr{H}(m_i)P + u_t\mathscr{H}(m_k)P, P\right). \tag{14}$$

Then, the verification equation (5) can be represented as

$$\mathrm{TP} = \mathrm{DP}^*\cdot e(R, P) = DP\ \cdot e(R, P). \tag{15}$$

Since the tag stored on BC is nonmodifiable and we always query the newest version, the tag proof is still TP.

Thus, we have $\mathscr{H}(m_k) = \mathscr{H}(m_t)$. However, $\mathscr{H}(m_k)$ cannot be equal to $\mathscr{H}(m_t)$ due to the anticollision property of hash function. So, the verification equation does not hold and the proof from CS cannot pass the verification. Therefore, the proposed scheme can resist the replace attack.

5.2.3. Fraud Attack Resistant

Theorem 4. *The proposed scheme prevents the mutual deception between CS and Client.*

Proof. As we know, the data tags stored on BC cannot be tampered with under our scheme. When verification equation (5) fails, it means the cloud data are corrupted. Thus, CS should pay compensation for Client without denying. At the same time, when a dishonest Client uploads incorrect data for the sake of defrauding CS's compensation, actually CS will perform verification stage before storing Client's data, so the dishonest Client can be found out and CS does not need to pay compensation. Thus, the proposed scheme prevents the mutual deception between CS and Client.

6. Performance Evaluation

This section analyzes and compares our scheme with related data integrity verification schemes cited in references in terms of function similarity, storage overhead, and verification efficiency.

6.1. Performance Analysis. As can be seen from Table 2, we compare our scheme with similar schemes in terms of blockchain-based, decentralization, data tag storage place, verification method, fast tag retrieval, data dynamics, and batch verification. In detail, our scheme provides an effective decentralized, dynamic, and batch verification scheme for outsourced data, while ensuring the fast tag retrieval on BC.

Table 2 shows that Yue's scheme [9] and Wang's scheme [27] are more close to our scheme in function. Thus, we compare our scheme with these two schemes from the aspects of storage structure of blockchain and verification efficiency in detail. Yue's scheme [9] uses a multibranch Merkle hash tree to store data blocks on BC, and the integrity of data relies on the root of the hash tree. Wang's scheme [27] is more similar to our scheme, since the verification method is also based on the ZSS short signature technique.

6.1.1. Computational Complexity of Algorithms. Our scheme consists of five basic algorithms: KeyGen, TagGen, ChallGen, ProofGen, and Verify. The KeyGen algorithm and TagGen algorithm are run by Client during initialization stage. The KeyGen algorithm generates the private and public tag keys. The TagGen algorithm calculates a data tag for each data block. The algorithms of ChallGen, ProofGen, and Verify are executed during verification stage. The ChallGen algorithm is run by Client (as verifier) to choose sampling challenge. The ProofGen algorithm is run by CS and BC to calculate the data proof and tag proof. The Verify algorithm is run by Client (as verifier) to validate the proofs. The computational complexity of algorithms is shown in Table 3. From Table 3, it can be seen that the complexity of both KeyGen and Verify algorithms is $O(1)$, the complexity of ChallGen and ProofGen algorithms is $O(c)$, and the complexity of TagGen algorithm is $O(n)$. Based on the results shown, we can conclude that both algorithms used in this work presented a linear growth rate. On the one hand, the algorithms of KeyGen and TagGen only execute one time during initialization stage. On the other hand, the verification efficiency is related to the number of challenged blocks, which is independent of total size of data file. So, we only compare the verification efficiency in part of experimental results.

6.1.2. Storage Structure of Blockchain. The storage structure of blockchain is compared from three aspects: generation complexity, storage space, and query efficiency. The storage structures of these three schemes are multibranch Merkle hash tree, Merkle hash tree, and T-Merkle hash tree, respectively. Generation complexity is measured by the number of hash operations during the construction of the tree. In our scheme, the hash operations come from two aspects: hash computation of each data block and hash computation of each node. Similarly, the storage space is determined by the number of nodes in the tree. The size of each tag in our scheme and Wang and Zhang scheme [27] are the same as the size of data block, which is $|F|/n$. However, we store data block on each node rather than just leaf nodes, so the number of nodes in the tree is fewer than the other two schemes. Since our scheme supports binary search on BC, it needs lower query cost. The detailed comparison among these three schemes is shown in Table 4.

