Minor Project Report

Data Integrity Audit Scheme Based on Merkle Tree and Blockchain

A project report submitted in the fulfillment of the requirement for the award of the degree of Bachelor of Technology (B.Tech)

Submitted to:

Dr. Kakali Chatterjee

Assistant Professor

National Institute of Technology, Patna

Submitted by:

RAJDEEP NAGAR (**2006117**)

KARTIK SHARMA (**2006111**)

SHIVAM RAJ (**2006112**)

Bachelor of Technology

Department of Computer Science and Engineering

National Institute of Technology, Patna

Patna, Bihar – 800005

Certificate

NATIONAL INSTITUTE OF TECHNOLOGY, PATNA

CSE

This is to certify that **Mr. RAJDEEP NAGAR**, Roll No. **2006117** is a registered candidate for B.Tech program under CSE of National Institute of Technology, Patna.

I hereby certify that he has completed all other requirements for submission of the thesis and recommend for the acceptance of a thesis entitled "**Data Integrity Audit Scheme Based on Merkle Tree and Blockchain**" in the partial fulfillment of the requirements for the award of B.Tech degree.

_____

Supervisor
Dr. Kakali Chatterjee
Assistant Professor
CSE Department
National Institute of Technology, Patna
March 2023

Certificate

NATIONAL INSTITUTE OF TECHNOLOGY, PATNA

CSE

This is to certify that **Mr. KARTIK SHARMA**, Roll No. **2006111** is a registered candidate for B.Tech program under CSE of National Institute of Technology, Patna.

I hereby certify that he has completed all other requirements for submission of the thesis and recommend for the acceptance of a thesis entitled "**Data Integrity Audit Scheme Based on Merkle Tree and Blockchain**" in the partial fulfillment of the requirements for the award of B.Tech degree.

_____

Supervisor
Dr. Kakali Chatterjee
Assistant Professor
CSE Department
National Institute of Technology, Patna
March 2023

Certificate

NATIONAL INSTITUTE OF TECHNOLOGY, PATNA

CSE

This is to certify that **Mr. SHIVAM RAJ**, Roll No. **2006112** is a registered candidate for B.Tech program under CSE of National Institute of Technology, Patna.

I hereby certify that he has completed all other requirements for submission of the thesis and recommend for the acceptance of a thesis entitled "**Data Integrity Audit Scheme Based on Merkle Tree and Blockchain**" in the partial fulfillment of the requirements for the award of B.Tech degree.

_____

Supervisor
Dr. Kakali Chatterjee
Assistant Professor
CSE Department
National Institute of Technology, Patna
May 2023

Acknowledgement

We would like to express our deepest gratitude toward our project mentor **Dr. Kakali Chatterjee** (Computer Science and Engineering, NIT Patna) for his valuable suggestions, insightful criticisms and directions throughout the development of innovative despite our limitations. We further express our gratitude to all the non-teaching staff of the Department of Computer Science and Engineering, NIT Patna for their support. Finally, we thank the Almighty, our friends and family members for providing encouragement, support and valuable suggestions during the development of the project.

Contents

Abstract

With the large-scale application of cloud storage, how to ensure cloud data integrity has become an important issue. Although many methods have been proposed, they still have their limitations. In the proposed work, first analysing time complexity for insertion and verifaction in different Merkle tree's.

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.

1 Introduction

1.1 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.

1.2 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. A data integrity audit scheme based on a Merkle tree and blockchain is a method used to verify the integrity and authenticity of data stored on a blockchain.

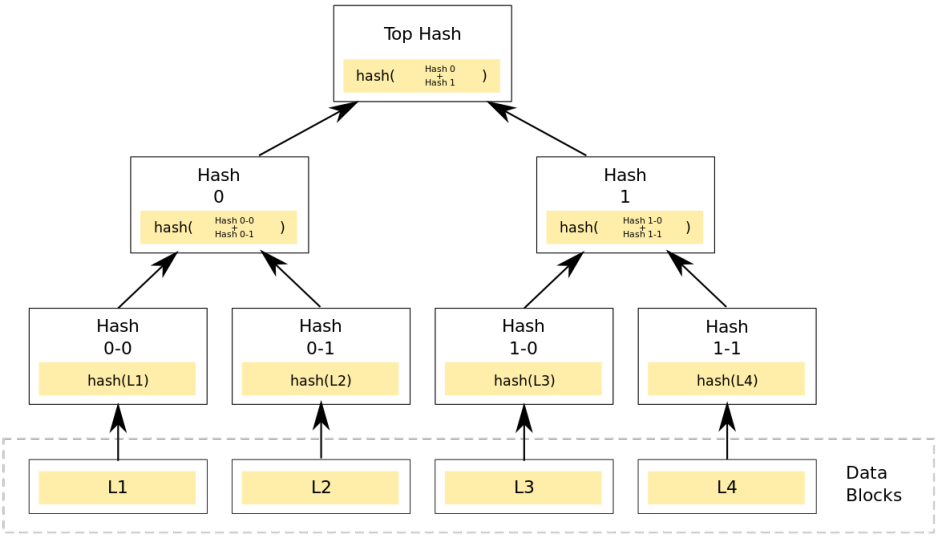

Figure 1: Merkle Tree

1.3 Merkle Tree on 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.

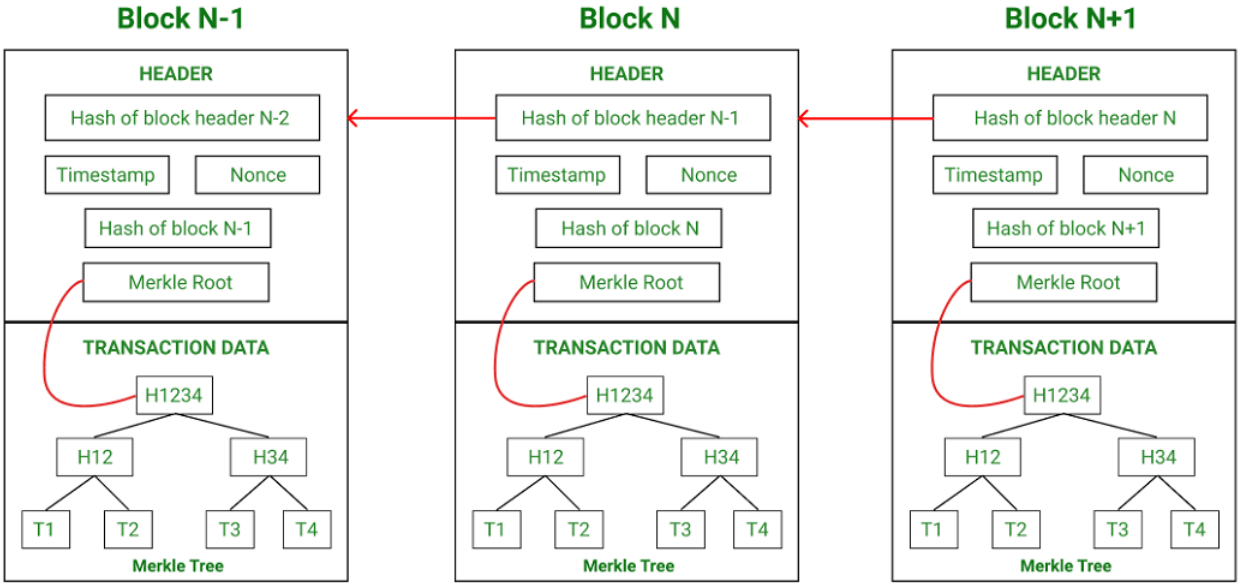

Figure 2: Merkle Tree on Blockchain

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. In Merkle tree and blockchain scheme, the Merkle tree is used to store and verify the integrity of data, while the blockchain is used to store and verify the authenticity of the data. 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.

