Blockchain Based Data Integrity Verification in P2P Cloud Storage

Dongdong Yue[1], Ruixuan Li[1] ✉ , Yan Zhang[2], Wenlong Tian[1], and Chengyi Peng[1]
[1]Intelligent and Distributed Computing Laboratory, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
[2]School of Computing, Engineering and Mathematics, Western Sydney University, Sydney, Australia
Email: rxli@hust.edu.cn

***Abstract*—With the popularity of cloud storage, how to verify the integrity of data on the cloud has become a challenging problem. Traditional verification framework involves the Third Party Auditors (TPAs) which are not entirely credible. In this paper, we present a framework for blockchain-based data integrity verification in P2P cloud storage, making verification more open, transparent, and auditable. In this framework, we present Merkle tree for data integrity verification, and analyze the system performance under different Merkle trees structures. Furthermore, we develop rational sampling strategies to make sampling verification more effective. Moreover, we discuss the optimal sample size to tradeoff the conflict between verification overhead and verification precision, and suggest two efficient algorithms of order of verification. Finally, we conduct a series of experiments to evaluate the schemes of our framework. The experimental results show that our schemes can effectively improve the performance of data integrity verification.**

***Keywords*-P2P Cloud Storage, Blockchain, Data Integrity Verification, Sampling, Merkle Trees**

I. Introduction

Due to the rapid growth of information sharing and exchange, more and more companies and individual users choose to store their data on the cloud. Traditional data privacy and integrity is ensured through data encryption, multiple signatures, anonymous mechanisms and so on. However, users lose control of these data when these data are stored on the cloud. Therefore, how to verify the integrity of the data stored on the cloud becomes an important problem. The data storage and integrity verification workflow framework under traditional cloud storage is shown as Fig. 1. In this framework, there are three objects: Clients, Cloud Storage Servers (CSS), and Third Party Auditor (TPA) [17]. The client stores his own data on the CSS, and sends relevant information to the TPA to verify the integrity of the data. When data integrity verification is performed, the CSS will submit the proofs to the TPA. Finally, the TPA verifies the integrity of the cloud-stored data based on these proofs and the user's previously transmitted useful information.

The rise of Peer-to-Peer (P2P) cloud storage, which exploiting a large amount of idle disk space, makes it possible to rent cheap storage space. In a P2P cloud storage system, each user can be either a client that rents storage space or a lender that lends his own idle storage space. Users can obtain cheap storage space, and lenders can benefit from lending their idle space. Sia [15] and Storj [18] are two mature P2P cloud storage platforms. Each user in these platforms can share its own storage space and gain revenue from the sharing. The redundant backup mechanism in these platforms makes the data storage more reliable.

Starting from an article by Satoshi Nakamoto in 2008 [11], Bitcoin has entered our horizon and triggered an upsurge in blockchain technology. In a simple term, blockchain is a distributed database that includes transactions, blocks, consensus mechanisms, smart contracts, and so on. Each work in the blockchain is recorded in the form of a transaction. Multiple transactions form a block, and multiple blocks are linked together to form a blockchain. The header field of each block contains the hash of the previous block, thus forming an ordered chain. The advantage of blockchain technology is that it provides a decentralized, open, transparent, auditable, and tamper-proof record. All blockchain participating nodes can verify the transactions recorded on the chain. These transactions are permanently recorded on the chain and cannot be maliciously modified. The consensus mechanism in the blockchain ensures that the state of the entire blockchain is consistent without the participation of any third parties. Smart contracts are contracts stored in the blockchain. When the system meets the contract execution conditions, the contract automatically executes the corresponding content. The emergence of smart contracts makes the blockchain more intelligent.

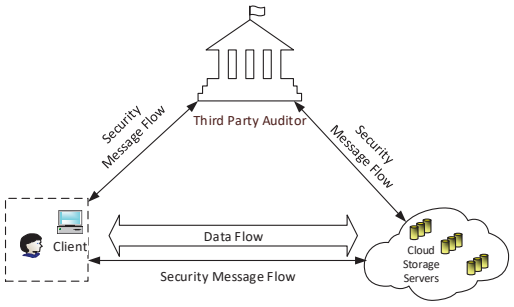


Fig. 1. Data storage and integrity verification on the cloud under a traditional architecture. There is only data flow between client and cloud storage servers. Third party auditor assists clients in verifing data integrity.

In the traditional scenario where the TPA is introduced for data integrity verification, the TPA may not be completely trusted. However, there are also trust issues in verification scenarios that do not involve TPA. From the perspective of CSS, malicious Clients may intentionally claim that their data is not completely preserved by CSS, thus extorting damages from CSS. From the Clients perspective, the dishonest CSS may still claim to guarantee the integrity of the data without saving the complete data, thus harming the Clients interests. Therefore, we can introduce blockchain for data integrity verification in the case of mutual distrust between CSS and Clients. Sia [15] is a P2P cloud storage platform that combines blockchain for data integrity verification. On this platform, blockchain is used to record relevant information for data integrity verification. Such information will be stored permanently on the blockchain and cannot be modified, which makes the verification results more reliable. However, Sia [15] neither provided a complete verification mechanism, nor considered how to select partial data shards for verification. While in the case of limited resources or high real-time requirements, it is necessary to verify the integrity of the whole data by verifying partial data shards. Therefore, how to select partial data shards and guarantee the performance of data integrity verification is a problem worth studying.