6.1.3. Verification Efficiency. The verification efficiency is estimated in terms of computation cost and communication cost. The computation cost is decided by cryptographic operations. Since the size of challenge request $\text{Chall} = \{i, u_i\}_{i \in I}$ of these three schemes is the same, which is $2c|p|$, the communication cost here refers to the size of data proof and tag proof. From Table 5, it can be seen that the proof size in our scheme is less than other two schemes and the computation cost of both our scheme and Wang's scheme [27] is $O(c)$, while that of Yue's scheme [9] is $O(clog_m^n)$. Furthermore, the computation cost of our scheme on the verifier side is less than that of the other two schemes because we transfer the computation load from the verifier to CS and BC which are more powerful than the verifier.

6.2. Experimental Results. We validate the proposed scheme on a 64-bit Ubuntu system (version 18.04) based PC (8-core Intel i7 processor with 2.11 GHz and 16G of memory). The blockchain platform we choose is Hyperledger Fabric 1.4.0. Implementation of verification algorithms in this work is based on the Pairing-Based Cryptography (PBC) library (pbc-0.5.14) with *a*.param, in which the group order and the base field order are 160 bits and 512 bits, respectively. The three different types of trees are implemented by Go programming language. We use the text file to do the experiment, read the file in binary way, and then slice it into blocks. The size of data blocks is constant, which is equal to 4 KB, and thus the size of files is changed according to the number of data blocks. All simulation results are the average of 50 tests.

6.2.1. Evaluation of Different Structures of Merkle Hash Tree. First, we compare the efficiency of tree generation under different numbers of data blocks. We assume the number of tag in a node of our scheme to be $k = 128$, which achieves better performance as shown in the next experiment. We assume the number of branch in Yue's scheme [9] to be $m = 8$, as eight-branching tree in their scheme achieves the best performance. We test the whole number of blocks from 2048 to 65536 (2048, 4096, 8192, 16384, 32768, and 65536). Figure 11(a) reveals a proportional relationship between the tree generation time and the number of blocks. Obviously, our T-Merkle hash tree shows a better performance than the other two trees because it reduces the number of nodes and the depth of tree under the same condition.

Second, we measure the query efficiency of these three different kinds of trees by setting the number of query blocks from 128 to 2048 (128, 256, 512, 1024, and 2048). The test size is reasonable because of the challenged number of blocks during verification is very small. From Figure 11(b), we can find that our scheme spends relatively low query time because our scheme supports binary search. When the number of query blocks reaches 1024, our scheme takes 1.6 ms, while

Table 2: Comparison of function.

Schemes	Blockchain based	Decentralization	Tag storage place	Verification method	Fast tag retrieval	Data dynamics	Batch verification
Scheme [4]	N	N	CS	BLS	N	Y	Y
Scheme [9]	Y	Y	Blockchain	MHT	N	N	N
Scheme [10]	Y	N	Blockchain	RSA	N	N	N
Scheme [27]	Y	Y	CS	ZSS	N	Y	N
Scheme [28]	Y	Y	CS	BLS	N	N	Y
Scheme [29]	Y	Y	CS	BLS	N	Y	Y
Scheme [30]	Y	Y	CS	BLS	N	N	N
Our scheme	Y	Y	Blockchain	ZSS	Y	Y	Y

Table 3: Computational complexity of algorithms.