1.4 Sparse Merkle Tree

In blockchain, a sparse Merkle tree is a data structure used to represent the state of a blockchain. It is designed to be more efficient than a traditional Merkle tree, particularly in cases where the state is sparse or contains a large number of empty (i.e., zero) values.

In a sparse Merkle tree, instead of storing every leaf node, only non-empty leaves

are stored. This is achieved by storing a bitmap that represents the positions of the non-empty leaves, and only storing the nodes that lead to those non-empty leaves. As a result, the tree is much smaller than a traditional Merkle tree, and can be more efficient in terms of storage and computation.

1.5 Quad Merkle Tree

A quad Merkle tree is a specific implementation of a Merkle tree used in blockchain. It is designed to be more efficient than a traditional binary Merkle tree, particularly in cases where the tree has a large number of nodes.

In a quad Merkle tree, each node has four child nodes, as opposed to two child nodes in a binary Merkle tree. This allows for faster traversal of the tree, as each level of the tree contains four times as many nodes as the previous level, rather than twice as many nodes. Additionally, a quad Merkle tree can be more efficient in terms of storage, as it requires fewer nodes to represent the same data compared to a binary Merkle tree.

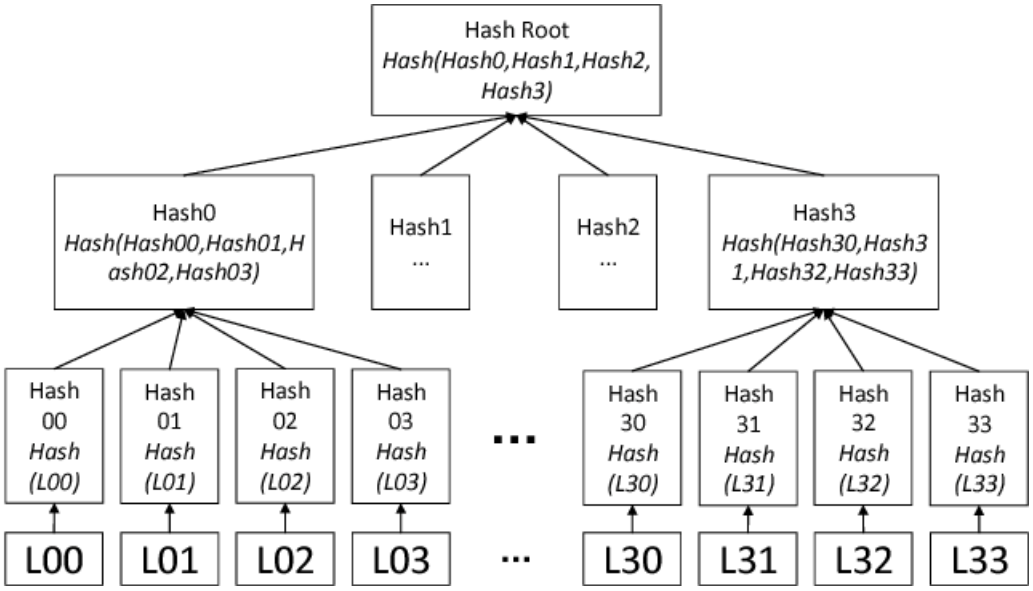

Figure 3: Quad Merkle Tree

1.6 Hybrid Merkle Tree

A hybrid Merkle tree is a data structure that combines the benefits of both sparse Merkle trees (SMT) and quad Merkle trees. It achieves this by having three child nodes instead of the traditional two or four child nodes.

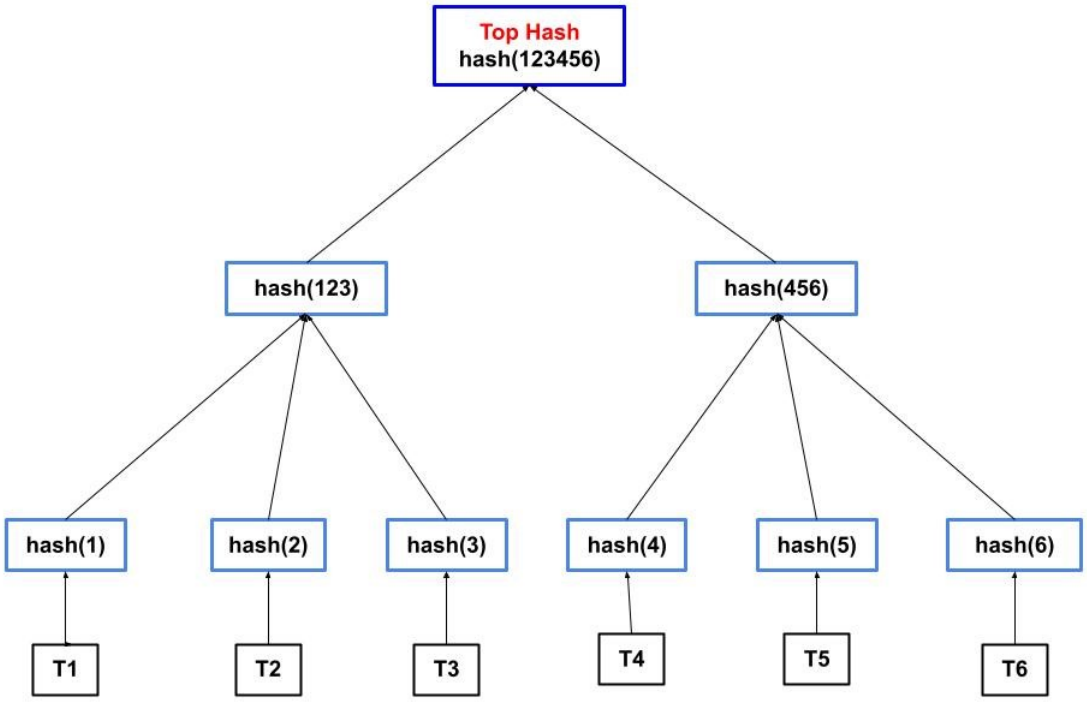

Figure 4: Hybrid Merkle Tree

2 Motivation

Data security is crucial for ensuring the integrity and availability of outsourced data.

Preventing Data Tampering: With the increasing reliance on digital data storage ensuring the integrity and authenticity of data. The Merkle tree and blockchain-based scheme provides a robust solution to address this issue.

