Received 5 December 2022, accepted 19 January 2023, date of publication 25 January 2023, date of current version 19 June 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3240066*

RESEARCH ARTICLE

# Data Integrity Audit Scheme Based on Quad Merkle Tree and Blockchain

**ZHENPENG LIU**[1,2], **LELE REN**[1], **YONGJIANG FENG**[1], **SHUO WANG**[1], **AND JIANHANG WEI**[2,3]
[1]School of Cyberspace Security and Computer, Hebei University, Baoding, Hebei 071002, China
[2]Information Technology Center, Hebei University, Baoding 071002, China
[3]Network and Experiment Management Center, Xinjiang University of Science and Technology, Korla 841000, China

Corresponding author: Jianhang Wei (wei@hbu.edu.cn)

This research was supported by the Natural Science Foundation of Hebei Province, China under Grant No. F2019201427 and Fund for Integration of Cloud Computing and Big Data, Innovation of Science and Education of China under Grant No. 2017A20004.

**ABSTRACT** Cloud storage is an essential method for data storage. Verifying the integrity of data in the cloud is critical for the client. Traditional cloud storage approaches rely on third-party auditors (TPAs) to accomplish auditing tasks. However, third-party auditors are often not trusted. To eliminate over-reliance on third-party auditors, this paper designs a blockchain-based auditing scheme that uses blockchain instead of third-party auditors to ensure the reliability of data auditing. Meanwhile, our scheme is based on the audit method of the quad Merkle hash tree, using the root of the quad Merkle hash tree to verify the integrity of data, which significantly improves computing and storage efficiency. Automated verification of auditing activities by deploying smart contracts on the blockchain allows us to have a more up-to-date picture of data integrity. The performance of the scheme is evaluated through security analysis and experiments, which prove that the proposed scheme is secure and effective.

**INDEX TERMS** Integrity auditing, blockchain, Merkle tree, smart contract.

## I. INTRODUCTION

An increasing number of people are storing their data in the cloud. Clients can enjoy many benefits by outsourcing data, such as alleviating heavy storage management, unlimited access anytime, anywhere, reduced hardware/software expenses, and staff maintenance. However, storing data in the cloud can cause a client to lose control over the data management, leading to security risks. Data loss or corruption in cloud servers is usually due to malicious attacks, hardware failures, internal attacks, or human errors [1], [2], [3]. These reasons lead to the need for cloud users to frequently use efficient ways to perform data integrity audits on outsourced data. Therefore, data integrity issues are a concern for many researchers.

To determine whether data is being handled and stored securely. Users usually adopt some strategies to ensure the integrity and availability of the outsourced data. Initially, researchers proposed provable data possession (PDP) [4], [5], [6] for checking whether the remote cloud servers are storing the data files correctly. In further research on integrity auditing, researchers have proposed proof of retrievability (POR) [7], [8], [9], which checks whether a remote server has the user's data. Later, to reduce the audit burden of client verification integrity and improve the fairness of data checking, some researchers have resorted to third-party auditors (TPAs) to implement data integrity checking [10], [11], [12]. Scheme [13] proposes an efficient and dynamic public auditing scheme audited by a third-party auditor. Scheme [14] proposes a multi-copy data integrity verification scheme based on temporal chaos, and Scheme [15] proposes a data label replacement algorithm for data integrity verification in cloud storage. Next, to remove the public critical management burden, some public auditing schemes for identity-based cryptosystems are proposed [16], [17], [18]. Scheme [19] proposes an identity-based remote data integrity verification scheme for cloud storage with privacy-preserving properties; scheme [20] proposes an effective identity-based

The associate editor coordinating the review of this manuscript and approving it for publication was Mehdi Sookhak.

public integrity audit of shared data stored in the cloud while protecting user privacy; scheme [21] proposes an identity-based efficient signature scheme; scheme [22] proposes a multi-copy data integrity verification scheme based on identity signatures. The above studies are all based on data integrity verification schemes by a third-party auditor. Nowadays, researchers believe that TPA cannot be trusted absolutely and may become a bottleneck of the system [23], [24]. Blockchain technology has been investigated to improve the situation. Blockchain is a decentralized new distributed computing paradigm. Applying blockchain technology to cloud computing [25], [26], [27], using the security mechanism of blockchain to improve the secure storage and computing performance of the cloud is a good research topic. Hao et al. [28] proposed a blockchain-based outsourced data integrity verification scheme in a non-trust environment. Ren et al. [29] proposed an identity-based proxy aggregation signature (IBPAS) scheme to improve the efficiency of signature verification, thus compressing the storage space and reducing the communication bandwidth. Ren et al. [30] introduced an innovative approach DCOMB method to build a blockchain-based query model for IoT data, which improves data interoperability and generality of IoT database systems by mining hash computation to achieve queries. Zhao et al. [35] propose a blockchain-based remote data integrity verification scheme for the privacy protection of IoT information systems. Zhu et al. [31] developed a blockchain-based document management system that specifically addresses the problem of easy tampering with project documents. In the field of cloud data integrity verification. Wang et al. [32] deeply combined blockchain with the PDP scheme to create the first efficient and secure blockchain-based PDP model. Wei et al. [33] proposed an integration model using blockchain technology. They used mobile agent technology to deploy a distributed VM agent model in the cloud to ensure reliable data storage, monitoring, and verification. The above studies are all blockchain-based data integrity verification schemes.

As we know, Merkle Hash Tree is one of the critical technologies of blockchain. It does not require downloading all transaction data to verify data integrity. Although the Merkle tree structure has outstanding advantages, its linear structure and a large number of hash operations make the processing speed not very satisfactory, and the value of each node in the binary tree structure also needs to be stored, which generates a large amount of storage overhead. Therefore, this paper proposes a data integrity auditing scheme based on a quad Merkle hash tree and blockchain, aiming to achieve efficient and secure data storage.

The contributions of this paper can be summarized as follows.

1) In this paper, we design a blockchain-based auditing scheme that uses blockchain instead of a third-party auditor to ensure the reliability of data auditing. It also makes the data stored in the cloud more secure and private and prevents data from being tampered with by people with ulterior motives.
2) Our scheme is based on the quad Merkle hash tree scheme. The quad Merkle hash tree is more efficient than the general binary Merkle hash tree, using the root of the quad Merkle hash tree to verify the integrity of the data, which greatly improves computing and storage efficiency.
3) In this paper, several smart contracts are deployed, and automatic verification of auditing activities is achieved using the deployment of smart contracts on the blockchain, allowing us to grasp the integrity of the data more easily.
4) Safety analysis and performance evaluation of the proposed scheme proved its feasibility of the scheme.

