Journal of Parallel and Distributed Computing 146 (2020) 1–14

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Blockchain-based verification framework for data integrity in edge-cloud storage

Dongdong Yue [a], Ruixuan Li [a,*], Yan Zhang [b], Wenlong Tian [c], Yongfeng Huang [d]

[a] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
[b] School of Computing, Engineering and Mathematics, Western Sydney University, Sydney, Australia
[c] School of Computer Science and Technology, University of South China, Hengyang, China
[d] Department Of Electronic Engineering, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 1 December 2019
Received in revised form 8 April 2020
Accepted 6 June 2020
Available online 22 June 2020

Keywords:
Blockchain
Edge-cloud storage
Data integrity verification
Merkle trees
Sampling

ABSTRACT

With the popularity of the Internet of Things (IoT), data integrity verification in the edge cloud storage attracts attentions from many researchers. Due to the over dependence of the Third Party Auditor (TPA) and the dynamical nature of the IoT data, the traditional data integrity verification framework for cloud storage can hardly work. To satisfy the characteristics of the IoT and avoid the over dependence of the TPA, we propose a blockchain-based framework without TPA for data integrity verification in a decentralized edge-cloud storage (ECS) scenario in this paper. In our framework, we employ the Merkle tree with random challenging numbers for data integrity verification and analyze different Merkle tree structures to optimize the system performance. To solve the problem of limited resources and high real-time requirements, we further propose sampling verification and develop rational sampling strategies to make sampling verification more effective. The overhead and precision of the verification in ECS are studied by an optimal sample size strategy. Finally, a prototype system is implemented based on our framework. We conduct a series of experiments to evaluate the effectiveness of the proposed schemes. The experimental results show that our schemes can effectively improve the performance of data integrity verification.

© 2020 Elsevier Inc. All rights reserved.

1. Introduction

Due to the rapid growth of the Internet of Things (IoT), how to transfer the data from resource-constrained IoT devices to the remote cloud becomes an urgent problem [15]. To solve this problem, the centralized cloud-assisted Internet of Things (CoT) model is changed into the decentralized edge-cloud storage (ECS) model [22]. However, both edge nodes and cloud are not completely secure. In other words, the data owners will lose control of these data when they are collected and uploaded to the cloud. Meanwhile, terminal devices in the IoT have limited resources and computing power, while the traditional methods of data integrity do not consider the limitations in the IoT scenarios. Therefore, how to verify data integrity to conquer these limitations under the decentralized ECS model is critical.

The data storage and integrity verification 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) [20]. The client stores his 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.

In this verification mechanism, the TPA is introduced to carry out the verification between the client and CSS. However, the TPA is composed of one or several organizations and is to some extent a centralized organization. If CSS colludes with TPA maliciously, the result of data integrity verification may not be credible. If the TPA fails, the overall verification work cannot be carried out. In addition, the emergence of TPA for data process increasing the risk of threats from hackers [1]. Therefore, the traditional verification mechanism exists a series of security issues.

Moreover, with the rapid growth of the number of IoT devices, it is difficult to transmit a huge amount of data generated by the IoT devices to the cloud. It is because the location and power of the IoT devices limit the data travel efficiency and the data computation pressure derived from the cloud processing capability. To mitigate the pressure of centralized cloud servers and reduce the cost of data communications, the centralized CoT model is changing to the decentralized ECS model. ECS model complements the traditional CoT mode in terms of high scalability, low

* Corresponding author.
E-mail addresses: ydd@hust.edu.cn (D. Yue), rxli@hust.edu.cn (R. Li), Yan.Zhang@westernsydney.edu.au (Y. Zhang), wenlongtian@usc.edu.cn (W. Tian), yfhuang@mail.tsinghua.edu.cn (Y. Huang).

https://doi.org/10.1016/j.jpdc.2020.06.007
0743-7315/© 2020 Elsevier Inc. All rights reserved.

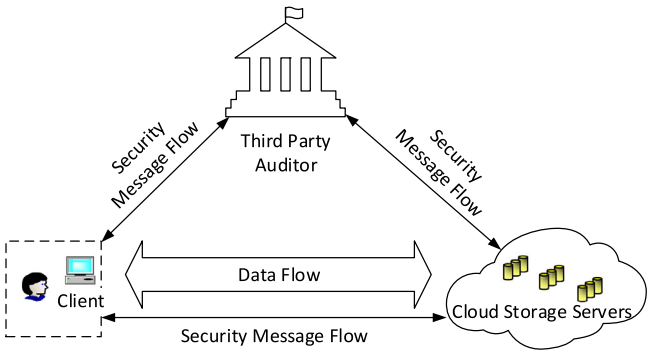

Fig. 1. Data storage and integrity verification on the cloud under traditional architecture.

latency, location awareness, and instant local client computing power. In the ECS model, both cloud and local edge devices can provide data storage and computing services.

Blockchain is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way [13]. The distributed nature of blockchain makes it an inevitable trend to apply blockchain to data integrity verification in the ECS model. The TPA was removed and replaced by introducing the blockchain technology, which makes the data integrity verification work more open, transparent and auditable. As the blockchain is a decentralized structure, multiple nodes jointly maintain the operation of the blockchain. Thus, there is no single point of failure. At the same time, the records of blockchain are determined by all the nodes in the blockchain, which makes the records that exist on the blockchain be more secure and credible. Therefore, applying blockchain technology to data integrity verification in the ECS model is feasible.

In most of the edge-cloud storage scenarios of IoT systems, the resources of the edge nodes are limited, and the real-time requirements of systems are very high. In these cases, it is difficult to verify the integrity of the whole data in the IoT systems. The methods of verifying data integrity by using a segment of the original data in place of verifying the whole data have been proposed in Sia [19]. Sia is a decentralized cloud storage platform that incorporates blockchain for data integrity verification. The storage space in Sia is idle disk space shared by hosts. Hosts prove their storage integrity by providing a segment of the original data and a list of hashes from the data Merkle tree. However, Sia does not provide how to select segments of the original data to verify, nor consider the size of the selected data. Therefore, how to design rational data integrity verification schemes for ECS is still an unsolved problem.

In this paper, we firstly propose a general data integrity verification framework by utilizing the blockchain technology into ECS to solve above issues. Then, we employ the Merkle tree for data verification under our proposed framework and analyze the performance of the Merkle tree with different structures. We further introduce a sampling strategy to select data shards for validation. To the best of our knowledge, we are the first to employ blockchain to solve data integrity verification in the ECS model. The main contributions of this paper are summarized as follows.

- We propose a general blockchain-based data integrity verification framework for decentralized edge-cloud storage. This framework mainly focuses on the problem of incredibility in traditional verification mechanisms and is more suitable for the ECS model.
- We present a sampling strategy to solve the problem of limited resources and high real-time requirements. Based on the sampling, we propose an optimum sample size to find a tradeoff between verification overhead and precision.
- We implement a prototype system based on the proposed framework and conduct extensive experiments to evaluate the performance of the proposed framework in the system. Experimental results demonstrate the feasibility of the proposed framework and validate our theoretical analysis.

The rest of this paper is organized as follows. In Section 2, we discuss the related work from three aspects. In Section 3, we introduce our system framework for data integrity verification. In Section 4, we present the verification mechanisms. The design of the system and the implementation of our framework are introduced in Section 5. Experimental studies are presented in Section 6, and the performance and security analysis are discussed in Section 7. Finally, Section 8 concludes the paper with some remarks.

2. Related work

In this section, we briefly outline data integrity verification techniques in traditional cloud storage, and introduce the related research work on edge storage and blockchain.

2.1. Edge storage and edge computing

With the development of IoT, billions of sensors are connected to the Internet. In this situation, traditional cloud computing models are not fully suitable because sensors generate too much data which may cause server network congestion. As a result, edge computing is proposed to solve this problem. However, edge computing exists problems of restricted computation, limited storage, and unstable network. Therefore, Xing et al. [22] propose a distributed multi-level storage (DMLS) model with a multi-factor least frequently used (mLFU) algorithm to solve the problem.

In the scenario of the IoT, dealers need to provide real-time information and feedback to end-users based on a large amount of data generated by IoT devices. Therefore, how to effectively extract useful features from these large amounts of heterogeneous data is a problem worth studying. Although cloud computing and edge computing have made parallel progress in solving some problems in data analysis, they have their own advantages and limitations. To solve this problem, Sharma et al. [18] propose a new framework for collaborative processing between edge and cloud computing/processing platforms by integrating their strengths.

Data analysis based on the data generated by IoT devices to perform near-real-time decision making is another problem worth studying. When performing near-real-time decisions at the edge, we need historical data to perform an accurate analysis. However, with limited edge storage capacity, it is a challenge to balance the amount of data stored with the quality of near-real-time decisions. To solve this problem, a three-layer architecture model for edge data storage management is proposed by Lujic et al. [12], which includes an adaptive algorithm that dynamically balances the high prediction accuracy with the minimum amount of data.