Algorithms	Computational cost	Computational complexity
KeyGen	$\mathscr{M}$	$O(1)$
TagGen	$n(\mathscr{H}+\mathscr{A}+\mathscr{M})$	$O(n)$
ChallGen	$c\mathscr{A}$	$O(c)$
ProofGen	$2c\mathscr{A}+2\mathscr{M}+2\mathscr{H}+2\mathscr{P}$	$O(c)$
Verify	$\mathscr{P}$	$O(1)$

n is the total number of blocks in the data file; m is the branch number of tree; c is the number of challenged blocks; $\mathscr{A}$ denotes one addition in G; $\mathscr{M}$ denotes one multiplication in G; $\mathscr{P}$ denotes one pairing operation; and $\mathscr{H}$ denotes one hashing operation.

Table 4: Comparison of storage structure.

Scheme	Generation complexity	Storage space	Query efficiency
Yue et al. scheme [9]	$\text{sum}(Y)$	$(\text{sum}(Y)+n)\lvert p\rvert$	$O(\log_m^n)$
Wang and Zhang scheme [27]	$\text{sum}(W)$	$(\text{sum}(W)+n)\lvert p\rvert$	$O(n\log_2^n)$
Our scheme	$n+n/k$	$(n+n/k)\lvert p\rvert+(n+3n/k)\lvert\text{id}\rvert$	$O(\log_2^{n/k}+\log_2^k)$

n is the total number of blocks in the data file; m is the branch number of tree; k is the number of tags in a T-Merkle hash tree node; $|\text{id}|$ is the size of index; and $|p|$ is group size. Let $\text{sum}(Y)=m^0+m^1+\cdots+m^{\log_m^n}$ and $\text{sum}(W)=2^0+2^1+\cdots+2^{\log_2^n}$.

Table 5: Comparison of verification efficiency.

Scheme	Communication cost		Computation cost		
	CS	BC	CS	BC	Verifier
Yue et al. scheme [9]	$c\lvert p\rvert$	$c\log_m^n\lvert p\rvert$	$c\mathscr{H}$	$c\log_m^n\mathscr{H}$	—
Wang and Zhang scheme [27]	$3\lvert p\rvert$	$c\lvert p\rvert$	$c\mathscr{A}+c\mathscr{H}$	$(c+1)\mathscr{A}+2\mathscr{M}$	$3\mathscr{P}$
Our scheme	$\lvert p\rvert$	$\lvert p\rvert$	$c\mathscr{A}+c\mathscr{H}+\mathscr{P}$	$c\mathscr{A}+2\mathscr{M}+\mathscr{P}$	$\mathscr{P}$

n is the total number of blocks in the data file; m is the branch number of tree; c is the number of challenged blocks; $\mathscr{A}$ denotes one addition in G; $\mathscr{M}$ denotes one multiplication in G; $\mathscr{P}$ denotes one pairing operation; and $\mathscr{H}$ denotes one hashing operation.

Yue's scheme [9] requires 15.1 ms and Wang and Zhang scheme [27] requires 54.7 ms.

6.2.2. Evaluation of Verification Efficiency. First, we count the total computation time in verification stage versus different number of challenged blocks. Yue's scheme I and Yue's scheme II mean that the file size is 65536 and 131072, respectively. Figure 12(a) shows that the computation time increases as the challenged size increases. When the number of challenged blocks is 500, no matter what the file size is, the computation time of both our scheme and Wang's scheme is around 940 ms, but that of Yue's scheme depends on the total number of blocks.