In this paper, we firstly propose a general data integrity verification framework to solve this problem for blockchain based P2P cloud storage. Then we analyze the performance of Merkle trees with different structures, and propose a sampling strategy to select shards for validation. The main contributions of this paper are summarized as follows.

- We propose a general data integrity verification framework for blockchain based P2P cloud storage. This framework solves the problem of untrustworthy in traditional verification mechanism. In this framework, clients and cloud servers that do not trust each other can interact.
- Based on the proposed framework, we use Merkle trees for data integrity verification based on blockchain and analyze the performance of Merkle trees under different structures. Then, we present a sampling strategy and discover an optimum sample size to verify data integrity. These make it more effective to verify the integrity of data when only a part of data can be verified because of limited computation resources.
- We conduct extensive simulations to evaluate the performance of the proposed framework by implementing a prototype system. Simulation results demonstrate the feasibility of the proposed framework and validates our theoretical analysis.

The rest of this paper is organized as follows. In Section II, we discuss the related work from three aspects. Then, Section III describes the detailed design of our framework. Experimental studies are presented in Section IV. Finally, Section V concludes the paper with some remarks.

II. Related Work

This section firstly reviews the P2P cloud storage. Then, we introduce how data integrity verification works in traditional cloud storage. Finally, the research work on applying blockchain to verify data integrity is elaborated.

A. P2P Cloud Storage

The mainstream cloud storage systems, such as Google's GFS (Google File System) [3], Amazon's elastic cloud, and open source HDFS (Hadoop Distributed File System), have adopted a similar centralized architecture, in which there is a huge risk of single point failure. The central server is easy to become the bottleneck of the system. Once the central server collapses, it may cause the whole cloud storage service to be unavailable. Many features of P2P systems, such as non centralization, scalability, robustness, high performance and load balancing, can solve this kind of problem. Ji-Yi *et al.* proposed a general model of P2P based cloud storage system, which can provide higher quality of cloud storage services [8].

Some other researchers work on data management in P2P cloud storage. As cloud computing is generally regarded as the technology enabler for Internet of Things, Teing *et al.* tried to ensure the most effective collection of evidence from P2P cloud enabled IoT infrastructure [14]. Since cloud severs and users usually locate outside the trusted domain of data owners, P2P storage cloud brings new challenges for data integrity and access control when data owners store data on it. To address this issue, He *et al.* designed a ciphertext-policy attribute-based encryption scheme and a proxy re-encryption scheme [7]. Based on these schemes, they further proposed a secure, efficient and fine-grained data access control mechanism for P2P storage cloud. However, there is little work on data integrity verification in P2P cloud storage.

B. Data Integrity Verification in the Traditional Cloud Storage

There are mainly two types of traditional data integrity verification mechanisms. One is Provable Data Possession (PDP); the other is Proofs of Retrievability (POR). PDP can quickly verify whether the data stored on the cloud is intact, while POR can restore the damaged data when the data integrity is compromised. The basic PDP authentication method is proposed by Deswarte *et al* [6]. Before the user uploads his own data, he uses the Hash-based Message Authentication Code (HMAC) to calculate the Message Authentication Code (MAC) value of the data and saves it at local. When verifying these data, the user first downloads the data stored on the cloud, then calculates the MAC value of the downloaded file, and compares it with the MAC value previously saved to determine whether the data integrity is guaranteed.

Although this mechanism is simple, directly downloading complete data requires a lot of resources and may lead to leakage of data privacy. Then Seb *et al*. proposed a block-based scheme to reduce the computational overhead [12]. Due to the deterministic verification method, the verification result may not be completely correct. Then Ateniese *et al.*

proposed using probabilistic strategies to complete the integrity verification. They used the homomorphic properties of RSA signature mechanism, gathered evidence in a very small value, which greatly reduced the communication overhead [1]. Subsequently, Curtmola *et al*. implemented the data integrity verification mechanism in the case of multiple copies, but it didn't support the dynamic operation of data [5]. Ateniese *et al.* first considered the dynamic operation of data. They presented simple modified mechanism of the PDP based on their previous work [1], making it support dynamic data manipulation [2]. However, this mechanism does not support insert data. In response to this problem, Wang *et al.* implemented a PDP mechanism that supports full dynamic operation. This mechanism uses the Merkle tree to guarantee the correctness of the data block, and uses the Boneh-Lynn-Shacham (BLS) signature to guarantee the correctness of the data block value [17]. Later, they also proposed a privacy protection verification scheme that uses random masking techniques to make TPA unable to know the data information provided by cloud service providers.