2.2. Data integrity verification in cloud storage systems

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. POR can not only verify the integrity of remote data, but also recover data with a certain probability when the data are damaged [10]. The basic PDP authentication method is proposed by Deswarte et al. [7]. 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 the above PDP mechanism is simple, directly downloading complete data requires a lot of resources and may lead to leakage of data privacy. To solve the problem, Sebe et al. [16] propose a block-based scheme to reduce the computational overhead. Due to the deterministic verification method, the verification result may not be completely correct. Then, Ateniese et al. [2] propose using probabilistic strategies to complete the integrity verification. They use the homomorphic properties of RSA signature mechanism, gathered evidence in a very small value, which greatly reduce the communication overhead. Subsequently, Curtmola et al. [6] implement the data integrity verification mechanism in the case of multiple copies, but it does not support the dynamic data operation. Ateniese et al. [3] consider the dynamic data operation firstly. They present a simple modified mechanism of the PDP based on their previous work [2], making it support dynamic data manipulation.

Although the PDP authentication mechanism can efficiently verify the integrity of data, it cannot recover invalid data as POR. Shacham et al. [17] use the BLS short message signature mechanism to construct homomorphic verification tags, which can reduce the communication overhead for verification. However, it is difficult to implement. Wang et al. [21] propose using the linear features of the error correction code to support partial dynamic operations, but it could not support the dynamic insertion of data. Chen et al. [5] optimize Wang's mechanism and uses the Reed-Solomon erasure code technique to recover the failed data, which can improve the recovery efficiency, but increases the computational cost.

In general, existing integrity verification schemes in traditional cloud storage system can hardly be applied into ECS scenario. It is because the data is distributedly stored in the edge nodes and cloud, which is different from the centralization property under traditional cloud storage scenario. Thus, it is worth to studying the data integrity verification in ECS model as the ECS becomes an important infrastructure of IoT.

2.3. 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. Gaetani et al. [8] devise a two-layer blockchain to solve the data integrity problem in cloud computing environment. More specifically, the first-layer aims at quickly storing evidence of every operation carried out on a distributed database. Thus, the first-layer adopts a lightweight consensus protocol to assure low latency and high throughput. The second-layer blockchain stores evidence of the database operations logged by the first-layer. To provide strong data integrity guarantees, the second-layer adopts prove-of-work (PoW) as its' consensus protocol. However, their paper focuses on how to prevent tampering of data recorded on the blockchain and does not cover how to use the blockchain to complete the verification of data integrity in cloud storage.

Although Sia [19] involved the Merkle tree in the smart contract in 2014 to complete data integrity verification in peer-to-peer (P2P) storage. However, there is no more detailed description of the data integrity verification scheme in their paper. At the same time, their work does not take into account the problem of verification cost and accuracy. Our paper focuses on how to balance the verification cost and accuracy and then proposes a sampling verification scheme to achieve the best verification performance.

Liu et al. [11] propose a blockchain-based framework for data integrity service. Without relying on TPA in this framework, data owners and data consumers can be provided with more reliable data integrity verification. In order to ensure the security of the whole data set when data owner application (DOA) shares data with the data consumer application (DCA), Liu et al. adopt sampling validation when DCA conducts data integrity verification. However, they did not consider that when DOA performs data integrity verification, CSS directly returns complete data blocks, which will also cause data set security problems of DOA if there is an attack. We believe that both DCA and DOA should adopt sampling verification to protect the security of the whole data set when they propose data integrity verification. Therefore, our paper focuses on how to design an appropriate sampling verification scheme to optimize the data integrity verification performance in edge-cloud storage scenario.

Blockchain technology is not only applied in data integrity verification but also in other scenarios. Nosouhi et al. [14] utilize the unique features of the blockchain technology to design a decentralized scheme for location proof generation and verification. Their work addresses the issue that dishonest users may submit fake location information to illegally access a service or obtain a benefit in location-sensitive applications. In the cloud data sharing scenario, a user will generate a new signature for the modified file after modification. If two or more users modify the same file at the same time, a signature conflict will occur. To addresses this issue, Huang et al. [9] propose a new mechanism for data storing based on blockchain to realize signature uniqueness.

3. System framework

3.1. System structure

The structure of the ECS model in our framework is shown as Fig. 2. It is a four-layer network architecture. The top layer is a remote cloud layer, which consists of cloud service providers. Cloud is used to store the main information in IoT, such as users' personal information, transaction information and so on. Since this type of information does not need to provide a high real-time service and is not often changed, it can be stored on the cloud. Moreover, this type of information will increase linearly with the accumulation of time and the increase in the number of users. Therefore, the amount of these data will be so large that they are only suitable to store on the cloud.

The second-layer is an edge network layer, which consists of several base stations. It is used to provide near-field storage services for IoT terminal devices. The communications between the first layer and the second layer are based on the route. The third-layer is the IoT terminal devices layer, which consists of a variety of IoT devices, such as mobile phones, personal computers, household devices and so on. IoT devices will communicate with their nearest base stations to get low latency of data storage service. Some regional information, such as regional news, is sent by the cloud to the regional base station, and then sent to the nearby IoT terminal devices by the base station. Information sharing can also be uploaded by one terminal device to the base station, and then shared by the base station to another terminal.

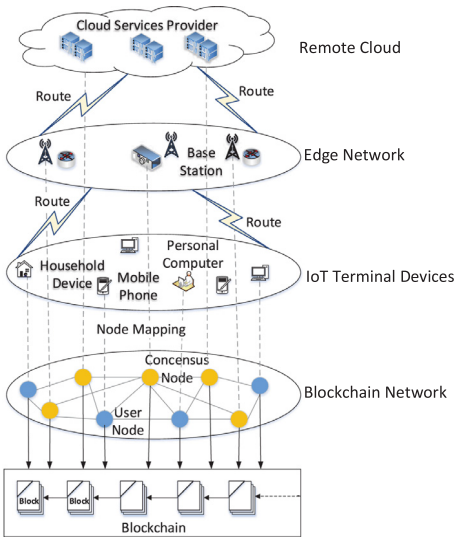

Fig. 2. The structure of edge-cloud storage model for IoT in our framework. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In addition, data can also be transferred directly between two IoT terminal devices, as long as the distance between them meets the requirements. This communication scheme is similar to Peer-to-Peer (P2P) model, a large scale distributed network, where every peer is capable of being both a client and a server in this model.

The fourth-layer is the blockchain (BC) layer. All nodes in our system, including cloud, edge nodes, and IoT device nodes, will join the blockchain network and participate in the maintenance of blockchain. Nevertheless, different nodes play distinct roles in the blockchain. Cloud and edge nodes can be the consensus nodes of the blockchain since the consensus of blockchain requires the participating nodes with strong computing and storage capabilities. They participate in the consensus calculation of blockchain so that blockchain maintains consistency in the whole network. These consensus nodes are highlighted in yellow in the blockchain network layer of Fig. 2. IoT terminal devices, which has limited computing and storage resources, can only be the user nodes of BC. They can send a transaction to the blockchain to verify data integrity and pay to the blockchain. These user nodes are highlighted in blue in the blockchain network layer of Fig. 2. Finally, those legally executed transactions are recorded on the blockchain.

There are two types of roles associated with data in the IoT terminal devices layer: data owner (DA) and data consumer (DC). The IoT devices that produce the data are data owners, and others are data consumers. The data owners will upload data to the edge network for storage and sharing. The data consumers do not generate data by themselves but consume data shared by DA, such as reading and forwarding. The requirements for data integrity verification come from DA and DC. DA needs to know whether the data uploaded to the edge network or cloud are still intact, so the integrity of the data needs to be verified. DC needs to know whether the data it consumes are consistent with the data shared by DA, so it also needs to verify the integrity of the data. For simplicity, we will unify DA and DC as clients in the following description.

3.2. Verification model

In our schema, we present a two-stage framework for blockchain-based data integrity verification in the ECS model. The first stage is the preparation stage, and the second stage is the verification stage. To facilitate understanding of the interaction process, we simplify our four-layer network structure to a three-part architecture. As shown in Figs. 3 and 4, there are three entities in our simplified three-part architecture: ECS, Clients, and BC. The ECS part is synthesized from the remote cloud and edge network. Clients are the IoT terminal devices layer in Fig. 2, which include DA and DC. Since clients can upload their own data and download data from ECS, data integrity verification requests are mainly from clients. Blockchain is introduced for data integrity verification between clients and ECS.

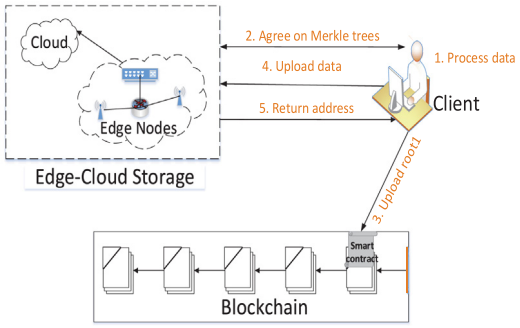

Fig. 3. The workflow of client uploading data to edge-cloud storage.