The rest of the paper is organized as follows. Section II introduces the relevant knowledge required for scheme construction. Section III presents the scheme we have designed, including the system model and threat model, and design goals. Section IV presents the specific implementation details of our scheme. In Section V and Section VI, the security analysis and performance evaluation are described, respectively. In the last section, the paper is summarized.

## II. PRELIMINARIES

### A. BLOCKCHAIN TECHNOLOGY

A blockchain is a chain of one block after another. Each block holds a certain amount of information, and they are connected in a chain in the order of the time they were created. This chain is kept in all the servers, and the whole blockchain is secure if there is one server working in the whole system. These servers are called nodes in the blockchain system, and they provide storage space and arithmetic support for the whole blockchain system. To modify the information in the blockchain, one must obtain the consent of more than half of the nodes and modify the information in all the nodes, which are usually in the hands of different subjects, making it extremely difficult to tamper with the information in the blockchain. Compared with traditional networks, blockchain has two core features: data is difficult to be tampered with and decentralized. Based on these two features, the information recorded by blockchain is more authentic and reliable, which can help solve the problem of people's mutual distrust.

### B. MERKLE TREE

Merkle hash trees are a class of hash-based binary or multinomial trees where the value on the leaf node is usually the hash of the data block, while the value on the non-leaf node is the hash of the combined result of all the children of that node.

Figure 1 below shows a Merkle hash tree, where the value of node A must be obtained by computing the values on nodes C and D. The leaf nodes C and D store the hashes of the data blocks $L_1$ and $L_2$, respectively, while the non-leaf node A stores the hash of the combination of its children C and D.

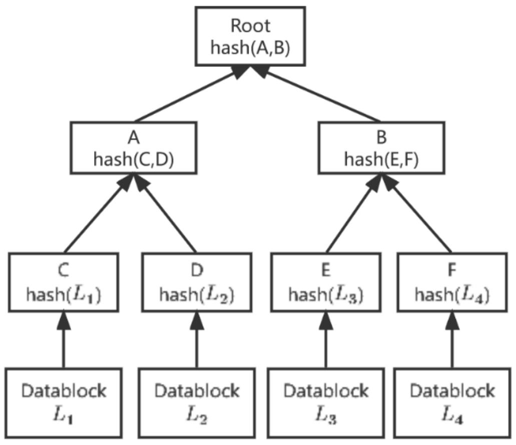


**FIGURE 1. The structure of Merkle hash tree.**

The hash of such non-leaf nodes is called the path hash, while the hash of the leaf nodes is the hash of the actual data.

When data is transmitted from A to B, to check the integrity of the data, it is only necessary to verify whether the root nodes of the Merkle trees constructed on A and B are the same. If it is consistent, the data has not been changed during the transmission. If it is not, it means that the data was modified during transmission. Furthermore, locating the tampered node through the Merkle tree is straightforward.

### C. MERKLE HASH TREE ON THE BLOCKCHAIN

Merkle trees on the blockchain are binary trees that are used to store transaction information. Figure 2 shows the structure of a Merkle hash tree on the blockchain. Each transaction in the figure is paired two by two to form the leaf nodes of the Merkle tree, which in turn generates the entire Merkle tree. The Merkle tree allows a client to verify whether the transaction is included in a block by using the Merkle tree root obtained from the block header and a list of intermediate hashes provided by other clients. The client providing the intermediate hashes does not need to be trustworthy since forging the block header is expensive, and forging the intermediate hashes would cause the verification to fail.

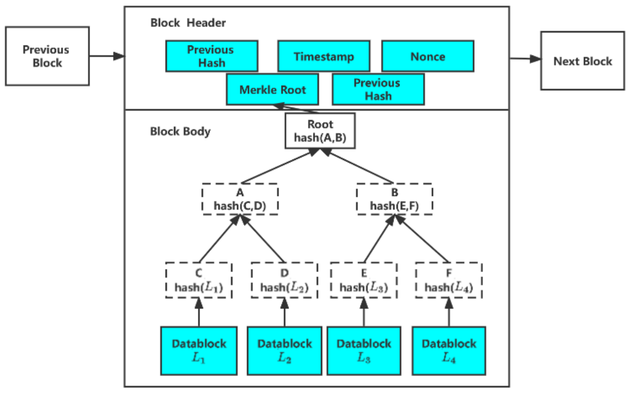


**FIGURE 2. The structure of the Merkle hash tree on the blockchain.**

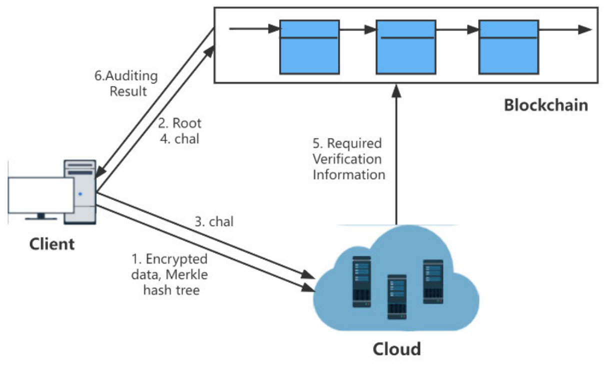
## III. SYSTEM FRAMEWORK

### A. SYSTEM MODEL

As shown in Figure 3, the system model of a blockchain-based cloud storage auditing scheme involves three entities, which are client, cloud, and blockchain (BC).

1) Client: The client refers to the data owner, which can be an individual or an organization. The client has a large amount of data and needs to use the cloud server to store the data and reduce its own storage and computing burden.
2) Cloud: The cloud server is managed and maintained by the cloud service provider with massive storage space and computing resources, which lays the foundation for storing large amounts of data from the client and verifying the integrity of the stored data.
3) Blockchain (BC): The blockchain stores the root of the Merkle hash tree constructed from the client's data and is used to verify the integrity of the data.

After encrypting the data, the client generates a quad Merkle hash tree using the data block signatures, sends the root *Root* to the blockchain for storage, and sends the encrypted data along with the Merkle hash tree to the cloud for storage but loses control of the data. Therefore, a query message is sent to the cloud and the blockchain, and the cloud returns the verification information related to the query to the blockchain. The blockchain uses the information sent by the cloud to calculate the new Merkle hash tree root *Root´*, compare it with the original root *Root*, verify its integrity, and send the result to the client.