Although the PDP authentication mechanism can efficiently verify the integrity of data, it cannot recover invalid data. Juels *et al.* proposed a sentinel-based POR mechanism [9]. Nevertheless, it can only conduct a limited number of verifications. Subsequently, Shacham *et al.* used the BLS short message signature mechanism to construct homomorphic verification tags, which can reduce the communication overhead for verification [13]. However, it is difficult to be implemented. Wang *et al.* proposed using the linear features of the error correction code to support partial dynamic operations, while it could not support the dynamic insertion of data [16]. Chen *et al.* optimized Wang's mechanism and used the Reed-Solomon erasure code technique to recover the failed data, which can improve the recovery efficiency, but increase the computational cost [4].

C. Blockchain based Data Integrity Verification

The problems of incomplete trust caused by traditional data integrity verification make it an inevitable trend to integrate blockchain technology into data integrity verification of cloud storage. Liu *et al.* applied the blockchain technology to the Internet of Things (IoT) and proposed a blockchain-based data integrity service framework. Without relying on TPA in this framework, data owners and data consumers can be provided with more reliable data integrity verification [10]. However, their work is to target at IoT and is not applicable to scenarios with P2P cloud storage. How to design a universal data integrity verification framework in P2P cloud storage is a worthy research issue. It is because that each node can be either a storage provider or a storage renter under the combination of P2P cloud storage and blockchain. Therefore, this paper focuses on proposing a general and practical data integrity verification framework combined with the blockchain under the P2P cloud storage scenario.

III. The Proposed Method

In this section, we firstly introduce our framework for data integrity verification. Then, we describe the structure of the Merkle trees. The performance of different structures of Merkle trees are analyzed in terms of computation overhead and communication overhead. Finally, we detailedly illustrate some strategies of sampling verification and propose the method about calculating the best sample size.

A. Data Integrity Verification Framework

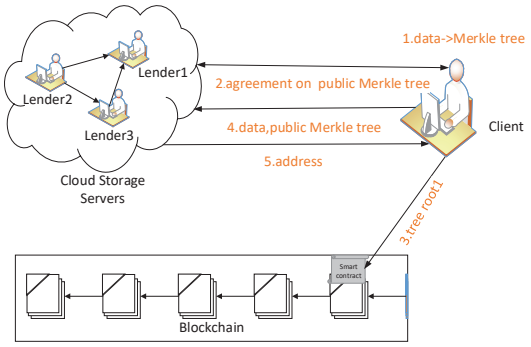
The framework of data storage and integrity verification under blockchain based P2P cloud storage is shown as Fig. 2. In this framework, there are three entites: Clients, Cloud Storage Servers (CSS), and Blockchain (BC). Clients upload their own data to the CSS and use BC to verify data integrity. The overall workflow is divided into two stages. As shown in Fig. 2(a), there are five steps in the preparation stage. In the first step, the client will slice his data into several shards, then uses these shards to construct a hash Merkle trees. In the second step, the client and CSS will agree on the hash Merkle trees. In the third step, the client will store the root of this hash tree denoted as $root1$ on the blockchain. In the fourth step, the client uploads his data and public Merkle trees to CSS. In the fifth step, CSS return the address that stores the client's data to client. As shown in Fig. 2(b), there are also five steps in the verification phase. In the first step, the client will send an challenge number si to CSS, which selects shard i to verify. In the second step, CSS use hash function to calculate a hash Digest i', according to si and shard i. In the third step, CSS send Digest i' and the corresponding auxiliary information to BC. In the fourth step, the smart contract on the blockchain will calculate a new hash root denoted as $root2$, and compare $root1$ with $root2$. If they are equal, the data integrity has been guaranteed; otherwise, the data integrity has been corrupted. In the last step, the BC will return the verify result to the client.

In this framework, clients will place the root of the Merkle trees on the blockchain before uploading the data. Due to the non tamperability property in blockchain, any client or CSS cannot modify the root stored on the blockchain, which makes integrity verification more credible. At the same time, due to the distributed nature of the blockchain, there is little possibility that the data on the blockchain will be damaged. Hence, the data integrity verification is more reliable.

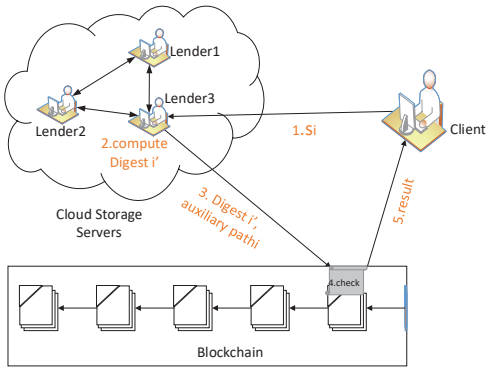
B. Structure of the Merkle Tree

The advantage of using Merkle trees to verify the data integrity is that the entire data file can be verified by a small segment of the entire data shards, which is relatively small regardless of the size of the original file. The structure of Merkle trees is shown as Fig. 3. The public part of this tree needs to be uploaded to the P2P CSS to assist in validating each data shard of the private part. The private part of this tree is composed of data shards $shard_i$ and random challenges r_i. The random challenges can only be sent to the P2P CSS when the client needs to verify the corresponding data shards.