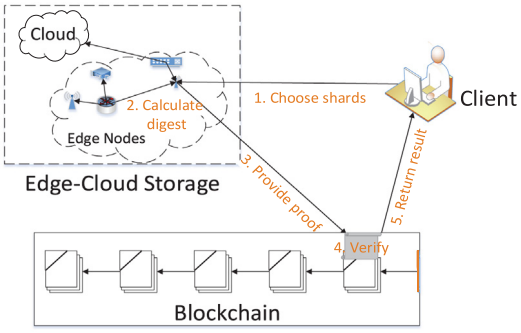

Fig. 4. The workflow of verifying data integrity in edge-cloud storage with blockchain.

3.2.1. Preparation stage

Fig. 3 shows the preparation stage of our framework. This stage is used to process clients' data and then upload them to ECS. As shown in Fig. 3, there are five steps in the preparation stage. In the first step, the client slices its data into several shards and uses these shards to construct a hash Merkle tree. In the second step, the client and ECS nodes agree on the hash Merkle trees. In the third step, the client stores the root of this hash tree denoted as *root*1 on the blockchain. In the fourth step, the client uploads its data and public Merkle trees to ECS. In the fifth step, ECS returns the address in which the client's data are stored to the client.

3.2.2. Verification stage

Fig. 4 shows the verification stage of our framework. This stage is to process data integrity verification requests proposed by the clients. As shown in Fig. 4, there are also five steps in the verification phase. In the first step, the client sends a challenge number *si* to ECS node, which selects shard *i* to verify. In the second step, ECS uses a hash function to calculate a hash digest i', according to *si* and shard *i*. In the third step, ECS sends digest i' and the corresponding auxiliary information to BC. As for what the auxiliary information is, we give a specific explanation in

Section 4.1.1. In the fourth step, the smart contract on the blockchain calculates a new hash root denoted as *root*2, and compare *root*1 with *root*2. If they are equal, the data integrity is then guaranteed; otherwise, the data integrity is corrupted. In the last step, the BC returns the verification result to the client.

In this framework, clients place the root of the Merkle trees on the blockchain before uploading the data. Due to the non-tamperability property of blockchain, any client or ECS node 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, we assume that the data on the blockchain will not be damaged. Hence, data integrity verification is more reliable.

4. Verification mechanism

In this section, we firstly introduce the structure of Merkle trees, which are employed to assist data integrity verification. The performance of different structures of Merkle trees are analyzed in terms of computation and communication overhead. Secondly, we illustrate the sampling strategies for data integrity verification in detail. Then, we propose the method for calculating the best sample size of data integrity sampling verification. Finally, we suggest five strategies to determine the order in which the samples are verified.

4.1. 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 the Merkle tree is shown as Fig. 5. There are two parts in the Merkle Tree, public and private. Each arrow in this figure represents a hash function execution. The bottom layer of the private part consists of shards and random challenging numbers. Shards are obtained by slicing the clients' original data. Each shard $Shard_i$ is assigned with a random challenge number s_i. The second layer of the private part contains digests. Each $Digest_i$ is the result of hash $(shard_i + s_i)$. $Leaf_m n$ in the public part of the Merkle tree is the nth leaf node of the mth layer. The top of the tree is the hash root of this Merkle tree, denoted as R.

The public part of the Merkle tree needs to be uploaded to the ECS node to assist in validating each data shard. The data shard $shard_i$ in the private part will also be uploaded to the ECS node. As for the random challenge number s_i, it can only be sent to the ECS node when the client needs to verify the corresponding data shard. Therefore, s_i is locally saved by the client. Since it is a tree structure, we can study the Merkle tree with different branches. This section analyzes different structures of Merkle trees, then discusses the communication and computation overhead of the system under different structures.

4.1.1. Auxiliary information

In the verification stage of Section 3.2.2, we mentioned that ECS needs to send auxiliary information to BC. Here, we will explain what auxiliary information is through a concrete example. We assume the original data are split into 12 shards, and the number of branches of our Merkle tree is four. Then, we construct a hash Merkle tree as shown in Fig. 6. We use D to represent each node in the Merkle tree. Assuming that we want to verify $shard_0$, which is highlighted in yellow in Fig. 6, we will hash $shard_0$ with s_0 to get $D0$. Then, we need $D1$, $D2$, $D3$ to get $D12$. After we get $D12$, $D13$ and $D14$ are needed to get $D15$, the root of the Merkle tree. It means, if we want to get the root of the Merkle tree from $shard_0$, we need the help from $D1$, $D2$, $D3$, $D13$, and $D14$. Therefore, these red boxes in Fig. 6 are the auxiliary information of yellow boxes.

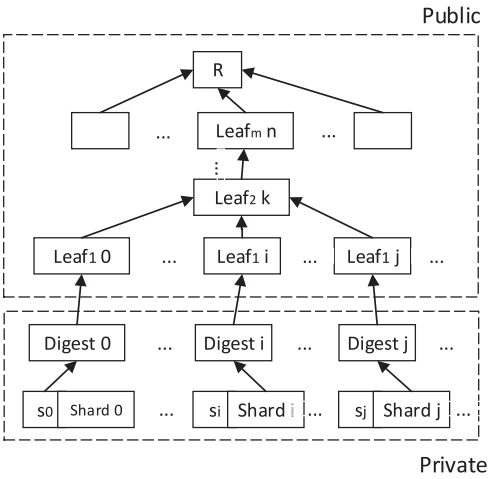

Fig. 5. The structure of Multi-Branch Tree.

4.1.2. Communication cost

Since the public tree needs to be passed from the client to the ECS node, the size of the public tree is proportional to the communication cost. Assuming that the output degree of each node of the tree is $m(m \geq 2)$, 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 numbers of branches of the two types of Merkle trees are m_1 and m_2 respectively, which are 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. Hence, we can make the conclusion that when m (the number of branches of the Merkle tree) increases, the size of public tree decreases, the communication cost decreases.

4.1.3. 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.

Verify shards. The latency of verifying shards = calculation 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. Thus, the latency of verifying shards decreases as m increases.

Generate Merkle trees. The latency of generating Merkle trees is proportional to the size of public tree. From previous description in Section 4.1.2, we know that the total number of nodes of the public tree decreases as m increases. Thus, the latency of generating Merkle trees decreases as m increases.

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 $F2(m)$.

$$F2(m) = (m-1)\log_m n = \ln n\left(\frac{m}{\ln m} - \frac{1}{\ln m}\right). \tag{2}$$

$$f2(m) = \left(\frac{m}{\ln m} - \frac{1}{\ln m}\right). \tag{3}$$

$$\begin{aligned} f2'(m) &= \frac{1}{\ln m} - \frac{m\frac{1}{m}}{(\ln m)^2} + \frac{\frac{1}{m}}{(\ln m)^2} \\ &= \frac{m\ln m + 1 - m}{(\ln m)^2 m} > 0. \end{aligned} \tag{4}$$

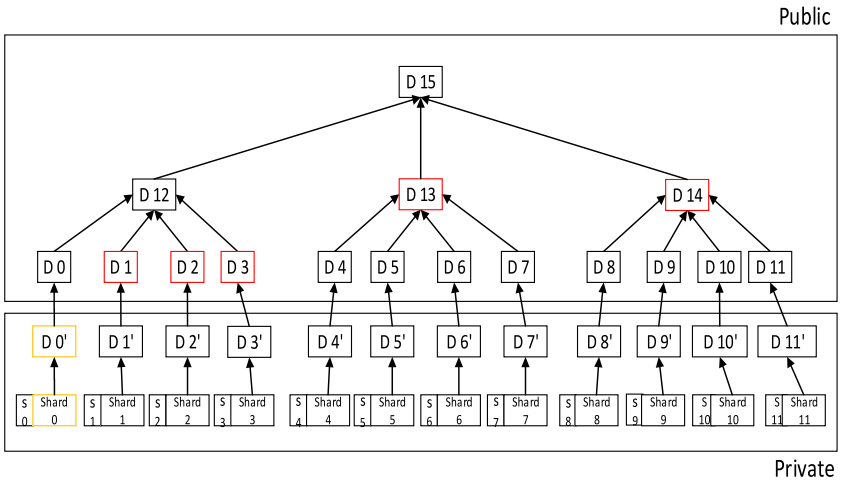

Fig. 6. The illustrate for auxiliary information in Multi-Branch Merkle Tree. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$F(2)$ is equal to $f(2)$ times a constant term, thus, they have the same monotonicity. $f2'(m)$ is the derivative of $f2(m)$. Since $f2'(m) > 0$, $F2(m)$ increases as m increases. Whereas, each element in the auxiliary path is a hash string, so that it takes up little memory overhead. It is also very fast to get the auxiliary path through the Merkle trees (as shown in Fig. 11 in Section 6.1). The cost of calculation is small. Therefore, the additional communication and computation costs of the auxiliary path caused by the multi-branch Merkle trees structure are negligible.