6.Auditing Result
2. Root
4. chal
Blockchain
5. Required Verification Information
3. chal
Client
1. Encrypted data, Merkle hash tree
Cloud

**FIGURE 3. System model.**

### B. THREAT MODEL

To analyze the security of data stored in the cloud, we consider two types of threats, i.e., semi-trusted or untrusted cloud servers and malicious clients. Cloud service providers are curious about the stored client data and may sell the data to other organizations for profit; they may intentionally choose to overwrite some data errors to maintain their reputation;

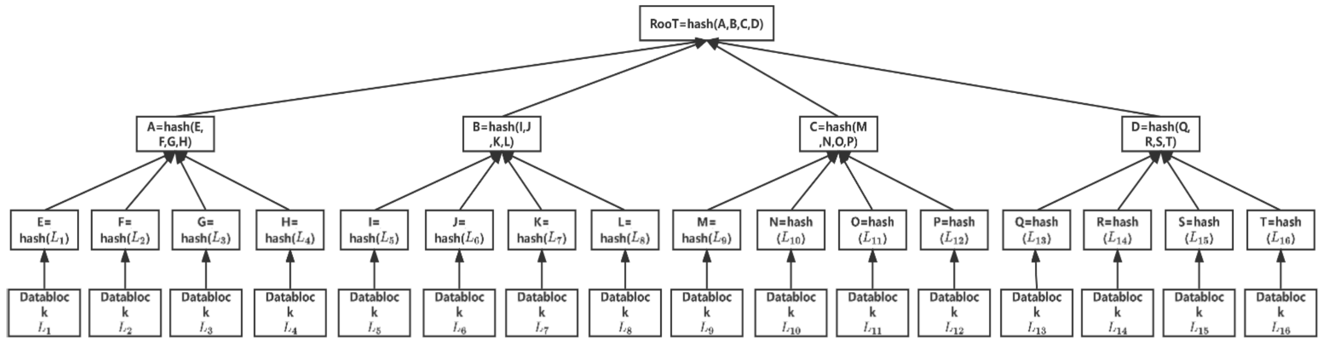


**FIGURE 4.** **Quad Merkle hash tree.**

and to save storage space, cloud service providers may delete some data that is not used for a long time. Also, the client may intentionally store wrong data in the cloud to claim compensation from the cloud service provider.

Therefore, cloud servers and clients may launch the following attacks:

1) Forgery attack: Cloud forges data information with the intention of deceiving the blockchain and the client to pass verification.
2) Replacement attack: The cloud server replaces the challenged data block with an older version of the previously validated data block or an uncorrupted data block to pass validation.
3) Fraud attack: The client uploads corrupt data blocks to the cloud or uploads incorrect Merkle tree root to the blockchain and then claims that the data is intact to trick the cloud into getting compensation.

### C. DESIGN GOAL

To ensure the privacy and security of data stored in the cloud, we aim to achieve the following goals:

1) Audit reasonableness: If the complete data stored by the client is not stored in the cloud, then it does not pass the verification of the blockchain. This means that any misconduct (such as modification or forgery) can be identified. DO can detect misbehavior when an untrusted cloud intentionally returns incorrect or false search results.
2) Data privacy: specific information in the data file is invisible to the cloud.
3) Low cost of blockchain storage: The storage cost of data stored on BC should be as low as possible.

## IV. SYSTEM DESIGN AND IMPLEMENTATION

### A. QUAD MERKLE TREE

The traditional Merkle tree is a binary tree structure that stores many hash values. Although the traditional Merkle tree structure has outstanding advantages, its linear structure and a large number of hash operations make the processing speed not very satisfactory. Moreover, a binary tree can only reduce the amount of data by at most half at a time, and the value of each node in each level of the structure also needs to be stored, resulting in a large amount of data storage. Therefore, we thought of using multinomial trees to improve the computational efficiency of data and save storage space. However, as the number of forks in a hash tree increases, its search complexity also increases. The search complexity of an $m$-fork tree is $m \log mN$ ($m$ is the number of branches in the hash tree). Therefore, we choose a moderate quad Merkle tree instead of the traditional Merkle tree structure to implement our scheme.

The quad Merkle tree can reduce the number of intermediate nodes and layers stored in the Merkle tree. Also, it can reduce the number of hash calculations when generating the root node. To explain the structure in detail, we provide the proposed structure with 16 input data in Figure 4 as an example. As shown in Figure 4, we choose SHA256 as the hash function to encrypt the data blocks in the quad Merkle tree.

Suppose the data block $L_i$ ($i = 1, 2, \ldots, 16$) is the 16 data blocks we want to encrypt. The steps to construct a quad Merkle hash tree for the data blocks are as follows.

1) Do a hash operation on the data block $L_i$($i = 1, 2, \ldots, 16$) respectively, e.g., E = hash ($L_1$).
2) Four adjacent hash blocks are concatenated in series and then do the hash operation, e.g., A = hash(E, F, G, H) = (hash($L_1$) + hash($L_2$) + hash($L_3$) + hash($L_4$)).
3) Four adjacent hash blocks are concatenated in series and then do the hash operation, e.g., Root = hash (A, B, C, D) = (hash (E, F, G, H) + hash (I, J, K, L) + hash (M, N, O, P) + hash (Q, R, S, T)).
4) Generate the root *Root* of the Merkle hash tree.

A quad Merkle tree handles 16 input data and only contains 4 layers, 21 hash operations, and 37 nodes to achieve storage. However, the traditional binomial Merkle tree structure requires 6 layers, 31 hash operations, and 47 nodes. Therefore, the storage space and computation of a quad Merkle tree structure are lower than that of a traditional Merkle tree structure, i.e., the latency of generating a quad Merkle tree is always less than that of generating a binary Merkle tree. Since

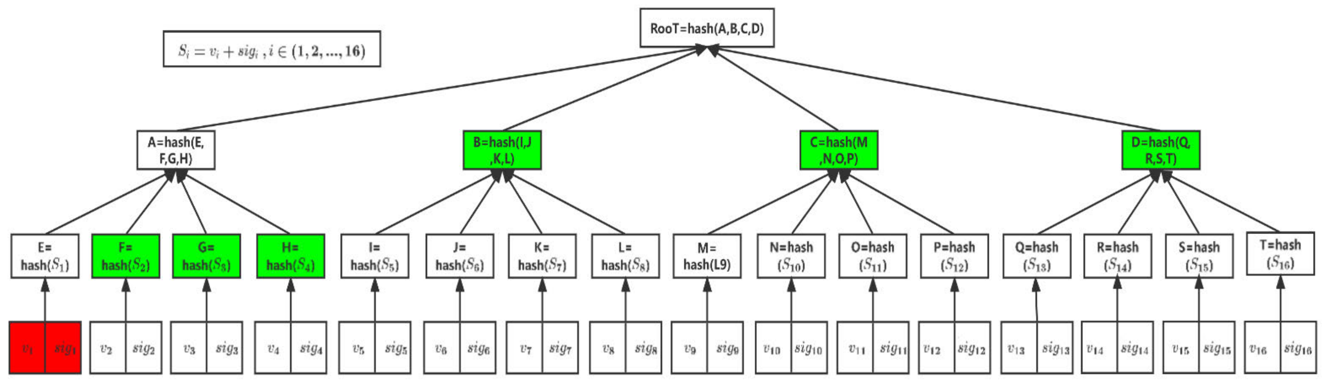


**FIGURE 5.** Auxiliary information.

the computational cost is positively correlated with the computational latency, it follows that the quadtree outperforms the binary tree in terms of computational overhead.