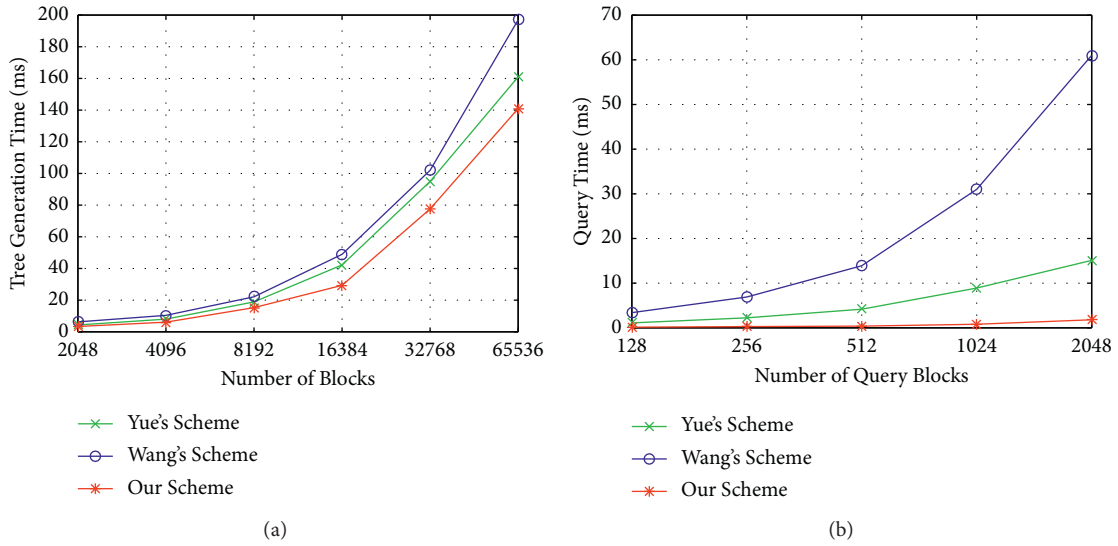

Figure 11: Comparison of different Merkle hash trees. (a) Tree generation time. (b) Query time.