4.2. Sampling strategies

Due to the limitation of resources or high real-time requirements in some scenarios of IoT, it is not possible to verify all data shards to confirm the data integrity. In these cases, we need to choose a part of shards to be verified. In our paper, selecting a portion of the shards from the overall data shards for validation is regarded as a sampling problem. 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.

We adopt two random sampling strategies: simple random sampling and stratified sampling. The descriptions of these two strategies are as follows.

Simple random sampling. Simple random sampling is first adopted at the beginning of the operation of the system, because we know little about ECS nodes at this time. We set an initial credit value for each ECS node at first. At each validation, if the ECS node keeps the complete data, its credit value increases one. Otherwise, its credit value reduces one.

Stratified sampling. After a period of simple random sampling, we get the credit values of these ECS nodes. These ECS nodes are layered based on their credit values. Then, we perform random sampling over each layer. For example, assuming these ECS nodes 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 of each layer is proportional to the credit value of the corresponding layer.

These two sampling strategies are performed alternately. At the beginning of the system, simple random sampling will be performed to get credit value of each ECS node. Then, according to these credit values, we can divide ECS nodes into several layers, and perform stratified sampling. After the system running for a while, we will rerun the simple random sampling to update the ECS nodes at each layer, then continue to perform stratified sampling.

4.3. Sample size

The total number of validated shards is called sample size. The sample size will affect the cost and precision of verification. For the verification cost, the larger the sample size is, the more pieces of shards need to be verified, and the higher the verification cost is. 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 is. It means that the verification precision is also positively correlated with the sample size. The cost and precision of verification in our system are explained in detail as follows.

Verification cost. A simple linear function can be used to express the relationship between sample size N and verification cost C:

$$C = c_0 + c_1 N, \tag{5}$$

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 direct interest, but used to establish a linear relationship between C and N.

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 that has been detected, which is denoted as follows.

$$\begin{aligned} P_V &= P\{V \geq 1\} = 1 - P\{V = 0\} \\ &= 1 - \underbrace{\frac{n-f}{n} * \frac{n-f}{n} * \cdots * \frac{n-f}{n}}_{\mathrm{N}} \\ &= 1 - (\frac{n-f}{n})^N. \end{aligned} \tag{6}$$

Since we use repeated sampling, the probability that a shard selected randomly is valid is $\frac{n-f}{n}$. Thus, when the sample size is N, the probability that no invalid shard is detected is

$$P\{V = 0\} = \underbrace{\frac{n-f}{n} * \frac{n-f}{n} * \cdots * \frac{n-f}{n}}_{\mathrm{N}}.$$

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 balance 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.

The Loss Function $L(N)$ should be able to correctly represent the contradictory relationship between the verification cost and the verification precision. In addition, the relationship among the loss, the verification cost and the verification precision reflected by the loss function should conform to the actual situation. That is, the higher the cost is, the greater the loss will be, i.e. the loss is proportional to the cost. The higher the precision is, the smaller the loss will be, i.e. loss and precision are inversely proportional. Therefore, we propose two types of Loss Functions $L_1(N)$ and $L_2(N)$. They are shown in Eqs. (7) and (11).

1. **Loss Function in Addition Form.** $L_1(N)$

$$\begin{aligned} L_1(N) &= C + \lambda \frac{1}{P_V} \\ &= c_0 + c_1 N + \lambda\left(\frac{1}{1-(\frac{n-f}{n})^N}\right), \end{aligned} \tag{7}$$

where $N \in (0, n], c_1 > 0, c_0 > 0$. 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, $L_1(N)$ can be regarded as a function of variable N. Our goal is to find an optimal N to make $L_1(N)$ 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 $L_1(N)$ minimum, where $N2 = log_a \frac{(2c_1 - lna) - \sqrt{lna^2 - 4c_1 lna}}{2c_1}$.

Proof 1. To verify the theorem, we have the following proof.

$$\lim_{N\to 0} L_1(N) = +\infty, \quad \lim_{N\to n} L_1(N) = c_0 + c_1 n + 1. \tag{8}$$

Variable substitution: $a = \frac{n-f}{n}, x = a^N$, then $a \in (0, 1), x \in (0, 1)$.

$$L_1(x) = c_0 + c_1 log_a x + \frac{1}{1-x}. \tag{9}$$

The derivative of $L_1(x)$ is $L_1'(x) = \frac{c_1}{xlna} + \frac{1}{(1-x)^2}$, when $L_1'(x) = 0$, then $c_1x^2 + (lna - 2c_1)x + c_1 = 0$. The roots of this equation are

$$\begin{aligned} x_1 &= \frac{(2c_1 - lna) + \sqrt{lna^2 - 4c_1 lna}}{2c_1}, \\ x_2 &= \frac{(2c_1 - lna) - \sqrt{lna^2 - 4c_1 lna}}{2c_1}. \end{aligned} \tag{10}$$

Easy to prove that $x_1 \in (1, \infty), x_2 \in (0, 1)$, corresponding $N1 < 0, N2 > 0$, then the valid data domain is $N \in (0, N2] \cup (N2, n]$. When $N \in (0, N2]$, $L_1'(N) < 0$. When $N \in (N2, n]$, $L_1'(N) > 0$. So, $L_1(N)$ will be minimum when $N = N2$, and $N2 = log_a \frac{(2c_1 - lna) - \sqrt{lna^2 - 4c_1 lna}}{2c_1}$, therefore the best sample size is $N2$.

2. **Loss Function in Division Form.** $L_2(N)$

$$L_2(N) = \frac{C}{P_V} = \frac{c_0 + c_1 N}{1 - (\frac{n-f}{n})^N}. \tag{11}$$

The variables description for Eq. (11) is the same as Eq. (7).

Theorem 2. When $N \in (0, n]$, there exists the optimal N to make $L_2(N)$ minimum.

Proof 2. To valid the theorem, we have the following proof.

$$\lim_{N\to 0} L_2(N) = +\infty, \quad \lim_{N\to n} L_2(N) = c_0 + c_1 n. \tag{12}$$

Variable substitution: $a = \frac{n-f}{n}, x = a^N$, then $a \in (0, 1), x \in (0, 1)$.

$$L_2(x) = \frac{c_0 + c_1 log_a x}{1 - x}. \tag{13}$$

The first derivative of $L(x)$ is $L_2'(x) = \frac{\frac{c_1}{xlna}(1-x)}{(1-x)^2} + \frac{c_0 + c_1 log_a x}{(1-x)^2}$. When $x > 0$, the second derivative $L_2''(x) > 0$, which means that the first derivative $L_2'(x)$ is an increasing function.

$$\lim_{x\to 0} L_2'(x) = -\infty, \quad \lim_{x\to 1} L_2'(x) = +\infty. \tag{14}$$

There must exist x_1 that makes $L_2'(x_1) = 0$. Thus, L(N) will be minimum when $N = N1$, and $N1 = log_a x_1$. Therefore, the best sample size is $N1$.

4.4. Order of verification

After getting the samples, appropriate strategies can be used to determine the order, in which the samples are verified. We 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 that we can verify the least amount of shards, namely the verification cost is minimal.

To facilitate the description, we use an array to represent the sample shards. The index of shards is from 0 to $N - 1$, while the invalid shard's index is i, and the verification cost (the number of the shards that need to be verified before verifying the invalid shard) is m. Then, we adopt the following five verification mechanisms.

1. Sequential Verification — Start with the first shard and validate each shard forwards in order until the invalid shard i is validated.
2. Block Verification — Slice all shards in the sample equally to several blocks, then verify the kth shard in each block in order in the kth round.
3. Exponential Verification — The verification starts from the first shard (index marked as c) and the position of the next verified shard is equal to $c + 2^i$ (where $i = 0, 1, 2, 3, \ldots, log_2 N$). If the index is out of range and the invalid shard i still has not been verified, then the index verification is performed again from the foremost one of the unverified shards until the invalid shard i is found.
4. Binary Verification — Verify the middle shard of the array each round. If the shard is not the invalid shard, first enter the front subarray segmented by the current validated shard recursively, then continue performing the same validation on the back subarray recursively until the invalid shard i is verified.
5. Fibonacci Verification — First, verifying the first three shards in sequence. Starting from the fourth shard, the index of each shard to be verified are the sum of the index of the previous two shards (As the index of the first shard is 0, so starting from the second shard to execute Fibonacci). If the index is out of range and the invalid shard i has not yet been verified, then Fibonacci verification is performed again on the foremost one of the unverified shards. At this time, the previous one shard will be used for help doing Fibonacci calculations.