Therefore, the private part is locally saved by the clients. The data uploaded by clients to P2P CSS are the data shards after data slicing. Since it is a tree structure, we can study the Merkle trees with different branches. This paper analyzes the different structures of Merkle trees, then discusses the communication overhead and computational overhead of the system under these structures.



(a) Preparation stage



(b) Verification stage

Fig. 2. Illustration of blockchain based data integrity verification framework. (a) shows the workflow of user uploading data to P2P cloud storage servers. (b) shows the workflow of verifying data integrity in P2P cloud storage servers combined with blockchain.

1) Communicational Cost: Since the public tree needs to be passed from the user to the cloud servers, the size of the public tree is proportional to the communication cost. Assuming that the output degree of each node of the tree is m, and the total number of leaf nodes, namely the total number of shards, is n, then the total number of nodes of the public tree is:

$$\begin{aligned} sum(m) &= m^0 + m^1 + m^2 + \cdots + m^{log_m n} \\ &= m^0 + m^1 + m^2 + \cdots + n. \end{aligned} \tag{1}$$

To explore the relationship between m and $sum(m)$, we assume that the number of branches of the two types of Merkle trees are m_1, m_2 respectively, which is satisfied with $m_1 = (m_2)^2$.

When $m_1 = (m_2)^2$ and n is fixed, the statement that $sum(m_1) < sum(m_2)$ is true. So we can get the conclusion that when m (the branching of the Merkle tree) increases, the size of public tree decreases, the communication cost decreases.

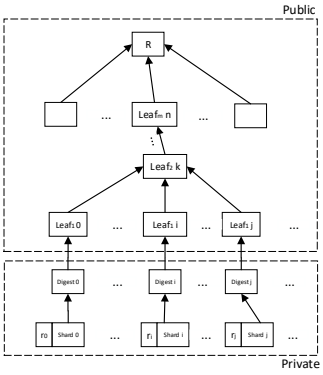


Fig. 3. The structure of Multi-Branch Tree. There are two parts of this Merkle Trees, public and private. The arrows in this figure represent performing a hash function. The bottom layer of the private part consists of shards $shard_i$ and random challenging numbers r_i. The second layer of the private part is the hash result $Digest_i$ of the bottom layer. $Leaf_m n$ in the public part of the Merkle tree is the nth leaf node of the m layer. The top of the tree is the hash root of this Merkle Trees, denoted as R. The public part is uploaded to cloud storage servers, and the private part is stored by the client.

2) Computational Cost: We measure the computational cost by calculating the delay in completing the computation, because the computational cost is proportional to the computational latency.

(i) Verify Shards - The latency of verifying shards = calculation times $\times$ time cost of each calculation, and the calculation times of each shard is $F1(m) = log_m n$. When n is fixed, $F1(m)$ decreases as m increases. So the latency of verifying shards decreases as m increases.

(ii) Generate Merkle Trees - The latency of generating Merkle trees is proportional to the size of public tree. From previous we know that the total number of nodes of the public tree decreases as m increases. So the latency of generating Merkle trees decreases as m increases.

(iii) Generate Auxiliary Path - The latency of generating the auxiliary path is proportional to the size of the auxiliary path. The size of the auxiliary path is $F2(m)$.

$$\begin{aligned} F2(m) &= (m-1)log_m n = (m-1)\frac{lnn}{lnm} \\ &= lnn(\frac{m}{lnm} - \frac{1}{lnm})(m \geq 2). \end{aligned} \tag{2}$$

$F2(m)$ increases as m increases. But each element in the auxiliary path is a hash string, so that it takes up little memory overhead. It is also very quick to get the auxiliary path through the Merkle trees (as shown in the experiment), the cost of calculation is small. Therefore, the additional communication and computational costs of the auxiliary path caused by the multi-branch Merkle trees structure are negligible.

C. Sampling Verification

Due to the limitation of real-time requirement, there is no need to verify all data shards to confirm the data integrity. In these cases, we need to choose a part of shards to verify.

Selecting a portion of the shards from the overall data shards for validation is regarded as a sampling problem. In our scenario, we choose random sampling strategy because the difference between each data shard is very small. Then, we adopt repeated sampling (sampling with replacement) to ensure that the probability of each shard to be chosen is the same, which guarantees the fairness.

1) Sampling Strategy: We adopt two sampling strategies: simple random sampling and stratified sampling. Firstly, simple random sampling is to generate sampling data through random functions. Secondly, stratified sampling is to stratify the overall data according to certain characteristics and then perform simple random sampling in each layer.