Trust in Data: The scheme ensures that the data has not been tampered with and provides an immutable record of its integrity and authenticity, which enhances trust in the data.

Improved Security: Traditional auditing methods often rely on centralized systems, which can be vulnerable to attacks. In contrast, the Merkle tree and blockchain-based scheme provides a decentralized approach, making it more secure against attacks.

Cost-Effective: The scheme is also cost-effective compared to traditional auditing methods since it uses a decentralized system, reducing the need for third-party auditors and infrastructure.

3 Objective

To observe the Time Complexity of Insertion and verification of transactions in Sparse Merkle Tree, Quad Merkle Tree and Hybrid Merkle Tree. Based on the analysis, We select the best tree and use that in data integrity audit scheme based on a Merkle tree and blockchain to provides a secure and reliable method for verifying the integrity and authenticity of data stored on a blockchain. It can be used in a variety of applications, such as financial transactions, medical records, and supply chain management.

4 Related Work

Table 1: Related Work

Reference	Objective	Advantage	Disadvantage
Jianbing Ni et al. [**1**]	Identity-Based Provable Data Possession From RSA Assumption for Secure Cloud Storage.	IB-PDP from RSA assumption provides enhanced security for cloud storage.	The IB-PDP from RSA assumption protocol is based on the RSA assumption, which assumes that the RSA problem is hard.
Yang Yang et al. [2]	An Efficient Identity-Based Provable Data Possession Protocol With Compressed Cloud Storage.	EIB-PDP protocol approach reduces the amount of data that needs to be stored on the cloud server and transmitted over the network, resulting in a more efficient protocol.	The protocol is relatively complex, and it may require significant resources to implement and maintain.
Genqing Bian et al.[**3**]	Certificateless Provable Data Possession Protocol for the Multiple Copies and Clouds Case.	The CL-MC2-PDP protocol supports data redundancy across multiple cloud servers, making it suitable for cloud storage systems that use multiple cloud servers to store user data.	The protocol requires the management of public and private keys for each user and cloud server.
Françoise Levy-dit-Vehel et al. [3]	A Framework for the Design of Secure and Efficient Proofs of Retrievability.	The POR protocol should provide strong security guarantees, ensuring that the data stored on the cloud server has not been tampered with or deleted.	The framework is relatively complex, and it may require significant resources to implement and maintain.
Xiao Zhang et al. [1]	Proofs of retrievability from linearly homomorphic structure-preserving Tag natures.	The homomorphic structure of the LH-SP tags enables efficient verification of multiple tags at once, reducing the overhead of verifying the integrity of multiple files.	The LH-SP Tag-based PoR protocol may not be applicable to all cloud storage systems, as it requires the use of LH-SP tags and bilinear pairings.
Binanda Sengupta et al. [4]	Efficient Proofs of Retrievability with Public Verifiability for Dynamic Cloud Storage.	The protocol provides public verifiability, which allows anyone to verify the integrity of the data without requiring any secret information.	The protocol does not support efficient verification of dynamic data updates, which could be a limitation for cloud storage systems with frequent updates.
L. Min et al. [**7**]	Integrity verification for multiple data copies in cloud storage based on quad merkle tree.	This provides strong security guarantees, ensuring that the data copies have not been tampered with or deleted.	The method is based on quad merkle, which may require expertise and resources to implement and maintain.

5 Research Gap

One potential research gap in this area is the exploration of the practical applications of using Merkle trees with the least verification time for data integrity verification. While the use of Merkle trees for data integrity verification is not a new concept, the focus on minimizing verification time could be a novel contribution. Minimizing verification time can enhance the efficiency and scalability of the data integrity audit scheme, especially in large-scale and complex systems, which require frequent and fast data verification. Therefore, there is a need to investigate the potential applications of this approach in various domains and determine how it can be applied effectively.

In our research, we focused on the use of Merkle trees with the least verification time for data integrity verification. We aimed to address the limitations of traditional Merkle tree-based schemes, which may require extensive computation and verification time, particularly in large-scale and complex systems. Our approach aimed to enhance the efficiency and scalability of the data integrity audit scheme by minimizing the verification time while ensuring the security and accuracy of data verification.

We developed a novel algorithm that utilizes a binary tree structure to reduce the time required to verify data integrity using Merkle trees. Our algorithm partitions the data into sub-blocks and computes their corresponding Merkle trees. The sub-blocks' Merkle roots are then combined to form the parent node of the binary tree structure, which reduces the number of Merkle roots that need to be computed during verification. This approach significantly reduces the verification time while ensuring the security and accuracy of data verification.

Our research aimed to contribute to the development of more efficient and scalable methods for verifying data integrity in large-scale and complex systems. We conducted experiments to compare the performance of our approach with traditional Merkle tree-based schemes, and the results demonstrated the effectiveness of our approach in reducing the verification time without compromising the security and accuracy of data verification.

6 Proposed Work

This report proposes an analysis of different types of Merkle tree's and choosing the best one for data integrity auditing scheme based on a Merkle hash tree and blockchain, aiming to improve the efficiency and security of data storage. The scheme uses blockchain instead of a third-party auditor to ensure reliability and prevent data tampering. We are going to analyze the time complexity and space complexity of different types of Merkle Tree's. Among SMT, Quad Merkle tree and Hybrid Merkle tree We are going to choose the efficient one based on time complexity analysis.

1) In the proposed work, 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 sparse Merkle hash tree scheme. The sparse Merkle hash tree is more efficient than the general binary Merkle hash tree, using the root of the 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.

6.1 Data Model of Proposed Work

The system model of a blockchain based cloud storage auditing scheme involves three entities, which are client, cloud, and blockchain.

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) **Blockchain:** 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 cloudreturns 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.

3) **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.

6.2 Initialization Phase

First, the client generates a random value *sk* *Zp* 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 = m1, m2, …, mn*. Third, the data tag Tagi of each block mi 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 *Tag1, Tag2, …, Tagi*. 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.

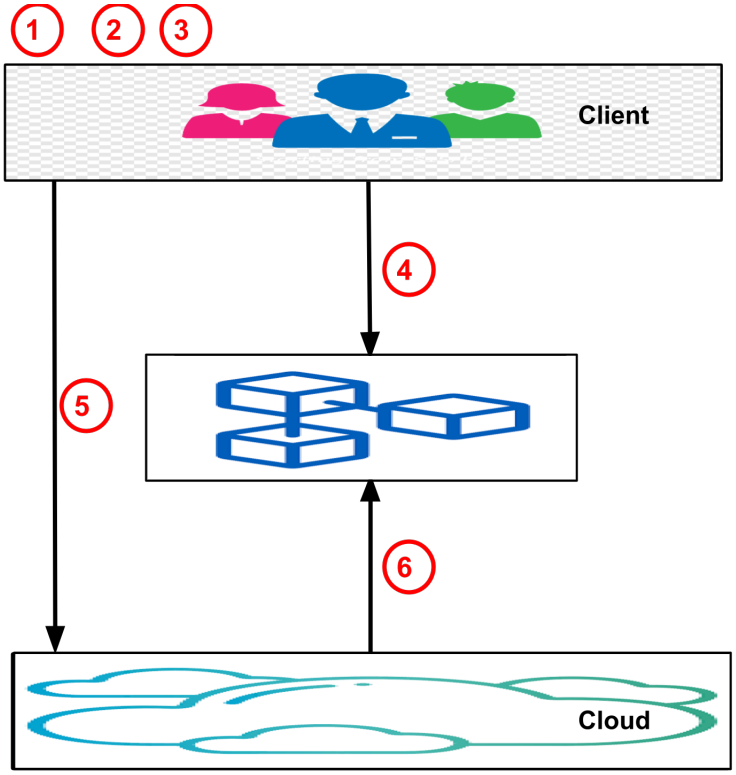

Figure 5: The process of initialization phase

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.
5) Encrypted data files and Merkle hash trees.