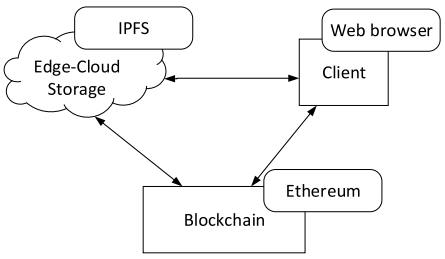

Fig. 7. The implementation of data integrity verification framework.

These above five verification mechanisms are traditional algorithms, but they are also applicable in our scenario.

5. System design and implementation

In this section, we introduce the implementation of data integrity verification framework, and the detailed design of the smart contracts and the major functions in the system.

5.1. Framework implementation

Fig. 7 shows the implementation of the data integrity verification framework. The blockchain system is implemented by a private Ethereum as Ethereum is a quite mature blockchain platform that supports smart contracts. The client is emulated through the web browser. By supporting data uploading, downloading, sharing, and viewing, the web browser can act as DA and DC. Since the implementation of a real-world edge-cloud storage integrated IoT is quite complex, we adopt the Inter-Planetary File System (IPFS) to simulate ECS. As a distributed P2P File System, IPFS will speed up the file retrieving process. The specific configuration of the experimental environment is: 8 GB RAM, 8 core processor, 500G hard disk, and the system is Ubuntu 16.04LTS. The programming language are Node.js and Solidity. The hash function used to calculate $Digest_i$ is SHA256.

This paper focuses on how to select a part of data shards for data integrity verification and guarantee high performance. We just use blockchain as a technology to assist verification and choose private Ethereum as our blockchain platform. As for the incentive mechanism and throughput, they are the research issues of blockchain itself, which deserve further study. Thus, we do not discuss them in our paper. A specific system workflow is shown in Fig. 8. The smart contract is deployed on the blockchain. We use a two-way arrow to connect the smart contract and the blockchain. A more detailed introduction to IPFS and Ethereum are as follows.

IPFS. Inter-Planetary File System (IPFS) [4] is a global, distributed P2P file system, which is an attempt to share files in an HTTP manner. According to the data communication mode in our four-layer architecture described in Section 3.2, we can adopt the P2P file system to simulate ECS.

Ethereum. The representative of blockchain 2.0 is Ethereum. Ethereum provides three types of network: public, test and private. Ethereum public network is a real global network that participators need money to deploy a smart contract on it. Ethereum test network is a global test network that participators do not need money to deploy a smart contract on it. Since Ethereum private network is a private network built by individual and does not need money to deploy a smart contract on it, it is suitable for developing a test smart contract.

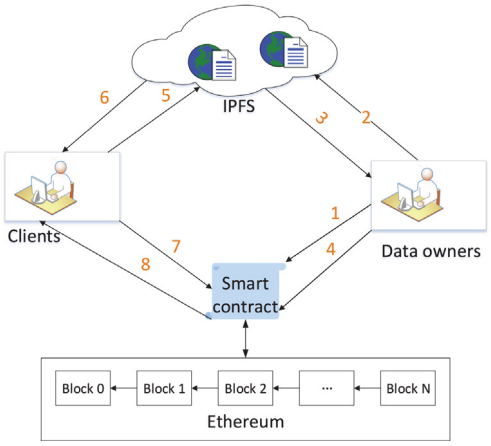

Fig. 8. The workflow of data integrity verification framework.

The preparation stage is from step 1 to step 4. In this stage, data owners need to do some preparatory works. The corresponding description of each step number in Fig. 8 is shown as follows:

step 1: Data owners deploy storage contract, compute contract, and compare contract on the Ethereum.

step 2: Data owners slice the data firstly and get multiple data shards. Then, data owners generate the Merkle tree and calculate *root*1 according to these shards. Finally, data owners upload file shards and *root*1 to the IPFS.

step 3: IPFS returns *root*1 IPFS address and data shards IPFS addresses to data owners.

step 4: Data owners store *root*1 on storage contract and store data shards IPFS addresses by themselves. If sharing, the IPFS address of the data shard will be sent to DC by DA.

The verification stage is from step 5 to 8. In this stage, as long as the client has the IPFS address of the data shards, the data integrity can be verified. The corresponding description of each step number is shown as follows:

step 5: Clients send data shards IPFS address to IPFS.

step 6: IPFS returns corresponding data shards and auxiliary information to the client.

step 7: Clients send data shards and auxiliary information to BC, then *root*2 is calculated by computing contract according to this information.

step 8: BC compares $root_1$ and $root_2$ by comparing contract, and returns the result to clients.

Through the above two stages, the work of data storage and integrity verification has been completed.

5.2. Smart contract design

Since the implementation of the smart contract requires the consumption of gas, the design of the smart contracts should be as short as possible. There are two types of smart contracts in our system, which are described as follows:

1. **Storage contract**. In our system, we denote the storage contract as *storageContract*. We use *storageContract* to store data on the blockchain. In the third step of the data integrity verification preparation stage, the client needs to store *root*1, the root of the Merkle tree, on the blockchain. At this point, the blockchain implements data storage by executing the data storage function *storeAddress*(). The instance of the *storageCont ract - storageContractInstance* will be called in *storeAddress*(). The pseudo-code for *storeAddre ss*() is shown in Algorithm 1.

2. **Comparison contract**. The comparison contract is denoted as *compareContract* in our system. We use *compareContract* to compare whether the two parameters passed to the contract are equal. In step 4 of the validation stage of data integrity validation, the smart contract needs to compare the two Merkle root $root1$ and $root2$ received. At this point, the smart contract executes the data comparison function *integrityVerify*() to determine if data integrity is guaranteed. The instance of the *compareContract* - *compareContractInstance* will be called in *integrityVerify*(). The pseudo-code for *integrityVerify*() is shown in Algorithm 2.

Algorithm 1 storeAddress (data, storageContractInstance)

Input: *data*, *storageContractInstance*
Ouput: *null*

```
1: if storageContractInstance == null then
2:     throw an error: the storageContract was not deployed successfully;
3:     return;
4: else
5:     call the set.sendtansacton() function of storage ContractInstance to store the data in the storage contract;
6: end if
```

Algorithm 2 integrityVerify(root1, root2, compareContractInstance)

Input: $root1$, $root2$, *compareContractInstance*
Ouput: *bool*

```
1: if compareContractInstance == null then
2:     throw an error: the compareContract was not deployed successfully;
3:     return;
4: else
5:     hash root1 and root2 separately to obtain the corresponding result r1 and r2;
6:     call the compare.call() function of compare ContractInstance to compare r1 and r2;
7:     if r1 == r2 then
8:         return true;
9:     else
10:        return false;
11:    end if
12: end if
13:
```

5.3. Function design

In our two-stage data integrity verification framework, the implementation of each step needs to be supported by corresponding functions. This subsection details the design of several major functions.

1. **Setup(data)** $\rightarrow$ $root1$**.** This function is executed by data owners. In the data preparation stage, the data owner will use the *Setup*() function to process the data each time before uploading the data to ECS. Then, the root of the Merkle tree will be generated. The pseudo-code for *Setup*() is shown in Algorithm 3. *sliceData*(*data*) is a function that shards the original data to obtain multiple data shards. The specific sharding method can be modified according to different sharding schemes. *publicTree*(*Digests*) is a function that used to obtain the first node of the common part of the Merkle tree. *merkleRoot*(*merkleTree*) is a function that calculates the Merkle tree root through the Merkle tree.
2. **computeRoot(**s_i**, auxiliary)**$\rightarrow$ $root2$**.** This function is used to calculate the root of the Merkle tree. Different from calculating $root1$ in the preparation stage, this function calculates $root2$ based on the auxiliary information of the shard and the random challenge number s_i corresponding to the shard. In the fourth step of the data integrity verification stage, this function will be triggered. This function corresponds to step 6 in Fig. 8 and the pseudo-code of this function is shown in Algorithm 4.

Algorithm 3 Setup(data) $\rightarrow$ $root1$

Input: *data*
Ouput: $root1$

```
1: if data == null then
2:     root1 == null;
3:     print error: The input is empty;
4: else
5:     shards = sliceData(data);
6:     get shards length len;
7:     for i = 0 to len
8:         choose challenge number s_i for shards_i;
9:         mapping s_i to (s_i, shards_i);
10:        Digest_i = hash(s_i, shards_i);
11:        add Digest_i to the collection Digests;
12:    merkleTree = publicTree(Digests);
13:    root1 = merkleRoot(merkleTree);
14: end if
15: return root1;
```

Algorithm 4 computeRoot(*auxiliary*, s_i)$\rightarrow$ $root2$

Input: *auxiliary*, s_i
Ouput: $root2$

```
1: if auxiliary == null || s_i == null then
2:     root2 = null;
3:     print error;
4: else
5:     split auxiliary to get shard_i and auxiliary information;
6:     root2 = hash(shard_i + s_i, auxiliary information);
7: end if
8: return root2
```

6. Experiments

In this section, we conduct experiments to test different structures of Merkle trees and the performance of sampling verification. Furthermore, we also measure the gas consumption of different functions in smart contracts.