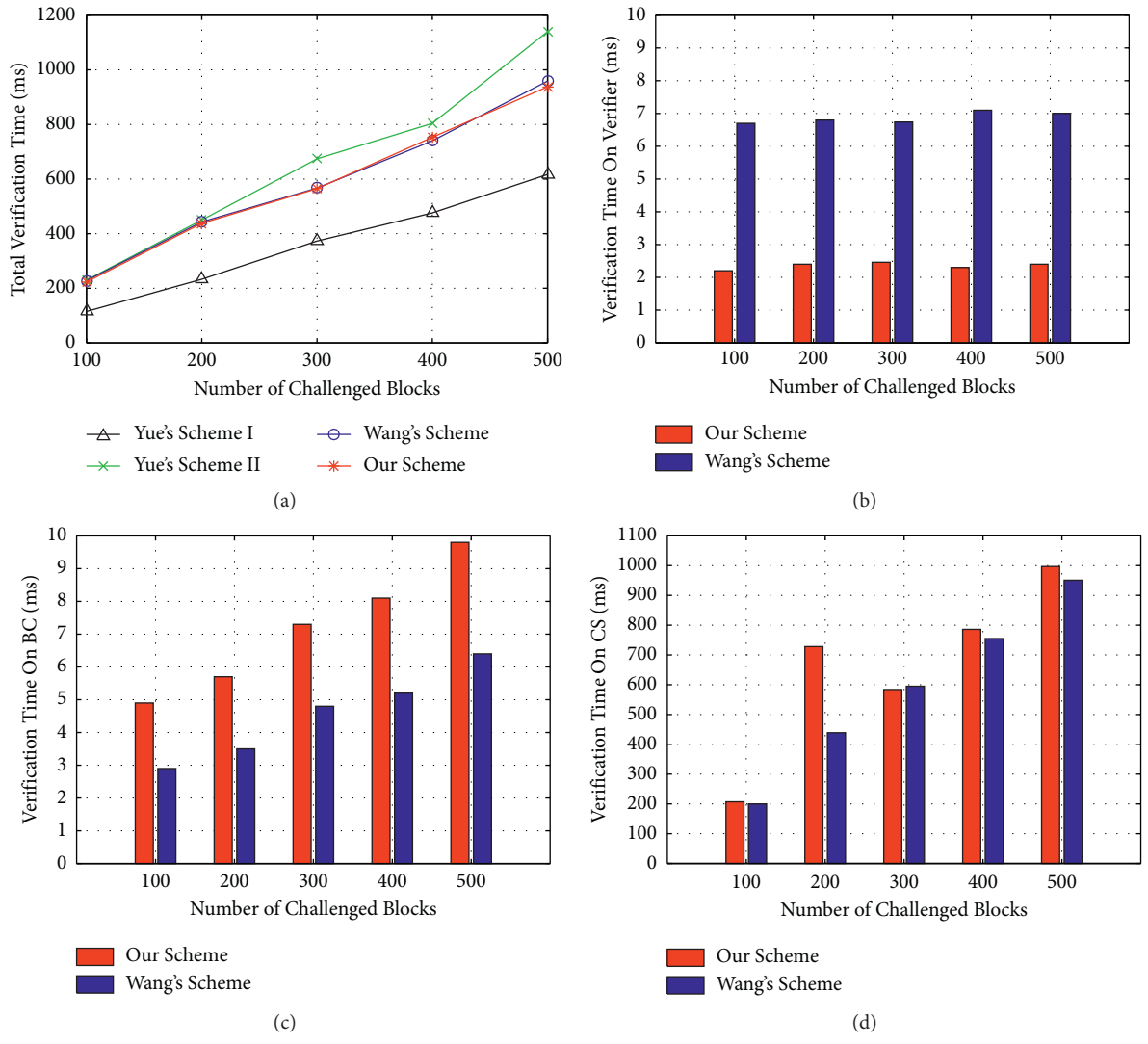

Figure 12: Comparison of verification time under different challenged blocks. (a) Total verification time. (b) Verification time on Verifier. (c) Verification time on BC. (d) Verification time on CS.

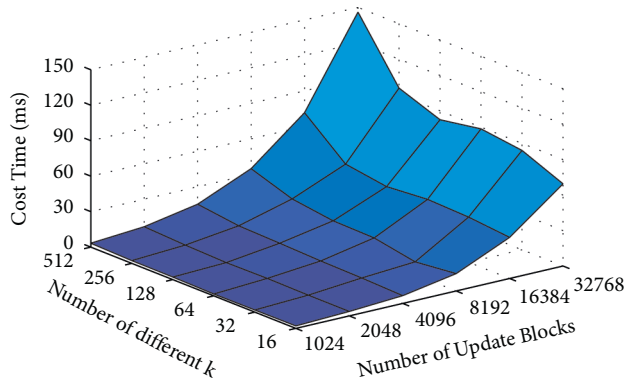

Figure 13: Comparison of dynamic update operation efficiency.

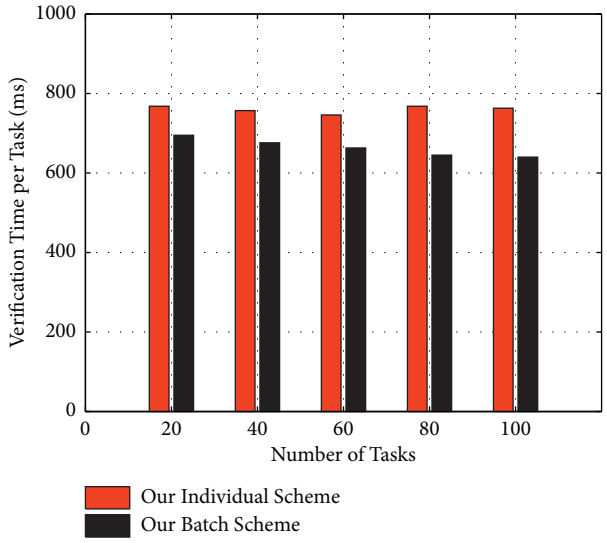

Figure 14: Comparison of batch verification time under different tasks.

Second, we further distinguish the computation time between our scheme and Wang's scheme from three sides, which include CS, BC, and verifier. From Figures 12(b)–12(d), it can be easily found that our scheme takes less computation time on the verifier than Wang's scheme, while spending more time on the side of BC and CS. By moving computation from verifier to BC and CS, the efficiency can be improved when data integrity is checked by a verifier with computation constrained device.

6.2.3. Evaluation of Data Dynamic Operations. In order to demonstrate the dynamic operation efficiency of our scheme, we compare our scheme under different numbers of tags in a node and different numbers of update blocks. The computation cost during data update comes from computing new tags and updating blockchain. In addition, both tag modification operation and tag insertion operation are the same as tag appending operation in our scheme and the deletion operation does not involve any computational overhead on BC. As shown in Figure 13, the dynamic operations do not cause heavy cost, which is efficient as tree generation compared with Figure 11(a). When the number of tags in a node is $k = 128$, the dynamic update achieves a better performance under the same number of update blocks.