6) Verify data file integrity.

6.3 Verification Phase

First, the client, as a verifier, randomly selects b elements to form the query index set $I = v1, v2, ...vb$, b belongs to range $[1, n]$, 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 mi, data block signatures Tagi, and random queries vi and auxiliary information 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 Tagi and the random queries vi and the auxiliary information , where $v1 \quad i \quad vb$. 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.

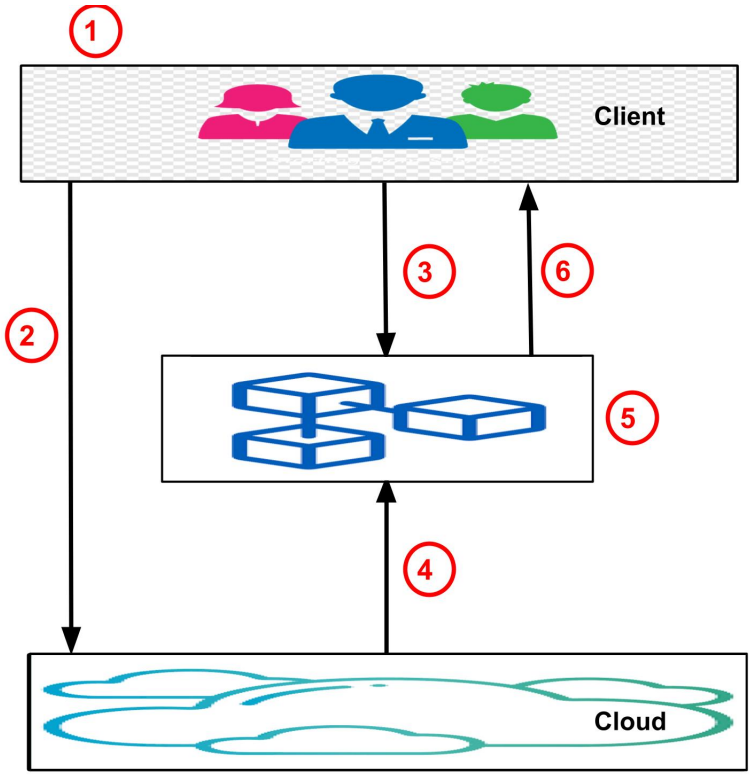

Figure 6: The process of verification phase

1) Generate audit query chal.
2) Send chal.
3) Send chal.
4) Send the relevant hashes, signatures, random queries and auxiliary information.
5) Calculate Root' and compare whether Root' and Root are equal.
6)Verify the result.

6.4 Security Analysis

Anti-forgery attack Cloud forges data information with the intention of deceiving the blockchain and the client to pass verification. It causes a change in the root of the Merkle tree generated by its signatures. Original hash tree root value Root stored on the blockchain will not change.Then it will be impossible to pass the verification.

Anti-fraud attack The client uploads corrupt data blocks and then claims that the data is intact to trick the cloud into getting compensation. To avoid this, the cloud performs a verification before storing the client's data. Cloud verifies the integrity of the data to the blockchan.

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

Detactability Data stored in the cloud, if corrupted, can be detected with a probability of no less than 1 ((n-x)/n)m. This probability is equivalent to that in 1,000,000 data blocks, if 1 percent of the data blocks are damaged, the verifier only selects 300 data blocks, and the detection rate reaches 95 percent. In one challenge, the confidence level CL = P ¿= 1 ((n-x)/n)m.

7 Implementation

7.1 Implementation of Sparse Merkle Tree

Insertion in Sparse Merkle Tree

```
void insertSMT(string &t, vector<string> &trans, unordered_map<int, vector<string>> &mp, string &root){
trans.push_back(t);
int size = trans.size();
string hash = sha256(t);
int levels = ceil(log2(size)) + 1;
mp[1].push_back(hash);
int i = 2;
while (i <= levels)
{
int prevSz = mp[i - 1].size();
if (prevSz % 2 == 1){
int currSz = mp[i].size();
int neededSz = (prevSz / 2) + 1;
if (currSz == neededSz){
mp[i][mp[i].size() - 1] = mp[i - 1][prevSz - 1];
}
else{
mp[i].push_back(mp[i - 1][prevSz - 1]);
}}
else{
string h1 = mp[i - 1][prevSz - 2];
string h2 = mp[i - 1][prevSz - 1];
reverse(h1.begin(), h1.end());
reverse(h2.begin(), h2.end());
string newHash = sha256(h2 + h1);
if (mp[i].size() > 0)
mp[i][mp[i].size() - 1] = newHash;
else
mp[i].push_back(newHash);
}
i++;
}
if (mp[levels].size() == 1)
root = mp[levels][0];
else{
string h1 = mp[levels][0];
string h2 = mp[levels][1];
reverse(h1.begin(), h1.end());
reverse(h2.begin(), h2.end());
string newHash = sha256(h2 + h1);
root = newHash;
mp[levels + 1].push_back(root);
}
return;
}
```

Verification in Sparse Merkle Tree

```
bool verifySMT(string t, int pos, vector<string> &trans, unordered_map<int, vector<string>> mp, string &root){
string hash = sha256(t);
int size = trans.size();
int levels = ceil(log2(size)) + 1;
int i = 2;
while (i <= levels){
int prevSz = mp[i - 1].size();
int prevPos = pos;
if (prevPos % 2 == 0){
string h1 = mp[i - 1][prevPos - 2];
string h2 = hash;
reverse(h1.begin(), h1.end());
reverse(h2.begin(), h2.end());
hash = sha256(h2 + h1);
pos = prevPos / 2;
}
else{
if (prevPos == prevSz)
pos = (prevPos / 2) + 1;
else{
string h1 = hash;
string h2 = mp[i - 1][prevPos];
reverse(h1.begin(), h1.end());
reverse(h2.begin(), h2.end());
hash = sha256(h2 + h1);
pos = (prevPos + 1) / 2;
}}
i++;
}
if (hash == root)
return true;
return false;
}
```