6.1. The structure of Merkle trees

We conduct the experiments under three different Merkle tree structures, which are Binary Branching tree (BBT), Four-Branching tree (FBT), and Eight-Branching tree (EBT). Then assuming the total number of shards n is from 16 to 16 384 (16, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16 384). The performance of the three Merkle tree 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. 9 shows the relationship between the verification latency and the total number of shards. m represents the number of branches of the Merkle tree. Since n and m need to satisfy the relationship $n = m^k (k = 0, 1, 2, \ldots,)$ to form a full tree, the n that different m can take is not completely the same. In Fig. 9,

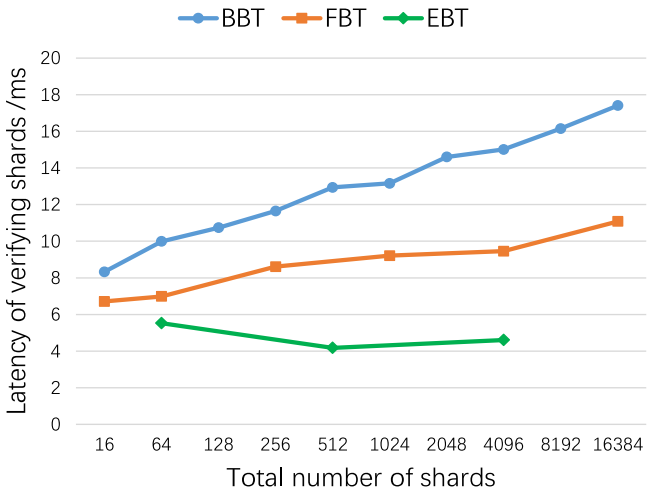

Fig. 9. The relationship between the verification latency and the total number of shards.

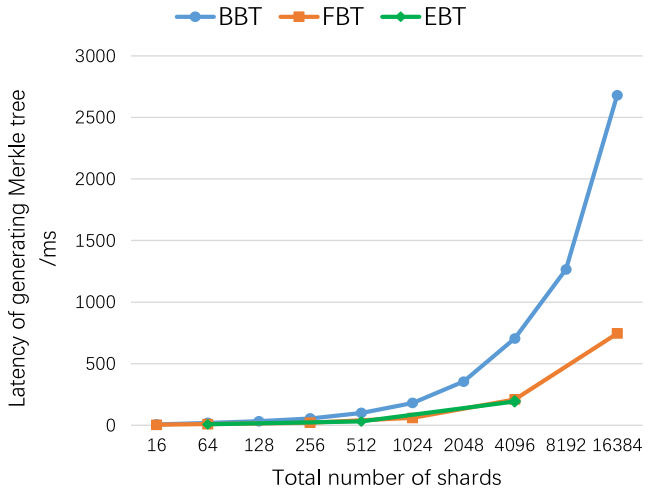

Fig. 10. The relationship between the latency of generating Merkle Trees and the total number of shards.

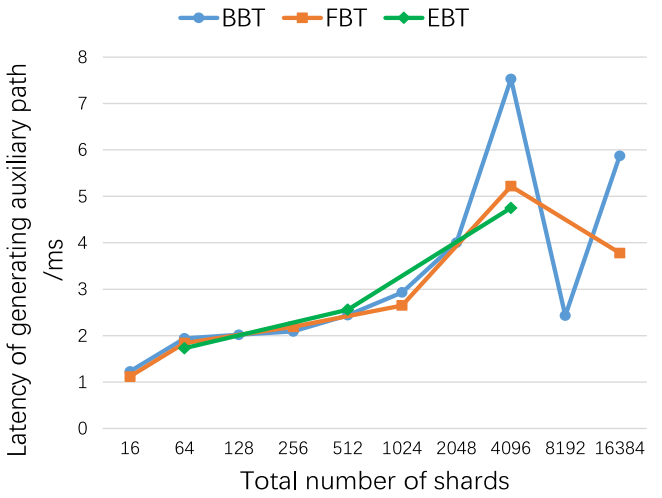

Fig. 11. The relationship between the latency of generating auxiliary path latency and the total number of shards.

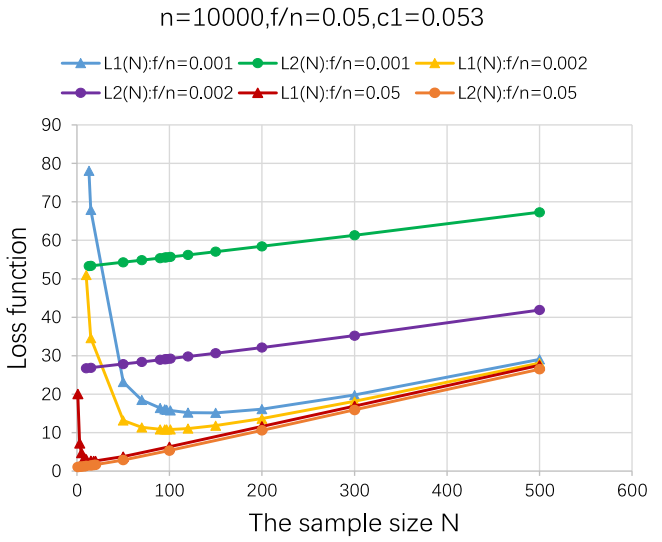

Fig. 12. The relationship between Loss Function $L1(N)$, $L2(N)$ and the sample size N when f/n changes.

the total number of shards that BBT, FBT, EBT can obtain are not exactly the same. Fig. 9 shows that EBT performs best and BBT performs worst in terms of the time cost of verifying shards. Fig. 10 shows the relationship between the latency of generating Merkle Trees and the total number of shards.

From Fig. 10, we can see that FBT and EBT are significantly better than BBT. This is mainly reflected in the following two aspects : (1) the latencies of generating the Merkle Tree of FBT and EBT are 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 computational delay, FBT and EBT are better than BBT in computational overhead.

In terms of the auxiliary path, the previous theoretical analysis has concluded that the auxiliary path size increases with the number of branches. However, each additional element of the auxiliary path is a hashed string, the increased storage space is small. Fig. 11 shows the relationship between the latency of generating auxiliary path and the total number of shards. It can be seen from Fig. 11 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 structures are negligible. In summary, the performances of FBT and EBT are better than BBT.

6.2. Sample size

In Section 4, we have introduced the Loss Function $L1(N)$ and $L2(N)$, which include c_0, c_1, n, f, N, to decide the suitable sample sizes. 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 values of f, which are $f/n = 0.001, f/n = 0.002, f/n = 0.01, f/n = 0.05$. In the second set of experiments, we assume the total number of shards $n = 10000$, $f/n = 0.002$ and take four different values of $c1$, which are $c1 = 0.7, c1 = 0.4, c1 = 0.1, c1 = 0.01$. Then we compare the performance of $L1(N)$ with $L2(N)$ when N changes.

From Figs. 12 and 13, we can see that as the sample size N increases, the value of the Loss Function $L1(N)$ decreases first and

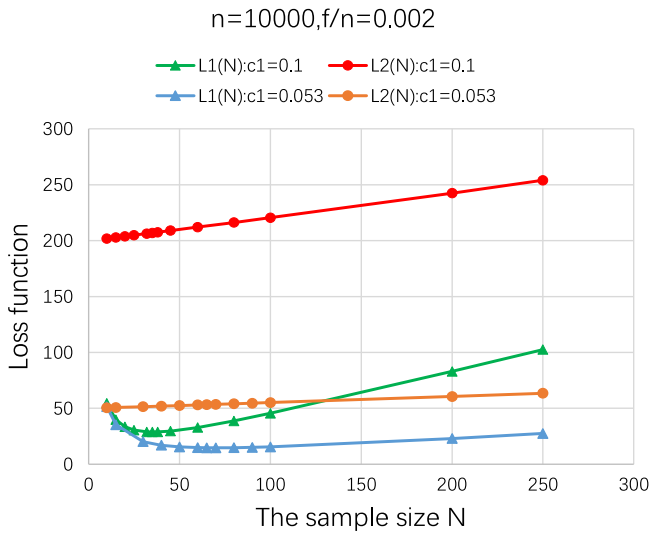

Fig. 13. The relationship between Loss Function $L(N)$, $L2(N)$ and the sample size N when $c1$ changes.