For auxiliary paths, each additional element of the auxiliary path is a hash string that adds very little storage space. The time delay for generating auxiliary paths does not increase significantly with the number of branches. Therefore, the additional communication and computational costs of auxiliary paths caused by the multi-branch tree structure are negligible.

However, as the number of forks of a hash tree increases, its search complexity also increases. The search complexity of an m-fork tree is $\mathrm{mlog_m N}$ (m is the number of branches of the hash tree). Therefore, we choose a quad Merkle tree instead of the traditional Merkle tree structure to implement our scheme, whose search complexity increases somewhat, but our computational overhead is greatly improved, and the search space also decreases by a constant multiple due to the reduction in the number of nodes and layers. Therefore, overall, the quadtree performs better than the binary tree.

## *B. AUXILIARY INFORMATION*

In our scheme, the cloud wants to send information about the verification of the blockchain, including auxiliary information. Here, we explain the auxiliary information for the quad Merkle tree used in this scheme. We assume that the data file sent by the client is divided into 16 data blocks. Then we construct a quad Merkle hash tree, as shown in Figure 5. In this Merkle hash tree, each data block tag $Tag_i$ is assigned a random query number $v_i$, where $S_i = v_i + Tag_i$, where $i \in \{1, 2, \ldots, 16\}$. Suppose we need to verify the integrity of data block 1, find the corresponding questioned random number and the corresponding data block signature, i.e., verify the integrity of node $Tag_1$ (the red node in the figure). To verify its integrity, we need to see if the Merkle tree root *Root* has changed. We also need B, C, D, F, G, and F to get the value of *Root*. Therefore, B, C, D, F, G, and F in Figure 5 (green node in the figure) is the auxiliary information to verify the data block tag $Tag_1$.

## *C. SCHEME IMPLEMENTATION*

Our scheme consists of two phases: the initialization phase and the verification phase.

### 1) INITIALIZATION PHASE

First, the client generates a random value $sk \in Z_p$ as its private key and computes the public key *pk*, and sends the relevant key information for verifying the signature to the blockchain.

Second, the client first encrypts the data file and then splits the encrypted data file $F$ into n blocks, i.e., $F = \{m_1, m_2, \ldots, m_n\}$.

Third, the data tag $Tag_i$ of each block $m_i$ is computed with the private key, and the signature method we use here is the ZSS short signature method. The set of tags of data file $F$ is $T = \{Tag_1, Tag_2, \ldots, Tag_i\}$.

Fourth, the client generates a quad Merkle hash tree with data tags and sends the root *Root* of the tree to the blockchain for storage.

Finally, the client outsources the encrypted data file $F$ and the quad Merkle hash tree to the cloud, and the client deletes the local data file and tags. The cloud will check the integrity of the data block before accepting the outsourced data to prevent malicious clients. The checking process is like the verification phase described below.

The process of the initialization phase is shown in Figure 6.

### 2) VERIFICATION PHASE

First, the client, as a verifier, randomly selects b elements to form the query index set $I = \{v_1, v_2, \cdots v_b\}, b \in [1, n]$, and forms the audit query *chal*, and sends it to the cloud.

Second, the cloud receives the query message *chal* and sends the corresponding series of encrypted data blocks $m_i$, data block signatures $Tag_i$, and random queries $v_i$ and auxiliary information $\sigma$ to the blockchain.

Third, the blockchain verifies the correctness of the signatures using the key information sent by the client, and after successful verification, *Root'* is calculated using the smart contract deployed on the blockchain based on the signatures

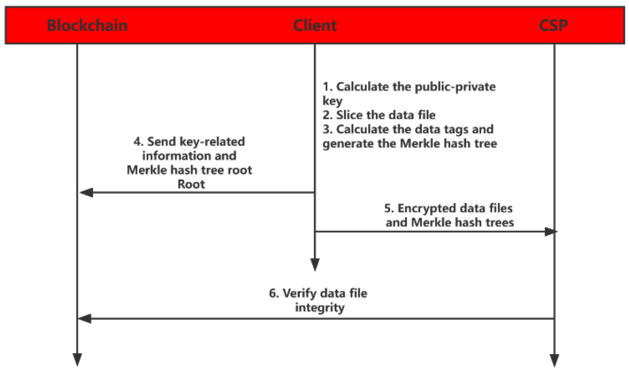

Blockchain
Client
CSP
1. Calculate the public-private key
2. Slice the data file
3. Calculate the data tags and generate the Merkle hash tree
4. Send key-related information and Merkle hash tree root Root
5. Encrypted data files and Merkle hash trees
6. Verify data file integrity

FIGURE 6. The process of initialization phase.

$Tag_i$ and the random queries $v_i$ and the auxiliary information $\sigma$, where $v_1 \leq i \leq v_b$.

Finally, the blockchain compares *Root* and *Root'* by the verifyContract. if *Root* = *Root'*, the data is complete, otherwise, the stored data is corrupted, and the verification result is returned to the client.

The process of the verification phase is shown in Figure 7.

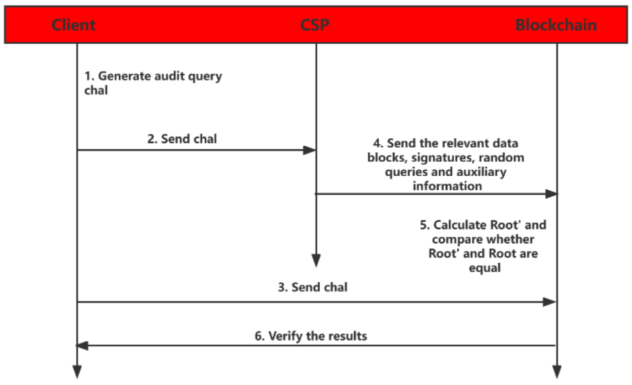

Client
CSP
Blockchain
1. Generate audit query chal
2. Send chal
4. Send the relevant data blocks, signatures, random queries and auxiliary information
5. Calculate Root' and compare whether Root' and Root are equal
3. Send chal
6. Verify the results

FIGURE 7. The process of verification phase.

In this process, the client places the root of the Merkle tree on the blockchain before uploading the data. Due to the tamper-evident nature of the blockchain, no client or cloud node can modify the root stored on the blockchain, which makes the integrity verification more credible. Also, due to the distributed nature of the blockchain, we assume that the data on the blockchain will not be corrupted. Therefore, data integrity verification is more reliable.