6.2.4. Evaluation of Batch Verification Efficiency. In the case of $c = 500$, we compare our batch verification with individual verification through different numbers of tasks in terms of computation. Figure 14 shows that the average computation time of each verification task in batch verification is less than that of single verification since the number of pairing operations in batch verification is 3, which is independent of the number of tasks.

7. Conclusion

We proposed a verification scheme based on blockchain for cloud storage, which aims to address the problem of untrusted TPA. Our scheme realized efficient and lightweight data integrity verification by transferring computation from verifier to CS and BC with the help of ZSS short signature and the bilinear pairing. We further designed a new storage structure called T-Merkle hash tree to achieve binary search on blockchain. We also extended our scheme to support block level based dynamic operation and presented a detailed tag update process on blockchain. Furthermore, we proposed a batch verification scheme to check massive verification tasks so as to save the computation and communication costs. We conducted experiments by constructing a blockchain network on Hyperledger Fabric. The security and efficiency of our scheme are proved by the analysis and experimental results. In future, we will extend our work to deal with blockchain-based integrity verification of shared data or deduplicated data. To support the case of multiusers, we need to improve the signature technology, and it will be very interesting and challenging.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Natural Science Foundation of Hubei Province (No. 2018CBF109), Research Project of Hubei Provincial Department of Education (No. Q20191710), National Science Foundation of China (No. 62062045), Project of Jiangxi Provincial Social Science Planning (No. 17XW08), and Science and Technology Research Project of Jiangxi Provincial of Education Department (No. GJJ180905).

References

[1] M. Li and P. P. C. Lee, "STAIR codes," *ACM Transactions on Storage*, vol. 10, no. 4, pp. 1–30, 2014.

[2] M. Antonio, M. Antonio, and G. Javier, "Dynamic security properties monitoring architecture for cloud computing," *Security Engineering for Cloud Computing:Approaches and Tools*, pp. 1–18, IGI Gobal, PA, USA, 2012.

[3] T. Jamal, M. Antonio, and S. Nesmachnow, "Evolution oriented monitoring oriented to security properties for cloud applications," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, Hamburg, Germany, August 2018.

[4] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013.

[5] K. Yang and X. Jia, *TSAS: Third-Party Storage Auditing Service," Security for Cloud Storage Systems*, pp. 7–37, Springer, Berlin, Germany, 2014.

[6] Y. Zhang, C. Xu, and X. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Transactions on Cloud Computing (Early Access)*, vol. 7, 2019.

[7] A. Maña and A. Mana, "TPM-based protection for mobile agents," *Security and Communication Networks*, vol. 4, no. 1, pp. 45–60, 2011.

[8] S. N. Bitcoin, *A Peer-To-Peer Electronic Cash System*, 2008, https://bitcoin.org/bitcoin.pdf.

[9] D. Yue, R. Li, Y. Zhang, W. Tian, and C. Peng, "Blockchain based data integrity verification in P2P cloud storage," in *Proceedings of the IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 561–568, Singapore, December 2018.

[10] H. Wang and D. He, "Blockchain-based private provable data possession," *IEEE Transactions on Dependable and Secure Computing (online)*, vol. 18, 2020.

[11] F. Zhang, R. Safavi-Naini, and W. Susilo, *An Efficient Signature Scheme from Bilinear Pairings and its Applications*, Springer, Berlin, Germany, pp. 277–290, 2004.

[12] A. Juels and B. S. Kaliski, "PORs: proofs pf retrievability for large files," in *Proceedings of ACM CCS*, pp. 584–597, Alexandria, VA, USA, October 2007.

[13] G. Ateniese, R. Burns, R. Curtmola et al., "Provable data possession at untrusted stores," in *Proceedings of ACM CCS 2007*, pp. 598–610, Alexandria, VA, USA, October 2007.