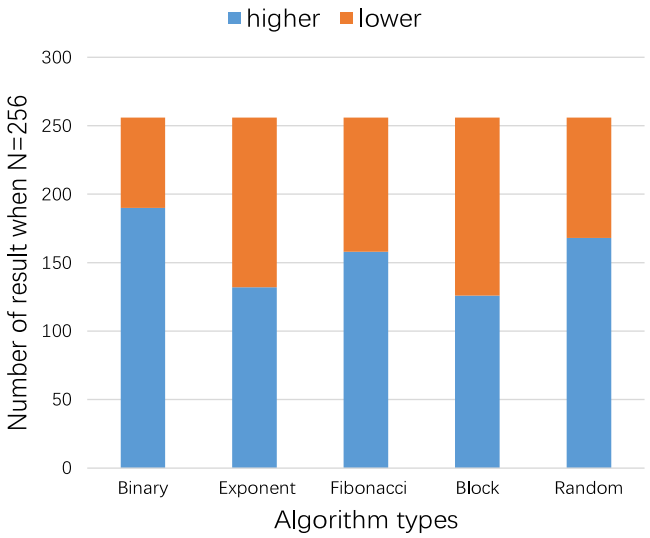

Fig. 15. Statistics on validation overhead when N = 256.

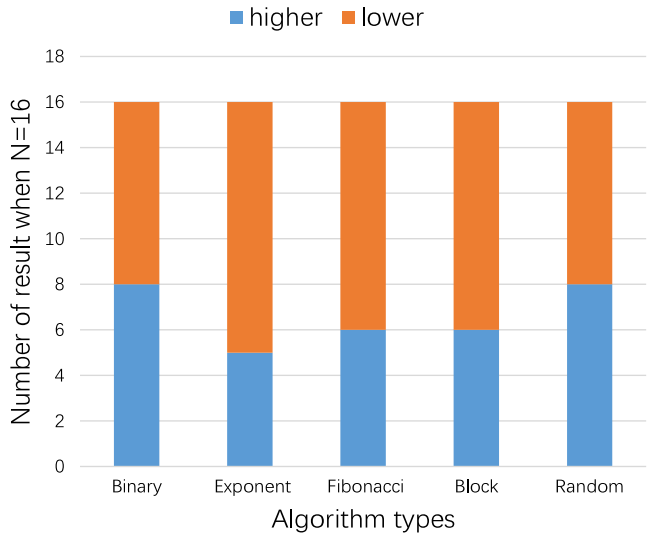

Fig. 14. Statistics on validation overhead when N = 16.

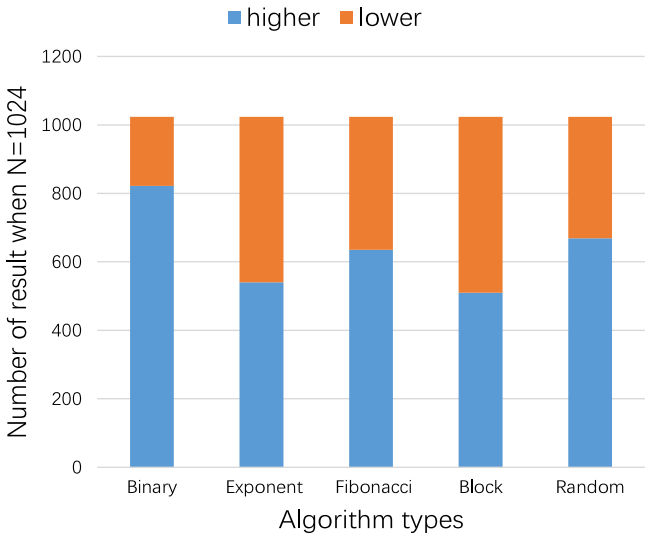

Fig. 16. Statistics on validation overhead when N = 1024.

then increases. Thus, there exists a most appropriate value of N leading to the minimum value of $L1(N)$. However, the value of the Loss Function $L2(N)$ increases linearly as N increases, and there is no optimal value of N leading to the minimum value of $L2(N)$. Therefore, it is more reasonable to define the Loss Function in the form of $L1(N)$.

From Fig. 12, we can see that when f/n is larger, the minimum value of $L1(N)$ is closer to the Y-axis and X-axis. That means the more shards fail, the smaller the optimal sample size is. As f/n increases, the minimum value of the Loss Function $L1(N)$ decreases. This implies that the overall validation performance of this system increases as the number of failed shards increases. From Fig. 13, 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.

6.3. 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 the cost of sequential validation 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.

Figs. 13–16 show that the proportion of verification cost higher than the benchmark increases with the increase of sample size N. From Fig. 14 we know that when N is small, almost each algorithm works better than the benchmark. Comparing Figs. 15–17, we can see that with the increase of sample size N, the performance of Binary verification is getting worse and worse, while the performance of other algorithms remains stable. Fibonacci verification and Random verification are work better than the baseline when $N = 16$. Exponent verification and Block

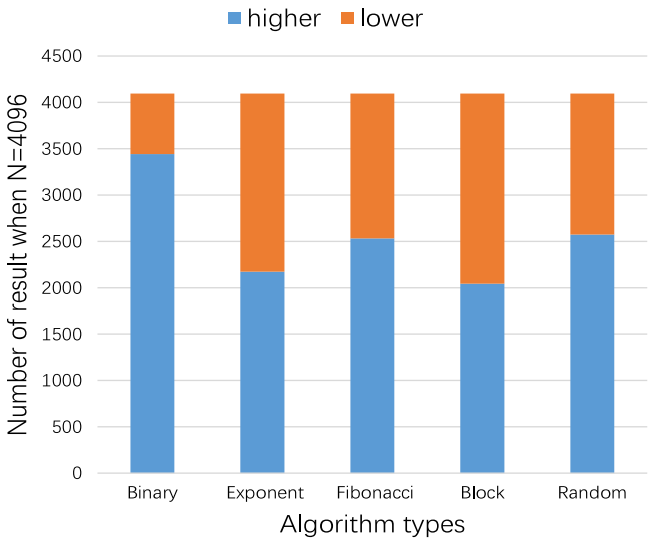

Fig. 17. Statistics on validation overhead when N = 4096.

Table 1
Transaction cost in our system.

Transaction	Gas consumption
deploy Storage Contract	233 299
storageContract.Set()	85 896
storageContract.Get()	22 908
deploy Comparison Contract	102 495
compareContract.Compare()	46 075

verification always work better than the baseline no matter how N increases. 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.

6.4. Smart contract cost

In Ethereum, every transaction will consume gas. Transactions in our system are mainly related to smart contracts, which include deployment and invocation of the smart contracts. The cost of these transactions measured in our experiment are listed in Table 1.

Table 1 shows that deploying a smart contract consumes more gas than calling a smart contract. We tested the amount of smart contract gas consumed under different number of files, ranging from 16 to 1024. 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 contract root, and the size of the root is independent of the number of files. We only counted gas consumption, but did not provide specific cost consumption. Because the cost of consumption in the Ethereum is equal to the gas consumption multiplied by the price of gas. While in the private network of Ethereum, the price of gas can be set by developers themselves, and there is no uniform standard. Therefore, we only measure the cost of the contract by the gas consumption.

7. Performance and security analysis

In our scheme, we adopt IPFS as the edge-cloud storage, and use blockchain to assist data integrity verification. To verify the rationality of our proposed framework, we compare our framework with the other two frameworks mentioned in related work. Table 2 shows the comparison result of proposed framework with existing frameworks in different parameters.

The explanation of Table 2 is shown below. In Sia [19], users can rent out their free disk space as a cloud storage server. This allows the disk space to be used. Thus, memory utilization has been improved relative to traditional cloud storage structures [20]. In our framework, we adopt IPFS to provide storage services. IPFS is a content-based address file system, that is, the file with the same content will return the same storage address no matter how many copies are stored to IPFS. Therefore, when multiple files of the same content are stored on IPFS, IPFS only stores one file. Hence, memory utilization is higher. Moreover, IPFS is a global, point-to-point distributed file system. Anyone with a file's IPFS address can access the corresponding file over the network at anytime. File access is faster, easier, and more public than traditional file systems.

Compared with centralized cloud storage structure in [20], decentralized cloud storage structure is more reliable due to the data stored on multiple cloud nodes in [19] and in our framework. In our framework, data is sliced into serval shards and then dispersed stored on different nodes. This storage mechanism ensures that no single cloud storage node has the complete data, which make data storage more security. While in [20] and [19], a cloud storage server with complete data may be able to exploit the data privately, thus breaking the security of data and violating the privacy of users. In addition, we add random number to each shard and then hash them to get the Merkle tree leaves. The random number is saved by the client, which ensures that only the client can generate the correct leaf node. Random numbers can be saved by the data owner themselves or distributed to trusted client. This ensures that the right to verify data is fully in the hands of the data owners, which increases the security and privacy of data validation.

In our scheme, we propose sampling verification. By selecting some data pieces to verify the integrity of the whole file, the verification overhead and delay are reduced compared with [20] and [19]. Since every transaction executed on the blockchain is recorded on the blockchain, the verification invoked by the client will be recorded on the blockchain and cannot be maliciously tampered. These records can be traced, which increases the reliability and transparency of data integrity verification.