## D. SMART CONTRACT

A smart contract is a computer protocol designed to disseminate, validate or enforce contracts in an informational manner. Smart contracts allow trusted transactions to be made without a third party, and these transactions are traceable and irreversible. A smart contract can be defined simply as computer code running on top of a blockchain. It contains a set of rules that determine how the parties involved interact with each other. As long as the pre-defined rules are satisfied, the protocol is automatically executed.

In our scheme, three types of smart contracts are designed to be deployed on the blockchain, and the specific smart contract algorithms are described as follows.

### 1) STORAGECONTRACT

In the initialization phase of our scheme, the client needs to store the root *Root* of the quad Merkle hash tree generated by the data file shards on the blockchain. Therefore, we design and deploy the storage contract on the blockchain for storing the tree root *Root*. We denote the storage contract as storageContract. The blockchain stores the *Root* by executing the storeData() function. The pseudocode of storeData() is shown in Algorithm 1 in Table 1.

TABLE 1. Algorithm 1.

```
Algorithm 1 storeData(data， storageContractInstance )
Input： data， storageContractInstance
Output： null
1.  If storageContractInstance == null then
2.      throw an error： the storageContract was not run
    successful；
3.      return；
4.  else
5.      call the set． sendtansaction() function of
    storageContractInstance to store the data in the
    storageContract；
6.  end if
```

### 2) COMPUTERCONTRACT

After the blockchain receives the verification-related information sent by the cloud, it needs to use this information to calculate the root *Root'* of the quad Merkle tree. We denote the smart contract that computes the root as computeRContract. Unlike computing *Root* in the initialization phase, this smart contract computes *Root'* based on the slice information and its corresponding auxiliary information. The blockchain computes *Root'* by executing the compute root function computeRoot The pseudocode of computeRoot() is shown in Algorithm 2 in Table 2.

TABLE 2. Algorithm 2.

```
Algorithm 2  computeRoot( auxiliary， vi， mi ) → Root'
Input： auxiliary， vi， mi
Output： Root'
1.  If auxiliary == null || vi == null || mi == null then
2.      Root' = null；
3.      print error；
4.  else
5.      Tagi = TagGen(mi)；
6.      Root' = hash( hash( Tagi  + vi )， auxiliary)；
7.  end if
8.  return Root'；
```

### 3) VERIFYCONTRACT

In our designed scheme, the blockchain needs to compare *Root* and *Root'*, the roots of two quad Merkle hash trees, to complete the verification work. Therefore, we design and deploy the verification contract on the blockchain. We denote the verification contract as verifyContract. Blockchain verifies the integrity of data by executing the integrity verification function integrityVerify() The pseudocode of integrityVerify() is shown in Algorithm 3 in Table 3.

**TABLE 3. Algorithm 3.**

```
Algorithm 3  integrityVerity( Root, Root', verifyContractInstance )
Input: Root, Root', verifyContractInstance
Output: bool
1.  If verifyContractInstance == null then
2.      throw an error: the verifyContract was not run
    successful;
3.      return;
4.  else
5.      hash Root and Root' separately to obtain the corresponding
    result R and R';
6.      call the compare. call() function of
    verifyContractInstance to compare R and R';
7.      If R == R' then
8.          return true;
9.      else
10.         return false;
11.     end if
12. end if
```

## V. SECURITY ANALYSIS

The blockchain is a decentralized distributed shared ledger system that combines blocks of data in chronological order to form a specific data structure. From a data perspective, the blockchain is a distributed database on which the data held is tamper-proof and unforgeable. Only nodes with more than 51% of the jointly initiated attacks can change the data on the blockchain. Therefore, blockchain enables secure storage of data. On the other hand, since the blockchain runs automatically, it is no problem to be an auditor, and it is impossible to collude with the cloud. Therefore, when a dispute occurs between the client and the cloud, the record on the blockchain can be used as valid evidence. At the same time, through blockchain instead of TPA, the information to be stored by the client can only be accessed by itself, thus avoiding access to the client's private information during the TPA verification process and safeguarding the client's privacy.

In summary, using blockchain as a third-party authentication platform ensures usability, security, efficiency, and client privacy. The specific analysis is as follows.

### A. AUDIT REASONABLENESS

If the complete data stored by the client is not stored in the cloud, then it cannot pass the verification of the blockchain. This means that any misconduct (such as modification or forgery) can be identified. The blockchain can detect misconduct when an untrusted cloud intentionally sends incorrect or false data information to the blockchain.

1) Anti-forgery attack: Cloud forges data information with the intention of deceiving the blockchain and the client to pass verification.

In the data verification phase, the cloud receives the audit query *chal* from the client and sends the verification information related to the *chal* (e.g., data blocks, auxiliary information, etc.) to the blockchain for verification. If it happens that the data block subject to verification is corrupted, the cloud forges the data block information and sends it to the blockchain to pass the verification. Since the data block signature information is stored through the Merkle hash tree, if the cloud sends forged data block information to the blockchain, it causes a change in the root of the Merkle tree generated by its signatures. On the other hand, the original hash tree root value Root stored on the blockchain will not change. Therefore, even if the cloud wants to forge a data message, it is impossible to pass the verification.

2) Anti-replacement attack: The cloud server replaces the challenged data block with an older version of the previously validated data block or an uncorrupted data block to pass validation. This is not possible to happen in our scheme.

In the data verification phase, the cloud receives the audit query *chal* from the client and sends a series of verification information of the data block associated with the *chal* to the blockchain to pass the verification. If the verified data block stored in the cloud is corrupted, but the cloud server wants to pass the verification, it sends the verification information to the blockchain using the previously verified data block or the uncorrupted data block in the cloud to pass the verification. However, in the verification phase of our scheme, when the client sends an audit query *chal* to the cloud, it also sends a copy to the blockchain. Therefore, when the cloud sends the verification of the data block 10 related to the *chal* to the blockchain, the blockchain compares the verification information with the query *chal* sent by the client. If the verification information is consistent with the queried data block in the *chal*, the verification continues. Otherwise, it directly returns a verification error to the client. Therefore, it is impossible for the cloud server to pass the verification if it wants to replace the challenged data block with an older version of the previously verified data block or an uncorrupted data block.

3) Anti-fraud attack: The client uploads corrupt data blocks to the cloud or uploads incorrect Merkle tree root to the blockchain and then claims that the data is intact to trick the cloud into getting compensation. This is not likely to happen in our scheme.

According to our scheme, the root *Root* of the Merkle hash tree stored on the blockchain cannot be tampered with.