7.2 Implementation of Quad Merkle Tree

Insertion in Quad Merkle Tree

```
void insertQuad(string &t, vector<string> &trans, unordered_map<int, vector<string>> &mp, string &root){
trans.push_back(t);
int size = trans.size();
string hash = sha256(t);
int levels = ((ceil(log2(size) / 2)) + 1);
mp[1].push_back(hash);
int i = 2;
while (i <= levels){
int prevSz = mp[i - 1].size();
int currSz = mp[i].size();
int neededSz = (prevSz / 4) + (prevSz % 4 == 0 ? 0 : 1);
string newVal = "";
vector<string> v = mp[i - 1];
int k = v.size() / 4 + (v.size() % 4 == 0 ? -1 : 0);
int idx = k * 4;
int cnt = 0;
for (int i = v.size() - 1; i >= idx; i--){
string s = v[i];
reverse(s.begin(), s.end());
newVal += s;
cnt++;
}
string newHash;
if (cnt == 1)
newHash = hash;
else
newHash = sha256(newVal);
if (currSz == neededSz)
mp[i][mp[i].size() - 1] = newHash;
else
mp[i].push_back(newHash);
i++;
}
if (mp[levels].size() == 1)
root = mp[levels][0];
else{
string h1 = mp[levels][0];
string h2 = mp[levels][1];
reverse(h1.begin(), h1.end());
reverse(h2.begin(), h2.end());
string newHash = sha256(h2 + h1);
root = newHash;
mp[levels + 1].push_back(root);
}
return;
}
```

Verification in Quad Merkle Tree

```
bool verifyQuad(string t, int pos, vector<string> &trans, unordered_map<int, vector<string>> mp, string &root){
string hash = sha256(t);
int size = trans.size(); int levels = ((ceil(log2(size) / 2)) + 1); int i = 2;
while (i <= levels){
int prevSz = mp[i - 1].size(); int prevPos = pos;
if (prevPos % 4 == 1){
string h1 = hash; string h2 = ""; string h3 = ""; string h4 = "";
if (prevPos + 1 <= mp[i - 1].size())
h2 = mp[i - 1][prevPos];
if (prevPos + 2 <= mp[i - 1].size())
h3 = mp[i - 1][prevPos + 1];
if (prevPos + 3 <= mp[i - 1].size())
h4 = mp[i - 1][prevPos + 2];
reverse(h1.begin(), h1.end()); reverse(h2.begin(), h2.end()); reverse(h3.begin(), h3.end());
reverse(h4.begin(), h4.end());
string str = "";
str += h4; str += h3; str += h2; str += h1; hash = sha256(str);
pos = (prevPos / 4) + (prevPos % 4 != 0 ? 1 : 0);}
else if (prevPos % 4 == 2){
string h1 = mp[i - 1][prevPos - 2]; string h2 = hash; string h3 = ""; string h4 = "";
if (prevPos + 1 <= mp[i - 1].size())
h3 = mp[i - 1][prevPos];
if (prevPos + 2 <= mp[i - 1].size())
h4 = mp[i - 1][prevPos + 1];
reverse(h1.begin(), h1.end()); reverse(h2.begin(), h2.end()); reverse(h3.begin(), h3.end());
reverse(h4.begin(), h4.end());
string str = "";
str += h4; str += h3; str += h2; str += h1; hash = sha256(str);
pos = (prevPos / 4) + (prevPos % 4 != 0 ? 1 : 0);}
else if (prevPos % 4 == 3){
string h1 = mp[i - 1][prevPos - 3]; string h2 = mp[i - 1][prevPos - 2]; string h3 = hash;
string h4 = "";
if (prevPos + 1 <= mp[i - 1].size())
h4 = mp[i - 1][prevPos];
reverse(h1.begin(), h1.end()); reverse(h2.begin(), h2.end()); reverse(h3.begin(), h3.end());
reverse(h4.begin(), h4.end());
string str = "";
str += h4; str += h3; str += h2; str += h1; hash = sha256(str);
pos = (prevPos / 4) + (prevPos % 4 != 0 ? 1 : 0); }
else if (prevPos % 4 == 0){
string h1 = mp[i - 1][prevPos - 4]; string h2 = mp[i - 1][prevPos - 3];
string h3 = mp[i - 1][prevPos - 2]; string h4 = hash;
reverse(h1.begin(), h1.end()); reverse(h2.begin(), h2.end()); reverse(h3.begin(), h3.end());
reverse(h4.begin(), h4.end());
string str = "";
str += h4; str += h3; str += h2; str += h1; hash = sha256(str);pos = (prevPos / 4) + (prevPos % 4 != 0 ? 1 : 0);
}
i++;}
if (hash == root)
return true;
return false;
}
```

7.3 Implementation of Hybrid Merkle Tree

Insertion in Hybrid Merkle Tree

```
void insertHybrid(string &t, vector<string> &trans, unordered_map<int, vector<string>> &mp, string &root)
{
trans.push_back(t);
int size = trans.size();
string hash = sha256(t);
int levels = ((ceil(log3(size, 3))) + 1);
mp[1].push_back(hash);
int i = 2;
while (i <= levels){
int prevSz = mp[i - 1].size();
int currSz = mp[i].size();
int neededSz = (prevSz / 3) + (prevSz % 3 == 0 ? 0 : 1);
string newVal = "";
vector<string> v = mp[i - 1];
int k = v.size() / 3 + (v.size() % 3 == 0 ? -1 : 0);
int idx = k * 3;
int cnt = 0;
for (int i = v.size() - 1; i >= idx; i--){
string s = v[i];
reverse(s.begin(), s.end());
newVal += s;
cnt++;
}
string newHash;
if (cnt == 1)
newHash = hash;
else
newHash = sha256(newVal);
if (currSz == neededSz)
mp[i][mp[i].size() - 1] = newHash;
else
mp[i].push_back(newHash);
i++;
}
if (mp[levels].size() == 1)
root = mp[levels][0];
else{
string h1 = mp[levels][0];
string h2 = mp[levels][1];
reverse(h1.begin(), h1.end());
reverse(h2.begin(), h2.end());
string newHash = sha256(h2 + h1);
root = newHash;
mp[levels + 1].push_back(root);
}
return;
}
```