(i) Simple Random Sampling - The random function is used to randomly select shards for verification. Simple random sampling was first adopted at the beginning of the operation of the system, because we knew little about the service providers at this time.

(ii) Stratified Sampling - After a period of simple random sampling, we would get the service providers' ability to guarantee data integrity. Higher ability means a higher probability of preserving complete data. According to the grade of service providers' ability, we can divide the providers into several layers. Then perform random sampling over each layer. Assuming that providers are divided into three layers, denoted as $R1, R2, R3$ respectively, the sample sizes for each layer are $N1, N2, N3$ respectively, and the sample size of sampling is N. We need to ensure $N = N1 + N2 + N3$. The sample size for each layer decreases proportionately to the grade of the service providers' ability to guarantee data integrity.

These two sampling strategies are combined to perform sampling. At the beginning of the system, Simple Random Sampling will be performed to get these service providers' credit rating. Higher credit rating means the ability to ensure higher data integrity. Then, according to these credit rating, we can divide the providers into several layers, and perform Stratified Sampling. After a while, we will rerun the Simple Random Sampling to update the providers at each layer, then perform Stratified Sampling continue.

2) Sample Size: The total number of validated shards is called sample size. Sample size of sampling will affect the cost and precision of verification. For the verification cost, the larger the sample size, the more pieces of shards need to be verified, and the higher the verification cost. That is the verification cost is positively correlated with the sample size. For the verification precision, the larger the sample size, the more representative it is of the overall data, and the higher the verification precision. It is means that the verification precision is also positively correlated with the sample size.

(i) Verification Cost - A simple linear function can be used to express the relationship between sample size N and verification cost C:

$$C = c_0 + c_1 N, \tag{3}$$

where $c_0 > 0, c_1 > 0$. c_0 represents the basic cost, and c_1 represents the influence degree of sample size. The values of c_0 and c_1 are not of direct interest, but used to establish a linear relationship between C and N.

(ii) Verification Precision - Suppose the total number of data shards is n, where there are f invalid (lost or tampered) shards, and the sample size of sampling is N. The variable V is used to represent the number of invalid shards detected in the sampled data, then the probability P_V represents at least one invalid shard has been detected, which is:

$$\begin{aligned} P_V &= P\{V >= 1\} = 1 - P\{V = 0\} \\ &= 1 - (\frac{n-f}{n})^N. \end{aligned} \tag{4}$$

Since we used repeated sampling, the probability that a shard selected randomly was valid is $\frac{n-f}{n}$. So when the sample size is N, the probability that no invalid shard is detected is $P\{V=0\} = \underbrace{\frac{n-f}{n} * \frac{n-f}{n} * \cdots * \frac{n-f}{n}}_{\mathrm{N}}$.

(iii) Overall Consideration - The ideal situation is to spend as little verification cost as possible and obtain as high verification precision as possible. However, the verification cost and the verification precision are in contradictory relationship. What we need to do is finding an optimal sample size to tradeoff the contradiction between the verification cost and the verification precision. Therefore, we propose a Loss Function $L(N)$ to comprehensively consider the impact of sample size on verification cost and verification precision, which is:

$$\begin{aligned} L(N) &= C + \lambda \frac{1}{P_V} \\ &= c_0 + c_1 N + \lambda(\frac{1}{1 - (\frac{n-f}{n})^N}), \end{aligned} \tag{5}$$

where $N \in (0, n], c_1 > 0, c_0 > 0$. The relationship among loss, cost and precision reflected by the loss function should conform to the actual situation. That is, the higher the cost, the greater the loss, i.e. the loss is proportional to the cost. The higher the precision, the smaller the loss, i.e. loss and precision are inversely proportional. Therefore, we adopted Equation. 5 to express the relationship among loss, cost and precision in a simplified way. λ balances the importance between verification cost and verification precision. In practice, if we have a different emphasis on validation precision and validation overhead, we can change the value of λ. In order to simplify the analysis, we set $\lambda = 1$ in our paper. As c_0, c_1, n, f can be obtained as constants, $L(N)$ can be regarded as a function of variable N. Our goal is to find an optimal N to make $L(N)$ minimum, we could get the following theorem to calculate the optimal N.

Theorem 1: When $N \in (0, n]$, there exists the optimal $N = N2$ to make $L(N)$ minimum, where $N2 = log_a \frac{(2c_1 - lna) - \sqrt{lna^2 - 4c_1 lna}}{2c_1}$.

3) Order of Verification: After getting the samples, appropriate strategies can be used to determine the order, in which the samples are verified. We can abstract this issue as follows. Given the sample size N, assuming there exists an invalid shard, denoted as i, to discover invalid shard i, which kind of validation strategies should be adopted so we can verify the least amount of shards, namely the verification cost is minimal.

Here we apply several basic algorithms, which are sequential verification, block verification, exponential verification, binary verification and fibonacci verification, to our new scenario. Due to the space limitation, we do not elaborate on the implementation steps of each algorithms in this paper.