If the verification fails, it means that the data in the cloud is corrupted. Therefore, the cloud should pay compensation to the customer without denying it. Therefore, the cloud can also detect the correctness of the uploaded data when a dishonest customer uploads incorrect data to cheat the cloud for compensation. In our scheme, the cloud performs a verification before storing the client's data, i.e., verifies the integrity of the data to the blockchain. If the verification is successful before storing, it indicates that the client is in good faith. On the one hand, the data sent to the cloud is complete and not corrupted; on the other hand, it indicates that the Merkle root *Root* from the client stored in the blockchain is correct. Otherwise, this client is malicious. Thus, the scheme prevents malicious clients from spoofing the cloud.

### B. DATA PRIVACY

The specific information in the data file is not visible to the cloud.

*Proof:* Privacy protection is a very important aspect in the field of data integrity audit research. It requires that the client's data file information is not compromised. The cloud is an untrusted party that may maliciously sell the stored data block information in the process of storing data to seek greater profit. In our scheme, what the cloud has is the encrypted data file and its signature. The signature set hides the data information well using hash functions, system parameters, client public keys, etc. The specific information of these data blocks is not visible to the cloud, even during public auditing. Therefore, our proposed scheme protects the client's data information very well.

### C. DETECTABILITY

Data stored in the cloud, if corrupted, can be detected with a probability of no less than $1-(\frac{n-x}{n})^y$.

*Proof:* The files stored in the cloud are divided into $n$ data blocks saved in the cloud, and assuming that $x$ data blocks are corrupted (e.g., deleted, modified, etc.), we will extract the detected data blocks as $m$ blocks, then the probability that TPA can detect the corrupted data is $P_z$, where $z = (x \cap m)$. So, it can be derived that

$$P_z = P\{z \geq 1\} = 1 - P\{z=0\} = 1 - \frac{n-x}{n} \times \frac{n-1-x}{n-1} \times \frac{n-2-x}{n-2} \times \cdots \times \frac{n-m+1-x}{n-m+1}.$$

Because of

$$(\frac{n-m+1-x}{n-m+1})^m \leq \frac{n-x}{n} \times \frac{n-1-x}{n-1} \times \cdots \times \frac{n-m+1-x}{n-m+1} \leq (\frac{n-x}{n})^m,$$

so

$$P_z \geq 1 - (\frac{n-x}{n})^m.$$

This probability is equivalent to that in 1,000,000 data blocks, if 1% of the data blocks are damaged, the verifier only selects 300 data blocks, and the detection rate reaches 95%. If 460 data blocks are selected, the detection rate is as high as 99%.

Our method is mainly used for relatively large files, i.e., the number of selected blocks m is much smaller than n. In one challenge, the confidence level $CL = P_z$, and similarly, after $k$ challenges, $CL' = 1 - (1 - CL)^k$. The $CL$ is only related to the anomaly ratio $p = x/n$ and the number of challenged blocks m, not to the file size. Figure 8 shows the $CL$ for different damage rates $p$. From Figure 8, higher confidence levels can be obtained by challenging fewer blocks when $p = 1\%$ and $10\%$. Even for a low $CL$, i.e., $p = 0.1\%$, a single challenge of 550 blocks only yields a $CL = 0.423$. However, considering that our challenge is a repeatable process, i.e., nine times, the confidence level can be increased to $CL' = 1 - (1 - 0.423)^9 = 0.993$, which satisfies our confidence requirement.

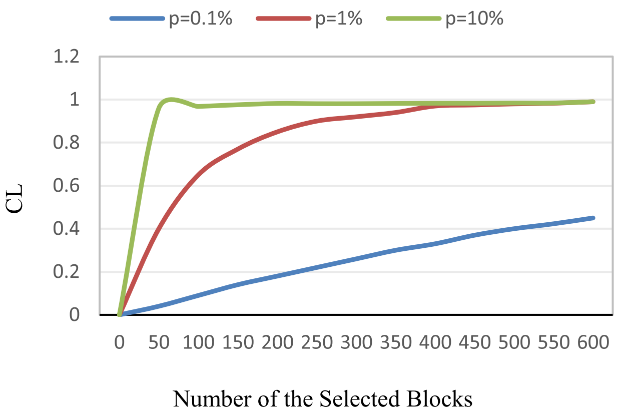


**FIGURE 8.** Confidence level.

## VI. EXPERIMENT

In this section, we evaluate the performance of our scheme by making a comparison with other schemes. Scheme [14] is a multi-copy data integrity verification scheme based on temporal chaos, and scheme [22] is a multi-copy data integrity verification scheme based on identity signature. Scheme [34] is a blockchain cloud storage integrity auditing scheme based on the T-Merkle hash tree. The three schemes are compared in terms of blockchain storage cost, signature generation time, and audit verification time to verify the performance of the schemes in this paper.

### A. BLOCKCHAIN STORAGE COST

There are many blockchain-based data integrity auditing schemes, but most of them store data or labels of data blocks on the blockchain (e.g., scheme [28], scheme [32], scheme [34], and scheme [35], etc.), resulting in a large memory overhead of the blockchain. Our scheme, however, stores only the root of the quad Merkel tree on the blockchain and thus has minimal storage overhead. Our scheme is mainly related to smart contracts, which include the deployment and invocation of smart contracts.

In Ethereum, each transaction consumes a certain amount of gas. We conducted a series of tests where we tested the gas consumption of smart contracts with a different number of files (from 16 to 1024), and the results are shown in Figure 9. The experimental results show that the gas consumption of the contract is independent of the number of files stored. This is also consistent with our theoretical analysis that we only store the Merkel tree root on the blockchain, and the size of the root is independent of the number of files. Therefore, the blockchain storage overhead of our scheme is extremely small. The cost of a smart contract in Ether is equal to the number of consumed gas multiplied by the unit price of gas, which is the same as Ether, so we use the consumption of gas to represent the cost of a smart contract.

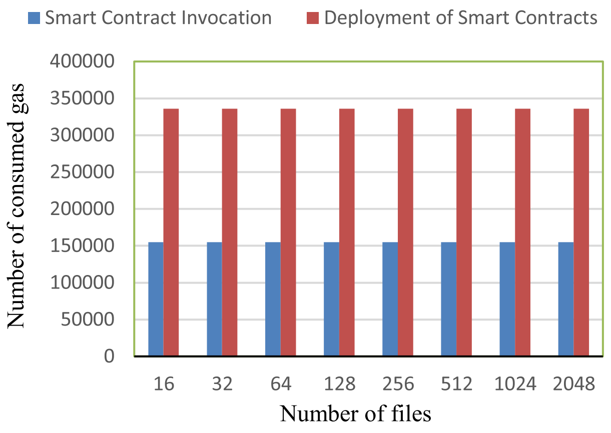


**FIGURE 9. Smart contract gas consumption.**

## B. EXPERIMENTAL RESULTS

Our experiment was implemented on a PC (8-core Intel i7 processor, 16G RAM) based on a 64-bit Ubuntu system (version 20.04.2), the chosen blockchain platform was Ethereum, and the smart contracts were implemented by Solidity programming language. The linear pairing algorithm based on the pairing-based cryptography (PBC) library with a.param (PBC-0.5.14) and the GNU multi-precision arithmetic library (GMP-6.2.1) was used in the experiments, where the base field size was set to 512 bits. We experimented with text files, reading them in binary and dividing them into blocks. The size of the data block is constant at 2KB, so the file size changes depending on the number of data blocks. All simulation results are the average of 100 tests.