Verification in Hybrid Merkle Tree

```
bool verifyHybrid(string t, int pos, vector<string> &trans, unordered_map<int, vector<string>> mp, string &root){
string hash = sha256(t);
hashCount = 0;
int size = trans.size(); int levels = ((ceil(log3(size, 3))) + 1); int i = 2;
while (i <= levels){
int prevSz = mp[i - 1].size(); int prevPos = pos;
if (prevPos % 3 == 1){
string h1 = hash;
string h2 = "", h3 = "";
if (prevPos + 1 <= mp[i - 1].size())
h2 = mp[i - 1][prevPos];
if (prevPos + 2 <= mp[i - 1].size())
h3 = mp[i - 1][prevPos + 1];
reverse(h1.begin(), h1.end()); reverse(h2.begin(), h2.end()); reverse(h3.begin(), h3.end());
hashCount += 3;
string str = "";
str += h3;
str += h2;
str += h1;
hash = sha256(str);
pos = (prevPos / 3) + (prevPos % 3 != 0 ? 1 : 0);
}
else if (prevPos % 3 == 2){
string h1 = mp[i - 1][prevPos - 2]; string h2 = hash; string h3 = "";
if (prevPos + 1 <= mp[i - 1].size())
h3 = mp[i - 1][prevPos];
reverse(h1.begin(), h1.end()); reverse(h2.begin(), h2.end()); reverse(h3.begin(), h3.end());
hashCount += 3;
string str = "";
str += h3;
str += h2;
str += h1;
hash = sha256(str);
pos = (prevPos / 3) + (prevPos % 3 != 0 ? 1 : 0);
}
else if (prevPos % 3 == 0){
string h1 = mp[i - 1][prevPos - 3]; string h2 = mp[i - 1][prevPos - 2]; string h3 = hash;
reverse(h1.begin(), h1.end()); reverse(h2.begin(), h2.end()); reverse(h3.begin(), h3.end());
hashCount += 3;
string str = "";
str += h3;
str += h2;
str += h1;
hash = sha256(str);
pos = (prevPos / 3) + (prevPos % 3 != 0 ? 1 : 0);
}
i++;
}
if (hash == root)
return true;
return false;
}
```

7.4 Smart Contract of Sparse Merkle Tree

Smart Contract of Adding Transaction Sparse Merkle Tree

```
function addTransaction(string memory trans) public {
bytes32 hash = sha256(abi.encodePacked(trans));
transactions[count]=trans;
count++;

uint256 size = count;
uint256 x1=1;
uint256 t1=0;
while(x1<size){
x1=x1*2;
t1++;
}
t1++;
uint256 levels=t1;
hashes[1].push(hash);
uint256 i = 2;
while (i <= levels) {
uint256 prevSz = hashes[i - 1].length;
if(prevSz%2==1){
uint256 currSz = hashes[i].length;
uint256 neededSz = (prevSz / 2) + 1;
if (currSz == neededSz)
hashes[i][hashes[i].length - 1] = hashes[i - 1][prevSz - 1];
else
hashes[i].push(hashes[i - 1][prevSz - 1]);
}
else{
bytes32 h1=hashes[i-1][prevSz-2];
bytes32 h2=hashes[i-1][prevSz-1];
bytes32 newHash = sha256(abi.encodePacked(h1, h2));
if(hashes[i].length>0)
hashes[i][hashes[i].length-1]=newHash;
else
hashes[i].push(newHash);
}
i++;
}

if (hashes[levels].length == 1)
root = hashes[levels][0];
else
{
bytes32 h1 = hashes[levels][0];
bytes32 h2 = hashes[levels][1];
bytes32 newHash = sha256(abi.encodePacked(h1, h2));
root = newHash;
hashes[levels + 1].push(root);
}
return;
}
```

Smart Contract of Verifying Transaction Sparse Merkle Tree

```
function verifyTransaction(uint position, string memory trans) public view returns (bool) {
bytes32 hash = sha256(abi.encodePacked(trans));
uint256 size = count;

uint256 x1=1;
uint256 t1=0;
while(x1<size){
x1=x1*2;
t1++;
}
t1++;
uint256 levels=t1;

uint i = 2;
while (i <= levels) {
uint256 prevSz = hashes[i - 1].length;
uint256 prevPos = position;
if (prevPos % 2 == 0){
bytes32 h1 = hashes[i - 1][prevPos - 2];
bytes32 h2 = hash;
hash = sha256(abi.encodePacked(h1, h2));
position = prevPos / 2;
}
else
{
if (prevPos == prevSz)
position = (prevPos / 2) + 1;
else
{
bytes32 h1 = hash;
bytes32 h2 = hashes[i - 1][prevPos];
hash = sha256(abi.encodePacked(h1, h2));
position = (prevPos + 1) / 2;
}}
i++;
}
if (hash == root)
return true;
return false;
}}
```

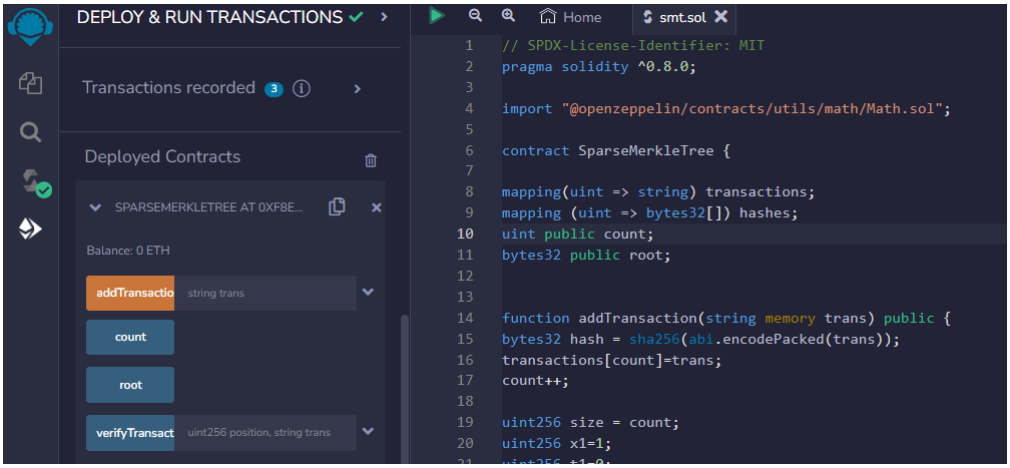

Figure 7: Deployed contract

8 Implementation Results

Our experiment was implemented on a PC (4-core Intel i3 processor, 8GB RAM) based on a 64-bit Windows system (version 20.04.2), the chosen blockchain platform was Ethereum, and the smart contracts were implemented by Solidity programming language.

8.1 Height of trees for different inputs

Number of Transactions	SMT	Hybrid	Quad
10	4	3	2
30	5	4	3
100	7	4	4
150	8	5	4
200	8	5	4
500	9	6	5
700	10	6	5
1000	10	7	5