IV. Experiments

In this section, we firstly describe the implementation of a prototype system of the proposed framework. Then, we conduct some experiments about the structure of Merkle trees and sampling verification, and analyze their performance through the experimental results.To simplify the description, symbols used in this paper are shown in Table I.

TABLE I
NOTATIONS IN THIS PAPER

Notation	Description
CSS	Cloud Storage Servers
TPA	Third Party Auditor
BC	Blockchain
n	the total number of shards
m	the branch number of the Merkle trees
BBT	Binary-Branching Merkle trees
FBT	Four-Branching Merkle trees
EBT	Eight-Branching Merkle trees

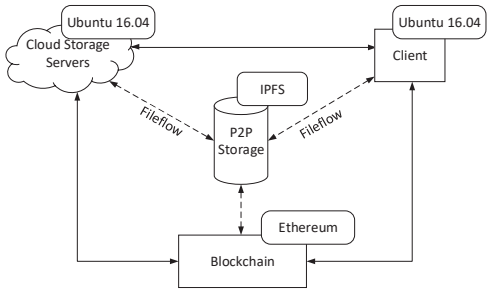


Fig. 4. The implementation of data integrity verification framework. In this fig, solid lines represent logical relationships and dotted lines represent actual interactions.

A. Framework Implementation

Fig. 4 shows the structure of the data integrity verification framework. The blockchain system is implemented by Ethereum as it is the most mature blockchain platform that supports smart contract. Clients and CSS are emulated through Ubuntu 16.04. The P2P cloud storage servers is implemented by IPFS, standing for Inter-Planetary File system, which is an attempt to share files in an HTTP manner. This paper focuses on how to select a part of data shards for data integrity verification and guarantee high performance. As for the impact of blockchain on the throughput of the system, this is the other issue deserves further study, which is not discussed in the experimental part of this paper. The implementation steps of this framework is illustrated in Table II.

TABLE II
IMPLEMENTATION STEPS OF DATA INTEGRITY VERIFICATION FRAMEWORK

Step	Entities	Operation
	Preparation Stage	
1	Client	data $\rightarrow$ Merkle trees
2	Client $\longleftrightarrow$ CSS	agreement on Merkle trees
3	Client $\rightarrow$ IPFS	upload root1
4	IPFS $\rightarrow$ Client	return ipfs-address-root1
5	Client $\rightarrow$ BC	upload ipfs-address-root1
6	Client $\rightarrow$ IPFS	upload data, Merkle trees
7	IPFS $\rightarrow$ Client	return ipfs address of each shard
	Verification Stage	
1	Client $\rightarrow$ IPFS	send ipfs address of shard i
2	IPFS	compute new root2
3	IPFS $\rightarrow$ BC	send ipfs-address-root2
4	BC	compare(ipfs-address-root1, ipfs-address-root2)
5	BC $\rightarrow$ Client	send verification result

B. The Structure of Merkle Trees

we conduct the experiments under three different Merkle trees structures, which are Binary Branching tree (BBT), Four-Branching tree (FBT), Eight-Branching tree (EBT). Then assuming the total number of shards is from 16 to 16384 (16, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384). The performance of the three Merkle trees structures was compared in terms of the time cost of verifying shards, the time cost of building Merkle trees, and the time cost of generating auxiliary path.

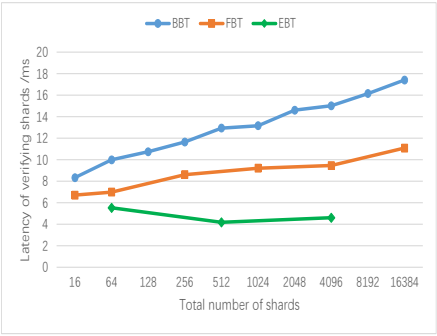


Fig. 5. The relationship between the verification latency and the total number of shards.

Since n and m need to satisfy the relationship $n = m^k (k = 0, 1, 2, ...,)$ to form a full tree, the n that different m can take is not completely the same. In the figure, the total number of shards that BBT, FBT, EBT can obtain is not exactly the same. Fig. 5 shows that EBT performs best and BBT performs worst in terms of the time cost of verifying shards. From Fig. 6, we can see that FBT and EBT are significantly better than BBT. This is mainly reflected in the following two aspects : (1) the latency of generating the Merkle Tree of FBT and EBT is always smaller than BBT and (2) as shards grow, the FBT's and EBT's latency growth rate are significantly smaller

than BBT's. As the computation cost is positively related to the computation delay, FBT and EBT are better than BBT in computation overhead. In terms of the auxiliary path, the previous theoretical analysis has concluded that the auxiliary path size will increase with the number of branches. However, each additional element of the auxiliary path is a hashed string, the increased storage space is small. At the same time, it can be seen through Fig. 7 that the time delay for generating auxiliary paths does not significantly increase with branches. Therefore, the additional communication and computation costs of the auxiliary path caused by the multiple branching tree structure are negligible. In summary, the performance of FBT and EBT are better than BBT.