In the experiment, signatures are performed using data blocks from 0 to 1000 with an interval of 100. Since scheme [14] and scheme [22], a data block is divided into multiple sectors, but this scheme is divided by data blocks, and no sector division is done, so the sectors of the scheme [14] and scheme [22] are 1, and the number of copies is 1, thus ensuring the fairness of the experiment. Comparing this scheme with scheme [14], scheme [22] and scheme [34] in terms of computational overhead, it is verified that the data block signature generation time increases roughly linearly with the increase of data blocks. The experimental results are shown in Figure 10. In Figure 10, it is also clear that this scheme does have a shorter signature generation time than the other two schemes.

Our scheme also compares the integrity audit overhead of the audit verification phase for 100 to 1000 challenged data blocks with other schemes. Figure 11 shows the computational overhead of the audit verification phase. It can be seen from the figure that there is a linear relationship between the time of the audit verification phase and the number of challenged data blocks for our scheme, scheme [22], scheme [14], and scheme [34], but our scheme is more efficient overall.

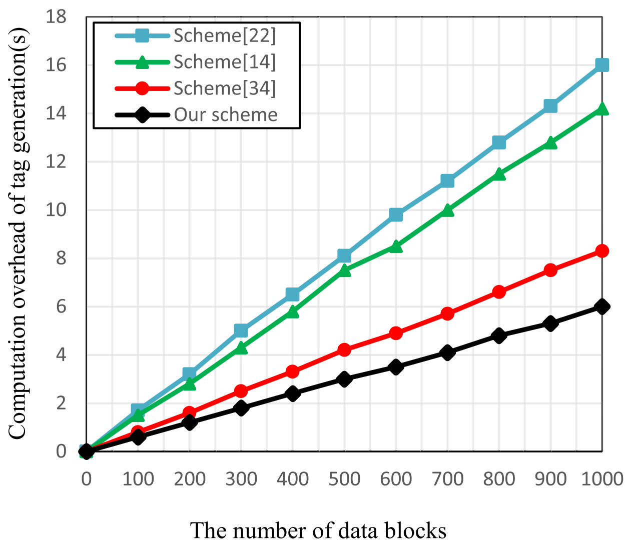


**FIGURE 10. Relationship between tag generation time and the number of data blocks.**

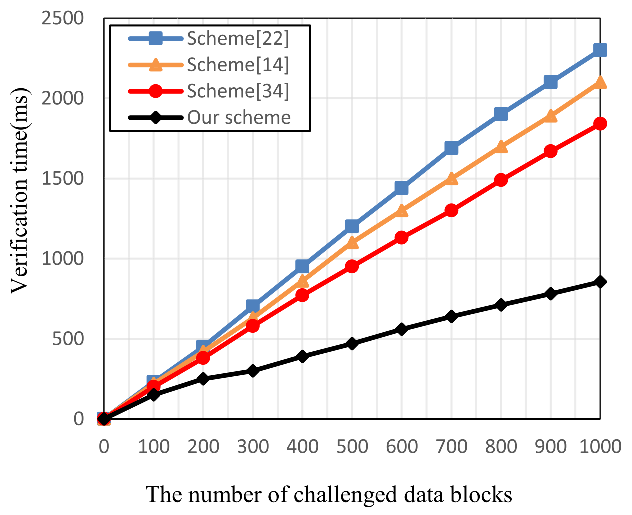


**FIGURE 11. Relationship between audit verification phase time and number of challenged data blocks.**

## VII. CONCLUSION

In this paper, we propose a data integrity auditing scheme based on a quad Merkle hash tree and blockchain. Using blockchain instead of a third-party auditor ensures the reliability of data auditing; the audit based on the quad Merkle hash tree improves the efficiency of computing and storage. At the same time, deploying smart contracts on the blockchain enables automatic verification of auditing activities, allowing us to have a timelier picture of data integrity. The feasibility of our scheme is proved theoretically, and the comparative experiments with other schemes confirm that the scheme outperforms other schemes in terms of blockchain storage cost and computational overhead. The experimental analysis shows that our scheme achieves the expected security and efficiency goals. In future work, we will continue to explore closer integration of blockchain and integrity verification schemes, such as how to avoid huge storage overhead by storing data blocks or data labels in the blockchain.

## REFERENCES

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

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

[3] J. Toutouh, A. Muñoz, and S. Nesmachnow, "Evolution oriented monitoring oriented to security properties for cloud applications," in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, Hamburg, Germany, Aug. 2018, pp. 1–7.

[4] J. Ni, K. Zhang, Y. Yu, and T. Yang, "Identity-based provable data possession from RSA assumption for secure cloud storage," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1753–1769, Jun. 2022.

[5] Y. Yang, Y. Chen, F. Chen, and J. Chen, "An efficient identity-based provable data possession protocol with compressed cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 1359–1371. 2022.

[6] G. Bian and J. Chang, "Certificateless provable data possession protocol for the multiple copies and clouds case," *IEEE Access*, vol. 8, pp. 102958–102970, 2020.

[7] F. Levy-dit-Vehel and M. Roméas, "A framework for the design of secure and efficient proofs of retrievability," in *Proc. Int. Conf. Cryptogr., Codes Cyber Security*. Cham, Switzerland: Springer, 2022, pp. 83–103.

[8] X. Zhang, S. Liu, and S. Han, "Proofs of retrievability from linearly homomorphic structure-preserving tag natures," *Int. J. Inf. Comput. Secur.*, vol. 11, no. 2, pp. 178–202, 2019.

[9] B. Sengupta and S. Ruj, "Efficient proofs of retrievability with public verifiability for dynamic cloud storage," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 138–151, Oct. 2020.

[10] J. Li, J. Li, D. Xie, and Z. Cai, "Secure auditing and deduplicating data in cloud," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2386–2396, Aug. 2016.

[11] Y. Yu, Y. Li, B. Yang, W. Susilo, G. Yang, and J. Bai, "Attribute-based cloud data integrity auditing for secure outsourced storage," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 377–390, Jun. 2020.