8. Conclusion and future work

In this paper, we propose a general data integrity verification framework in edge-cloud storage. It solves the problem of incredibility exists in traditional verification mechanism by utilizing blockchain. We adopt edge nodes to jointly maintain a blockchain, which makes our scheme better combined with edge-cloud storage scenario. In our framework, we describe the data integrity verification process in detail and analyze the verification performance under various Merkle tree structures. To improve the verification performance while keeping a high verification precision, we propose rational sampling strategies and calculate the optimal sample size. For data integrity verification, we design two types of smart contracts and give the detailed description for the design. Finally, we demonstrate the feasibility of the proposed framework by implementing a prototype system and validating our analysis through extensive experiments. As the future work, we will deploy our framework under larger number of clients and explore the optimization to combine blockchain with edge-cloud storage.

Table 2
Comparison of proposed framework with existing frameworks.

Parameter	Proposed framework	Framework in [20]	Framework in [19]
Memory utilization	High	Low	Middle
Convenience of data access	Anytime and anywhere	Completely dependent on cloud servers	Some rely on cloud servers
Reliability of data storage	Data storage is more reliable due to the data stored on multiple different nodes	Centralized cloud storage may have single point of failure	Data storage is more reliable due to the data stored on multiple different nodes
Security of data storage	Data storage is more secure because no single cloud storage node stores complete data	Data storage is less secure because the cloud storage server stores your complete data	Data storage is less secure because the cloud storage server stores your complete data
Permissions of data integrity verification	Permissions of verification are determined flexibly by the data owner	Anyone	Verification are performed by cloud storage server
Performance of verification	Lower overhead and higher real-time	Higher overhead and lower real-time	Higher overhead and lower real-time
Reliability and transparency of verification	High	Low	High

Table 3
Notations in this paper.

Notation	Description
TPA	Third Party Auditor
ECS	Edge-Cloud Storage
CoT	Cloud-Assisted Internet of Things
BC	Blockchain
DA	Data Owner
DC	Data Consumer
D_i	Digest of $shard_i$ in the Multi-Branch Merkle Tree
N	The sample size of sampling
n	The total number of shards
m	Vhe branch number of the Merkle trees
C	Verification cost
P_V	Verification precision
$L(N)$	Loss function
BBT	Binary-Branching Merkle trees
FBT	Four-Branching Merkle trees
EBT	Eight-Branching Merkle trees

CRediT authorship contribution statement

Dongdong Yue: Conceptualization, Methodology, Software, Writing - original draft. **Ruixuan Li:** Conceptualization, Methodology, Supervision. **Yan Zhang:** Methodology, Writing - review & editing. **Wenlong Tian:** Data curation, Software, Visualization, Investigation. **Yongfeng Huang:** Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Key Research and Development Program of China under grants 2016YFB0800402 and 2016QY01W0202, National Natural Science Foundation of China under grants U1836204, U1936108, 61572221, 61433006, U1401258 and 61502185 and Major Projects of the National Social Science Foundation under grant 16ZDA092.

Appendix

To simplify the description, symbols used in this paper are shown in Table 3.

References

[1] S. Aldossary, W. Allen, Data security, privacy, availability and integrity in cloud computing: issues and current solutions, Int. J. Adv. Comput. Sci. Appl. (IJACSA) 7 (2016) 485–498.
[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, Provable data possession at untrusted stores, in: ACM Conference on Computer and Communications Security (CCS 2007), 2007, pp. 598–609.
[3] G. Ateniese, R.D. Pietro, L.V. Mancini, G. Tsudik, Scalable and efficient provable data possession, in: The 4th International Conference on Security and Privacy in Communication Netowrks (SecureComm 2008), 2008, pp. 1–10.
[4] J. Benet, IPFS - content addressed, versioned, P2P file system, 2014, CoRR abs/1407.3561.
[5] B. Chen, R. Curtmola, Robust dynamic remote data checking for public clouds, in: The 19th ACM Conference on Computer and Communications Security (CCS 2012), ACM, 2012, pp. 1043–1045.
[6] R. Curtmola, O. Khan, R. Burns, G. Ateniese, Mr-pdp: Multiple-replica provable data possession, in: The 28th International Conference on Distributed Computing Systems (ICDCS2008), IEEE, 2008, pp. 411–420.
[7] Y. Deswarte, J.-J. Quisquater, A. Sadane, Remote integrity checking, in: Integrity and Internal Control in Information Systems VI, Springer, 2004, pp. 1–11.
[8] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, V. Sassone, Blockchain-based database to ensure data integrity in cloud computing environments, in: The First Italian Conference on Cybersecurity (ITASEC 2017), 2017, pp. 146–155.
[9] L. Huang, G. Zhang, S. Yu, A. Fu, J. Yearwood, Seshare: Secure cloud data sharing based on blockchain and public auditing, Concurr. Comput.: Pract. Exper. 31 (22) (2019).
[10] A. Juels, B.S. Kaliski Jr, Pors: Proofs of retrievability for large files, in: The 14th ACM Conference on Computer and Communications Security (CCS 2007), ACM, 2007, pp. 584–597.
[11] B. Liu, X.L. Yu, S. Chen, X. Xu, L. Zhu, Blockchain based data integrity service framework for iot data, in: The 24th IEEE International Conference on Web Services (ICWS 2017), IEEE, 2017, pp. 468–475.
[12] I. Lujic, V.D. Maio, I. Brandic, Efficient edge storage management based on near real-time forecasts, in: The 1st IEEE International Conference on Fog and Edge Computing (ICFEC2017) Madrid, Spain, May 14-15, 2017, pp. 21–30.
[13] I. Marco, L.K. R., The truth about blockchain, in: Harvard Business Review, Harvard University, 2017.
[14] M.R. Nosouhi, S. Yu, W. Zhou, M. Grobler, H. Keshtiar, Blockchain for secure location verification, J. Parallel Distrib. Comput. 136 (2020) 40–51.
[15] J. Pan, J. McElhannon, Future edge cloud and edge computing for internet of things applications, IEEE Internet Things J. 5 (1) (2018) 439–449.
[16] F. Sebé, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, J.-J. Quisquater, Efficient remote data possession checking in critical information infrastructures, IEEE Trans. Knowl. Data Eng. 20 (8) (2008) 1034–1038.
[17] H. Shacham, B. Waters, Compact proofs of retrievability, J. Cryptol. 26 (3) (2013) 442–483.
[18] S.K. Sharma, X. Wang, Live data analytics with collaborative edge and cloud processing in wireless iot networks, IEEE Access 5 (2017) 4621–4635.
[19] D. Vorick, L. Champine, Sia: simple decentralized storage, 2014, http://www.sia.tech/.

[20] Q. Wang, C. Wang, J. Li, K. Ren, W. Lou, Enabling public verifiability and data dynamics for storage security in cloud computing, in: The 14th European Symposium on Research in Computer Security (ESORICS 2009), Saint-Malo, France. Proceedings, 2009, pp. 355–370.
[21] C. Wang, Q. Wang, K. Ren, W. Lou, Ensuring data storage security in Cloud Computing, in: The 17th International Workshop on Quality of Service (IWQoS 2009), Charleston, South Carolina, USA, 2009, pp. 1–9.
[22] J. Xing, H. Dai, Z. Yu, A distributed multi-level model with dynamic replacement for the storage of smart edge computing, J. Syst. Archit. - Embedded Syst. Des. 83 (2018) 1–11.

Dongdong Yue received her B.S. degree from School of Computer Science and Technology at Wuhan University of Science and Technology in 2017. Now she is a master in the Intelligent and Distributed Computing Laboratory, School of Computer Science and Technology, Huazhong University of Science and Technology. Her research interests include blockchain, access control, and cloud storage. She is a student member of the IEEE.

Ruixuan Li received the B.S., M.S., and Ph.D. degrees from School of Computer Science and Technology at Huazhong University of Science and Technology in 1997, 2000, and 2004 respectively. He is currently a professor of School of Computer Science and Technology at Huazhong University of Science and Technology, and is the director of the Intelligent and Distributed Computing Laboratory. His research interests include blockchain, cloud computing and big data security. He is a member of the IEEE and ACM.

Yan Zhang obtained his Ph.D. degree in Computer Science from the University of Sydney, Australia, in 1994. He is a professor at University of Western Sydney and the leader of Artificial Intelligence Research group in Western Sydney University. His research interests include knowledge representation and reasoning, ontology based data access, logic programming, intelligent agents, and information security. Prof Yan Zhang has published over 150 articles in these research areas in various international journals and conferences.

Wenlong Tian received his M.S. degree and Ph.D. degrees in School of Software and the School of Software from Huazhong University of Science and Technology in 2015 and 2019 respectively. Now he is currently a lecturer in the School of Computer Science and Technology, University of South China. His research interests include cloud computing, system security, and big data management. He is a member of the IEEE.

Yongfeng Huang received the Ph.D. degree in computer science and engineering from the Huazhong University of Science and Technology, in 2000. He is a professor in the Department of Electronic Engineering, Tsinghua University, Beijing, China. His research interests include cloud computing, data mining, and network security. He is a senior member of the IEEE.