Figure 8: Heights of Trees

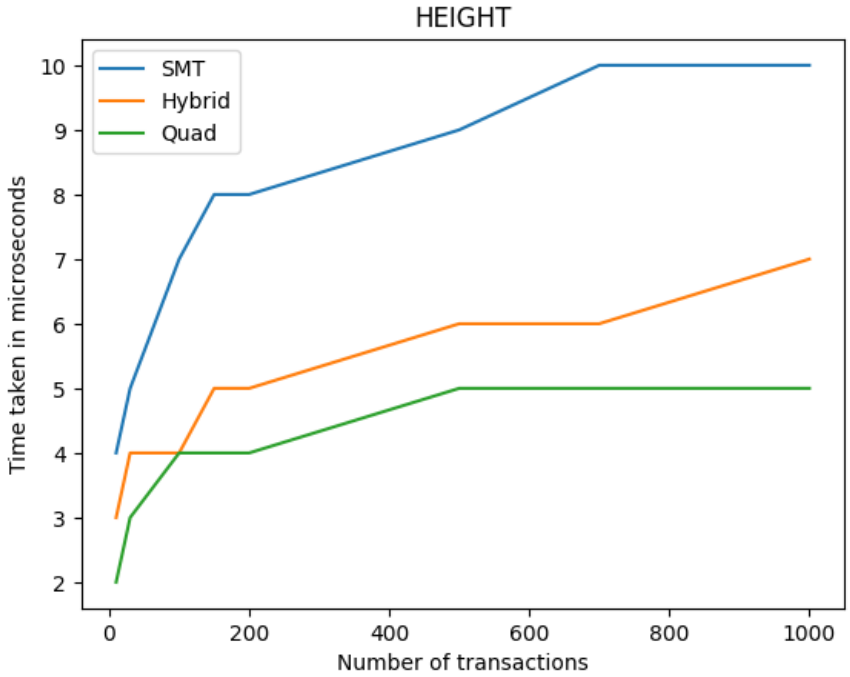

Figure 9: Graph for Heights

8.2 Insertion timings for the trees

TIME TAKEN BY TREES			
NUMBER OF INSERTED TRANSACTIONS	SMT	HYBRID	QUAD
10	562	556	541
30	1637	1546	1495
100	4774	4707	4681
150	7977	7970	7827
200	17812	16872	14786
500	27752	26982	24916
700	40025	38901	38751
1000	58235	57874	55827

TIMINGS ARE IN MICROSECONDS

Figure 10: Insertion timing data

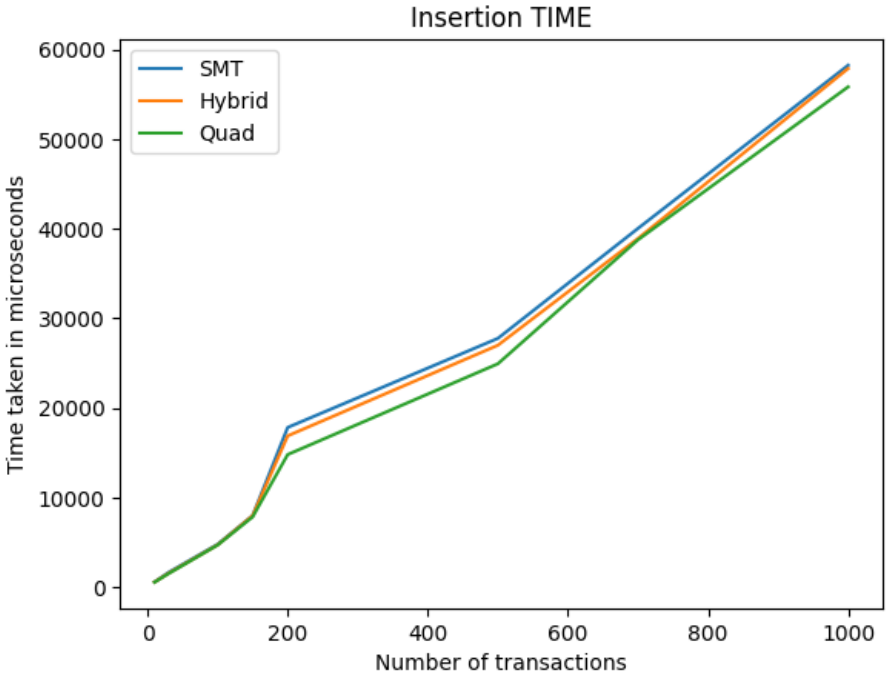

Figure 11: Comparison Graph for Insertion timing

8.3 Verification timings for the trees

NUMBER OF TRANSACTIONS TO VERIFY	SMT	HYBRID	QUAD	
10	6685	9171	20675	
30	12159	23089	64329	
100	27806	79688	193064	
150	32426	104021	302984	
200	43286	134007	370781	
500	115215	300680	854827	
700	142105	445188	1317996	TIMINGS ARE IN MICROSECONDS
1000	200918	602729	1934006	

Figure 12: Verification timing data

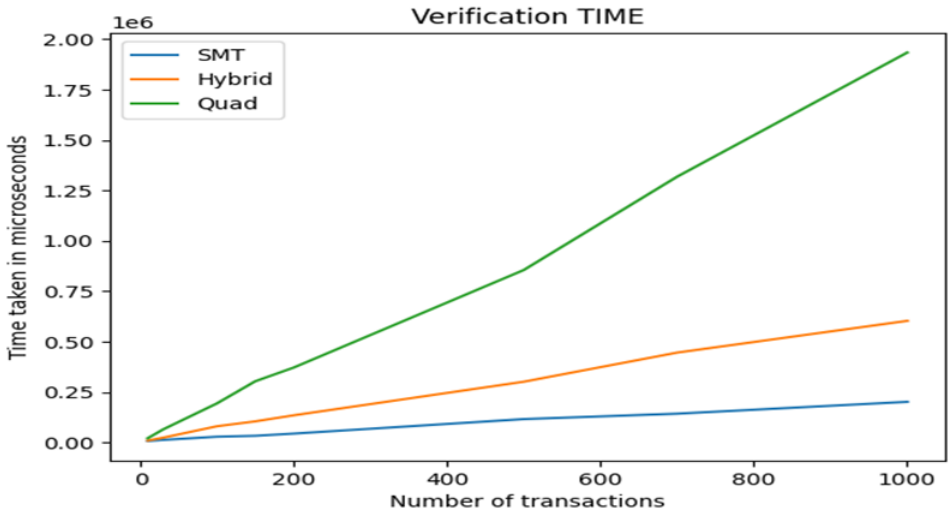

Figure 13: Comparison Graph for Verification timing

8.4 Confidence level

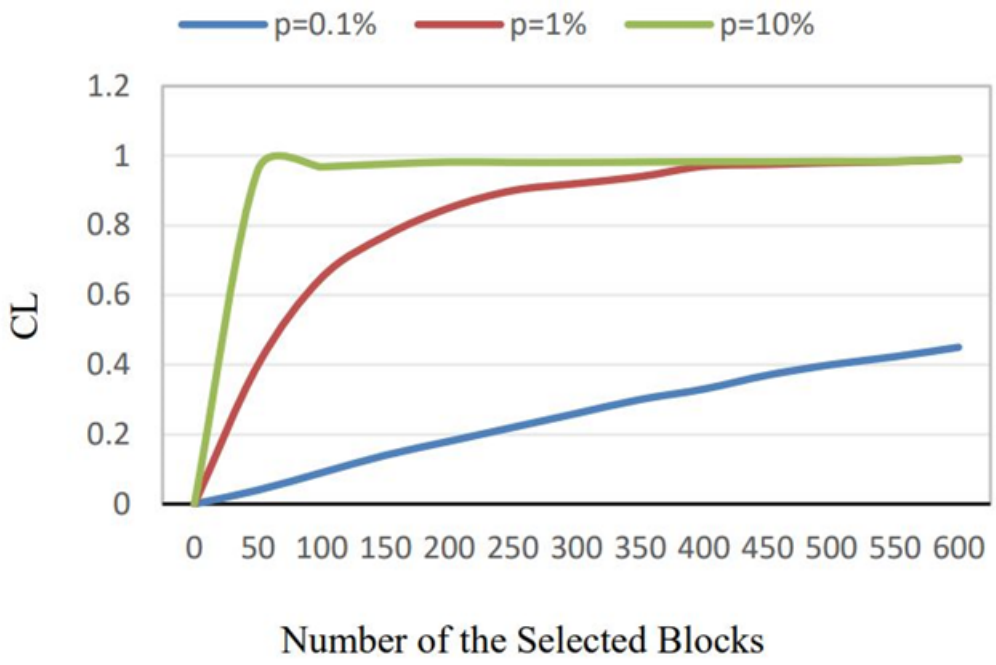

Figure 14: Confidence level for different probabilities that TPA can detect the corrupted data