[12] N. Garg, S. Bawa, and N. Kumar, "An efficient data integrity auditing protocol for cloud computing," *Future Gener. Comput. Syst.*, vol. 109, pp. 306–316, Aug. 2020.

[13] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 10, pp. 2402–2415, Oct. 2017.

[14] M. Long, Y. Li, and F. Peng, "Integrity verification for multiple data copies in cloud storage based on spatiotemporal chaos," *Int. J. Bifurcation Chaos*, vol. 27, no. 4, Apr. 2017, Art. no. 1750054.

[15] G. Xu, S. Han, Y. Bai, X. Feng, and Y. Gan, "Data tag replacement algorithm for data integrity verification in cloud storage," *Comput. Secur.*, vol. 103, Apr. 2021, Art. no. 102205.

[16] S. Anbuchelian, C. M. Sowmya, and C. Ramesh, "Efficient and secure auditing scheme for privacy preserving data storage in cloud," *Cluster Comput.*, vol. 22, pp. 1–9, Dec. 2017.

[17] L.-B. Wu, J. Wang, D.-B. He, and M.-K. Khan, "Crypt analysis of an identity-based public auditing protocol for cloud storage," *Frontiers Inf. Technol. Electron. Eng.*, vol. 18, no. 12, pp. 1972–1977, 2017.

[18] G. Bian, R. Zhang, and B. Shao, "Identity-based privacy preserving remote data integrity checking with a designated verifier," *IEEE Access*, vol. 10, pp. 40556–40570, 2022.

[19] Y. Ji, B. Shao, J. Chang, and G. Bian, "Flexible identity-based remote data integrity checking for cloud storage with privacy preserving property," *Cluster Comput.*, vol. 25, no. 1, pp. 337–349, Feb. 2022.

[20] H. Yan and W. Gui, "Efficient identity-based public integrity auditing of shared data in cloud storage with user privacy preserving," *IEEE Access*, vol. 9, pp. 45822–45831, 2021.

[21] P. Yi, J. Li, C. Liu, J. Han, H. Wang, Y. Zhang, and Y. Chen, "An efficient identity-based signature scheme with provable security," *Inf. Sci.*, vol. 576, pp. 790–799, Oct. 2021.

[22] J. Li, H. Yan, and Y. Zhang, "Efficient identity-based provable multi-copy data possession in multi-cloud storage," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 356–365, Mar. 2019, doi: 10.1109/TCC.2019.2929045.

[23] Y. Zhang, C. Xu, X. Lin, and X. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 923–937, Sep. 2021.

[24] A. Muñoz and A. Maña, "TPM-based protection for mobile agents," *Secur. Commun. Netw.*, vol. 4, no. 1, pp. 45–60, Jan. 2011.

[25] Y. Yuan, J. Zhang, W. Xu, and Z. Li, "Identity-based public data integrity verification scheme in cloud storage system via blockchain," *J. Supercomput.*, vol. 78, no. 6, pp. 8509–8530, Apr. 2022.

[26] G. Xie, Y. Liu, G. Xin, and Q. Yang, "Blockchain-based cloud data integrity verification scheme with high efficiency," *Secur. Commun. Netw.*, vol. 2021, pp. 1–15, Apr. 2021.

[27] S. Cao, G. Zhang, P. Liu, X. Zhang, and F. Neri, "Cloud-assisted secure eHealth systems for tamper-proofing EHR via blockchain," *Inf. Sci.*, vol. 485, pp. 427–440, Jun. 2019.

[28] K. Hao, J. Xin, Z. Wang, and G. Wang, "Outsourced data integrity verification based on blockchain in untrusted environment," *World Wide Web*, vol. 23, no. 4, pp. 2215–2238, Jul. 2020, doi: 10.1007/s11280-019-00761-2.

[29] Y. Ren, Y. Leng, J. Qi, P. K. Sharma, and A. Tolba, "Multiple cloud storage mechanism based on blockchain in smart homes," *Future Gener. Comput. Syst.*, vol. 115, pp. 304–313, Feb. 2021.

[30] Y. Ren, F. Zhu, P. K. Sharma, T. Wang, J. Wang, O. Alfarraj, and A. Tolba, "Data query mechanism based on hash computing power of blockchain in Internet of Things," *Sensors*, vol. 20, no. 1, p. 207, Dec. 2020.

[31] L. Zhu, Y. Wu, K. Gai, and K.-K. R. Choo, "Controllable and trustworthy blockchain-based cloud data management," *Future Gener. Comput. Syst.*, vol. 91, pp. 527–535, Feb. 2019.

[32] H. Wang, Q. Wang, and D. He, "Blockchain-based private provable data possession," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2379–2389, Oct. 2019.

[33] P. Wei, D. Wang, Y. Zhao, S. K. S. Tyagi, and N. Kumar, "Blockchain data-based cloud data integrity protection mechanism," *Future Gener. Comput. Syst.*, vol. 102, pp. 902–911, Jan. 2020.

[34] K. He, C. Huang, J. Shi, X. Hu, and X. Fan, "Enabling decentralized and dynamic data integrity verification for secure cloud storage via T-merkle hash tree based blockchain," *Mobile Inf. Syst.*, vol. 2021, pp. 1–17, Nov. 2021.

[35] Z. Quanyu, S. Chen, Z. Liu, T. Baker, and Y. Zhang, "Blockchain-based privacy-preserving remote data integrity checking scheme for IoT information systems," *Inf. Process. Manage.*, vol. 57, no. 6, 2020, Art. no. 102355, doi: 10.1016/j.ipm.2020.102355.

**ZHENPENG LIU** received the M.S. and B.S. degrees from the School of Cyberspace Security and Computer, Hebei University, and the Ph.D. degree from Tianjin University. He is a Professor with the School of Cyberspace Security and Computer, Hebei University. His main research interests include network information security and outlier detection.

**LELE REN** is currently pursuing the M.S. degree with the School of Cyberspace Security and Computer, Hebei University, Baoding, Hebei. Her research interests include data integrity verification and privacy protection.

**YONGJIANG FENG** is currently pursuing the M.S. degree with the School of Cyberspace Security and Computer, Hebei University, Baoding, Hebei. His research interests include cloud computing, cloud storage, and data integrity verification.

**SHUO WANG** is currently pursuing the M.S. degree with the School of Cyberspace Security and Computer, Hebei University, Baoding, Hebei. His research interests include network information security and data integrity verification.

**JIANHANG WEI** received the B.S. degree from the School of Mathematics and Information Science, Hebei University, and the M.E. degree from the School of Education, Hebei University. Currently, he is a Senior Experimental Engineer with the Information Technology Center, Hebei University. His research interests include big data and information security.