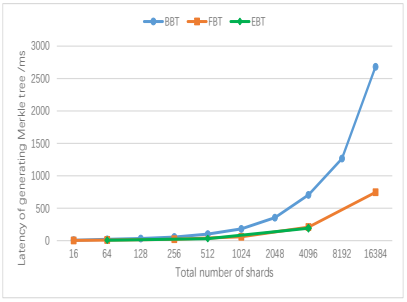


Fig. 6. The relationship between the latency of generating Merkle Trees and the total number of shards.

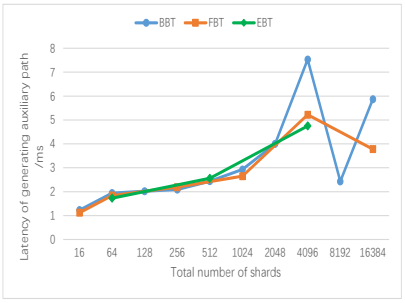


Fig. 7. The relationship between the latency of generating auxiliary path latency and the total number of shards.

C. Sample Size

In Section III, we have introduced the Loss Function $L(N)$, which involves $c_0, c_1, n, f,\ N$, to decide the suitable sample size. To simplify the calculation, we set $c_0 = 0$, then conduct two sets of experiments. In the first set of experiments, we assume the total number of shards $n = 10000$, $c1 = 0.053$ and take four different value of f, that is $f/n = 0.001, f/n = 0.002, f/n = 0.01, f/n = 0.05$. In the second set of experiments, we assume the total number of shards $n = 10000$, $f/n = 0.002$ and take four different value of $c1$, that is $c1 = 0.7, c1 = 0.4, c1 = 0.1, c1 = 0.01$. Then describe the general direction of $L(N)$ as N changes.

Fig. 8 and Fig. 9 show that as the sample size N increases, the value of the Loss Function $L(N)$ decreases from a value firstly, after reaching a minimum value, it starts to increase continuously. Thus, there exits a most approprite value of N leading to the minimum value of $L(N)$. From Fig. 8, we can see that when f/n is larger, the minimum value of $L(N)$ is closer to the Y-axis and X-axis. That means the more shards fail, the smaller the optimal sample size. As f/n increases, the minimum value of the Loss Function $L(N)$ decreases. This means that the overall validation performance of this system increases as the number of failed shards increases. From Fig. 9, we can see that when $c1$ increases, the optimal sample size decreases while the minimum value of the Loss Function increases. It means that when the weight of verification overhead increases, the optimal sample size decreases, while the overall validation performance of this system decreases.

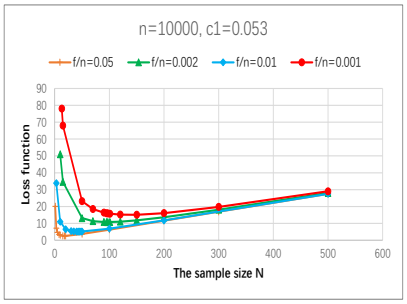


Fig. 8. The relationship between Loss Function $L(N)$ and the sample size N when f/n changes.

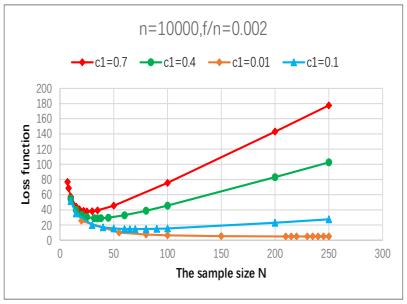
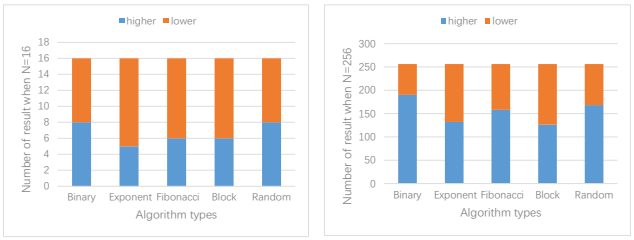


Fig. 9. The relationship between Loss Function $L(N)$ and the sample size N when $c1$ changes.

D. Order of Verification

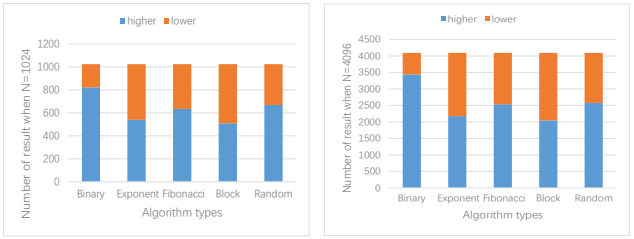
In order to compare the verification performance of different algorithms, we set the sample size from 16 to 4096 (16, 256, 1024, 4096), and use sequential validations' cost as a benchmark. We make the sample data shard one invalid at a time, then count the number of verification costs higher or lower than the baseline at the failure location using different algorithms.