[14] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847–859, 2011.

[15] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 19, pp. 1717–1726, 2013.

[16] B. Wang, B. Li, and H. Li, "Panda: public auditing for shared data with efficient user revocation in the cloud," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 92–106, 2015.

[17] J. Yuan and S. Yu, "Efficient public integrity checking for cloud data sharing with multi-user modification," in *Proceedings of IEEE INFOCOM*, pp. 2121–2129, Hong Kong, China, March 2014.

[18] C. H. Chen and P. C. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage: theory and implementation," *IEEE Transactions on Reliability*, vol. 64, no. 3, pp. 840–851, 2015.

[19] C. Liu, J. Chen, L. T. Yang et al., "Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2234–2244, 2014.

[20] S. Ghimire, J. Y. Choi, and B. Lee, "Using blockchain for improved video integrity verification," *IEEE Transactions on Multimedia*, vol. 22, no. 1, pp. 108–121, 2020.

[21] K. Nandakumar, N. Ratha, and S. Pankanti, "Proving multimedia integrity using sanitizable signatures recorded on blockchain," in *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, pp. 151–160, Denver, CO, USA, June 2019.

[22] J. Xue, C. Xu, J. Zhao, and J. Ma, "Monitoring file integrity using blockchain and smart contracts," *Science China Information Sciences*, vol. 62, no. 3, pp. 32–104, 2019.

[23] Y. Zhang, C. Xu, X. Lin, and X. S. Shen, "Blockchain-based public integrity verifiation for cloud storage against procrastinating auditors," *IEEE Transactions on Cloud Computing (Early Access)*, vol. 29, 2019.

[24] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain based data integrity service framework for IoT data," in *Proceedings of the ICWS*, pp. 468–475, Honolulu, Hawaii, June2017.

[25] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, "ProvChain: a blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 468–477, Madrid, Spain, May 2017.

[26] C. Wang, S. Chen, Z. Feng, Y. Jiang, and X. Xue, "Block chain-based data audit and access control mechanism in service collaboration," in *Proceedings of the ICWS*, pp. 214–218, Orlando, FL, USA, July 2019.

[27] H. Wang and J. Zhang, "Blockchain based data integrity verification for large-scale IoT data," *IEEE Access*, vol. 7, pp. 164996–165006, 2019.

[28] H. Yu, Z. Yang, and R. O. Sinnott, "Decentralized big data auditing for smart city environments leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 6288–6296, 2019.

[29] P. Huang, K. Fan, H. Yang, K. Zhang, H. Li, and Y. Yang, "A collaborative auditing blockchain for trustworthy data integrity in cloud storage system," *IEEE Access*, vol. 8, pp. 94780–94794, 2020.

[30] Y. Miao, Q. Huang, M. Xiao, and H. Li, "Decentralized and privacy-preserving public auditing for cloud storage based on blockchain," *IEEE Access*, vol. 8, pp. 139813–139826, 2020.

[31] X. Yang, X. Pei, M. Wang, T. Li, and C. Wang, "Multi-replica and multi-cloud data public audit scheme based on blockchain," *IEEE Access*, vol. 8, pp. 144809–144822, 2020.

[32] K. He, J. Shi, C. Huang, and X. Hu, "Blockchain based data integrity verification for cloud storage with T-merkle tree," *Algorithms and Architectures for Parallel Processing*, Springer, Berlin, Germany, pp. 65–80, 2020.

[33] H. Lu, Y. Y. Ng, and Z. Tian, ""T-tree or B-tree: main memory database index structure revisited," in *Proceedings 11th Australasian Database Conference. ADC 2000 (Cat. No.PR00528)*, pp. 65–73, Canberra, Australia, January 2000.

[34] K. He, C. Huang, J. Shi, and J. Wang, "Public integrity auditing for dynamic regenerating code based cloud storage," in *Proceedings of the 2016 IEEE Symposium on Computers and Communication (ISCC)*, pp. 581–588, Messina, Italy, June 2016.