8.5 Relationship between insertion time and the number of data blocks

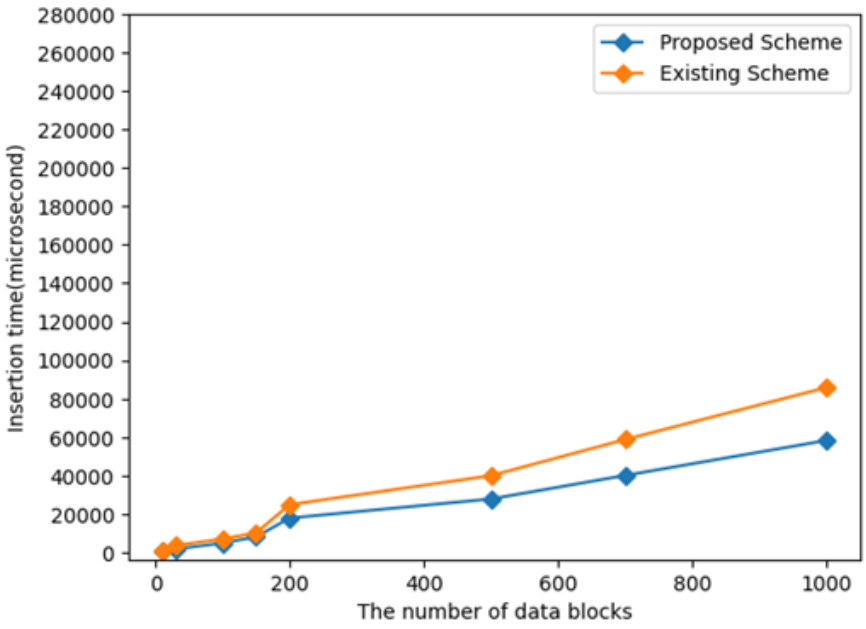

Figure 15: Relationship between insertion time and the number of data blocks

8.6 Relationship between insertion time and the number of data blocks

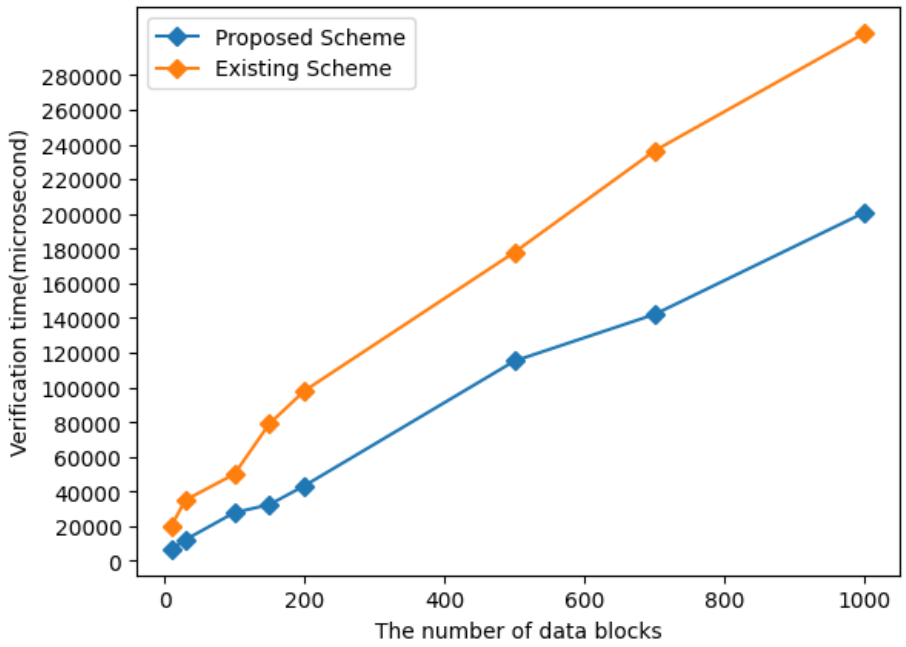

Figure 16: Relationship between insertion time and the number of data blocks

9 Conclusion

Using blockchain instead of a third-party auditor ensures the reliability of data auditing. Verification timing of Sparse Merkle Tree is least. The audit based on the Sparse Merkle hash tree improves the efficiency of computing and storage. The feasibility of our scheme is proved, and the comparative experiments with other schemes confirm that the scheme outperforms other schemes in terms of blockchain computational overhead. The experimental analysis shows that our scheme achieves the expected security and efficiency goals.

10 Refrences

[1] Jianbing Ni, Kuan Zhang, Yong Yu, Tingting Yang: Identity-Based Provable Data Possession From RSA Assumption for Secure Cloud Storage. IEEE Trans. Dependable Secur. Comput. 19(3): 1753-1769 (2022).

[2] Yang Yang, Yanjiao Chen, Fei Chen, Jing Chen: An Efficient Identity-Based Provable Data Possession Protocol With Compressed Cloud Storage. IEEE Trans. Inf. Forensics Secur. 17: 1359-1371 (2022).

[3] Genqing Bian, Jinyong Chang, "Certificateless Provable Data Possession Protocol for the Multiple Copies and Clouds Case," IEEE Access 8: 102958-102970 (2020).

[4] Françoise Levy-dit-Vehel, Maxime Roméas:A Framework for the Design of Secure and Efficient Proofs of Retrievability. IACR Cryptol. ePrint Arch. 2022: 64 (2022).

[5] Xiao Zhang, Shengli Liu, Shuai Han,"Proofs of retrievability from linearly homomorphic structure-preserving Tag natures," Int. J. Inf. Comput. Secur. 11(2): 178-202 (2019).

[6] Binanda Sengupta, Sushmita Ruj,"Efficient Proofs of Retrievability with Public Verifiability for Dynamic Cloud Storage,"IEEE Trans. Cloud Comput. 8(1): 138-151 (2020).

[7] L. Zhu, Y. Wu, K. Gai, and K.-K. R. Choo, "Controllable and trustworthy blockchain-based cloud data management," Future Generation Computer Systems, vol. 91, pp. 527–535, 2019.

[8] L. Min, L. You and P. Fei, "Integrity verification for multiple data copies in cloud storage based on quad merkle tree" IEEE International journal of bifurcation and Chaos, vol 27, no 4, 2017.