Fig. 10-11 show that the proportion of verification cost higher than the benchmark increases with the increase of sample size N. From Fig. 10a we know that when N is small, almost each algorithm works better than the benchmark. Comparing Fig. 10b, Fig. 11a and Fig. 11b, we can see that with the increases of sample size N, the performance of Binary verification is getting worse and worse, while the performance



(a) N=16 (b) N=256

Fig. 10. Statistics on validation overhead when N=16 and N=256.



(a) N=1024 (b) N=4096

Fig. 11. Statistics on validation overhead when N=1024 and N=4096.

of other algorithms remain stable. Fibonacci verification and Random verification are work better than the baseline when $N = 16$. Exponent verification and Block verification always work better than the baseline no matter how N increase. Thus, we can get the conclusion that although the verification costs of these algorithms are increasing, Exponential verification and Block verification work better than others.

V. Conclusion and Future Work

In this paper, we have proposed a general verification framework in P2P cloud storage. It solves the problem of untrustworthy in traditional verification mechanism by utilizing blockchain. As the data integrity verification method in blockchain, we also analyze the verification performance under various Merkle trees structures. To improve the verification performance while also to keep a high verification precision, we further design rational sampling strategies and calculating the optimal sample size. Finally, we demonstrate the feasibility of the proposed framework by implementing a prototype system and validating our analysis of Merkle trees and sampling strategy through extensive experiments. In the future work, we will further improve the presentation of equations, elaborate our experiments and evaluate the performance of the system with real Blockchain systems.

VI. Acknowledgement

This work is supported by the National Key Research and Development Program of China under grants 2016QY01W0202 and 2016YFB0800402, National Natural Science Foundation of China under grants 61572221, U1401258, 61433006 and 61502185, Major Projects of the National Social Science Foundation under grant 16ZDA092, Science and Technology Support Program of Hubei Province under grant 2015AAA013, Science and Technology Program of Guangdong Province under grant 2014B010111007 and Guangxi High level innovation Team in Higher Education InstitutionsłInnovation Team of ASEAN Digital Cloud Big Data Security and Mining Technology.

References

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *ACM Conference on Computer and Communications Security*, pages 598–609, 2007.

[2] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, pages 1–10, 2008.

[3] G. Chairscott, L. Michael, P. Chairpeterson, and Larry. Proceedings of the nineteenth acm symposium on operating systems principles. 2003.

[4] B. Chen and R. Curtmola. Robust dynamic remote data checking for public clouds. In *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*, pages 1043–1045. ACM, 2012.

[5] R. Curtmola, O. Khan, R. Burns, and G. Ateniese. Mr-pdp: Multiple-replica provable data possession. In *The 28th International Conference on Distributed Computing Systems (ICDCS2008)*, pages 411–420. IEEE, 2008.

[6] Y. Deswarte, J.-J. Quisquater, and A. Saïdane. Remote integrity checking. In *Integrity and Internal Control in Information Systems VI*, pages 1–11. Springer, 2004.

[7] H. He, R. Li, X. Dong, and Z. Zhang. Secure, efficient and fine-grained data access control mechanism for p2p storage cloud. *IEEE Transactions on Cloud Computing*, 2(4):471–484, 2014.

[8] W. U. Ji-Yi, F. U. Jian-Qing, L. D. Ping, and Q. Xie. Study on the p2p cloud storage system. *Acta Electronica Sinica*, 35(5):1100–1107, 2011.

[9] A. Juels and B. S. Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 584–597. ACM, 2007.

[10] 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 24th IEEE International Conference on Web Services (ICWS 2017)*, pages 468–475. IEEE, 2017.

[11] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[12] F. Sebé, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater. Efficient remote data possession checking in critical information infrastructures. *IEEE Transactions on Knowledge and Data Engineering*, 20(8):1034–1038, 2008.

[13] H. Shacham and B. Waters. Compact proofs of retrievability. *Journal of Cryptology*, 26(3):442–483, 2013.

[14] Y. Teing, A. Dehghantanha, K. R. Choo, and L. T. Yang. Forensic investigation of P2P cloud storage services and backbone for iot networks: Bittorrent sync as a case study. *Computers & Electrical Engineering*, 58:350–363, 2017.

[15] D. Vorick and L. Champine. Sia: simple decentralized storage. http://www.sia.tech/, 2014.

[16] C. Wang, Q. Wang, K. Ren, and W. Lou. Ensuring data storage security in cloud computing. In *17th International Workshop on Quality of Service (IWQoS 2009), Charleston, South Carolina, USA*, pages 1–9, 2009.

[17] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS 2009), Saint-Malo, France. Proceedings*, pages 355–370, 2009.

[18] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin. Storj a peer-to-peer cloud storage network. https://storj.io/, 2014.