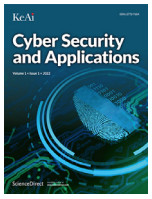
Cyber Security and Applications 1 (2023) 100017

Contents lists available at ScienceDirect

Cyber Security and Applications

journal homepage: http://www.keaipublishing.com/en/journals/cyber-security-and-applications/



Blockchain-powered distributed data auditing scheme for cloud-edge healthcare system

Yi Li [a], Meiqin Tang [b,*]

[a] Nanjing University of Information Science and Technology, No. 219, Ningliu Road, Pukou District, Nanjing, 210044, China
[b] Wuxi Vocational College of Science and Technology, No. 8 Xinxi Road, Wuxi, 214029, China

ARTICLE INFO

Keywords:
Data auditing
Healthcare system
Blockchain
Cloud-edge computing

ABSTRACT

Cloud-edge healthcare system provides storage and computing functions at the hospital servers, bringing low latency for doctors and patients. However, hospital servers cannot be trusted and have limited computing resources. Data integrity verification for the cloud-edge healthcare system is an urgent concern. To this end, we proposed a data integrity auditing scheme based on blockchain. First, a distributed data integrity verification method without a third-party auditor is designed. The data are divided into smaller parts and hashed into a hash table. The verification tag is constructed according to the column of the hash table and secret string generated by a pseudo-random function. Then, a detailed blockchain-based data integrity auditing scheme is proposed, including Proof of Auditing Frequency and block structure. Besides, a security analysis for the common attacks is given. Finally, the proposed scheme is evaluated against two start-of-the-art schemes in a simulated cloud-edge healthcare system. The results demonstrate that the proposed scheme can verify data integrity without losing efficiency.

The healthcare system is an actual demand for society. If data has to be shared among multiple healthcare organizations, cloud-edge computing will be prominent under architecture. Based on this precondition, blockchain can be a technology to support tamper proofing and anti-collusion. Different hospitals tend to cooperate and share medical data for better medical and diagnostic services. Cloud computing [1] architecture can serve as a straightforward solution. Users in various hospitals, such as doctors and patients, can directly upload data to the central cloud server to share medical data. However, the centralized cloud computing model faces the problems of a single point of failure and high latency between some users and cloud servers. For this reason, medical systems with cloud-edge computing architecture are favored [2]. While the hospital servers acts as an edge servers, users directly store data with the nearest hospital server, as shown in Fig. 1.

Nothing is perfect. Cloud-edge healthcare system brings the new problem of data insecurity [3]. The hospital servers cannot be trusted like the cloud server by users. Moreover, different hospital servers belong to various organizations. They do not trust each other. Provable Data Possession [4] and Proof of Retrievability [5] are two classical data integrity verification methods for cloud computing. The schemes proposed after PDP and POR are basically within these two paradigms. Traditional data integrity verification methods for cloud computing are not suitable for cloud-edge healthcare system for two reasons: (1)Third-party auditors (TPAs) for auditing are not ideal for the distributed characteristics of the cloud-edge healthcare system. (2) The hospital server has limited computing resources. In a nutshell, verifying the data integrity for the cloud-edge healthcare system is the problem we want to figure out in this paper.

Blockchain provides a possible direction for data integrity auditing for cloud-edge healthcare system [6]. Each hospital server can become a blockchain node to join the consensus and maintain a distributed ledger for data integrity auditing. Intuitively, cloud-edge computing is a prominent architecture for a data-shared healthcare system, and blockchain is a technology to support tamper-proofing and anti-collusion.

However, there are still pending solutions to establish a decentralized architecture among the hospital servers, how to guarantee the efficiency of distributed data integrity verification, and how to defend against the threats of malicious hospital servers. To this end, we proposed an efficient data integrity auditing scheme based on blockchain. Our main contributions are as follows:

- A distributed data integrity verification method without third-party auditor is proposed. Based on the constructed hashTable, each hospital server can audit the data in a distributed manner.
- A blockchain-based auditing scheme is designed for cloud-edge healthcare system, which includes the consensus protocol Proof of Auditing Frequency (PoAF) and related block structures.

* Corresponding author.
E-mail addresses: 20211155002@nuist.edu.cn (Y. Li), tmq220@163.com (M. Tang).

https://doi.org/10.1016/j.csa.2023.100017
Received 20 November 2022; Received in revised form 7 March 2023; Accepted 3 April 2023
Available online 6 April 2023

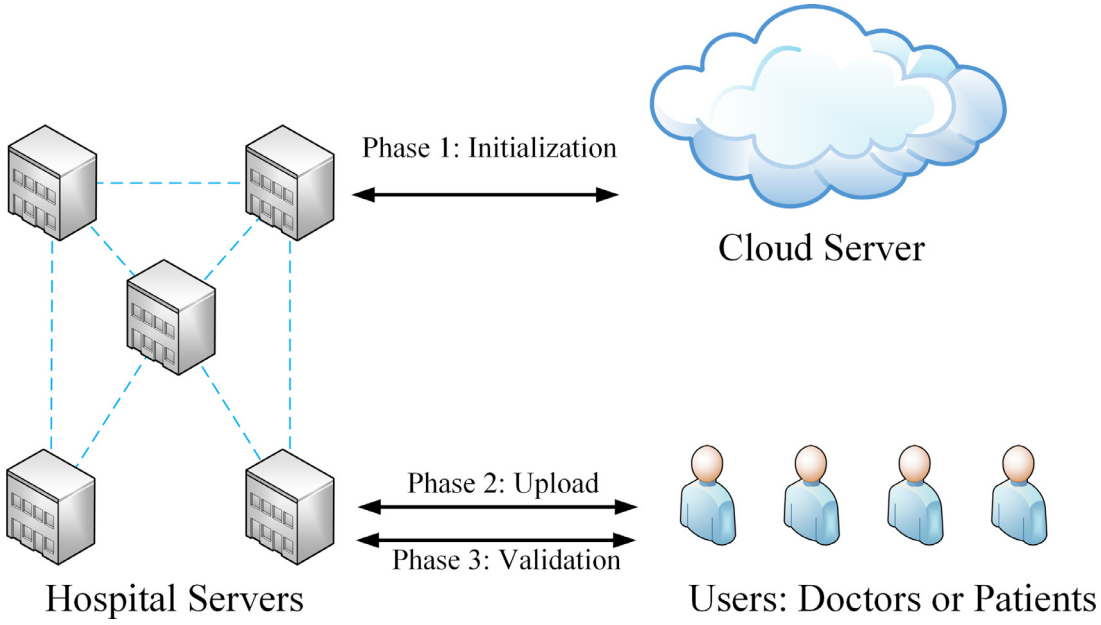


Fig. 1. Edge Computing Architecture.

- The proposed scheme is evaluated against two start-of-the-art schemes in a simulated cloud-edge healthcare system and comprehensively evaluated.

The rest of this paper is organized as follows. The related work is discussed in Section 2. The proposed scheme is given in Section 3 specifically. The results of experiments and the performance evaluation of the proposed scheme are shown in Section 4. Finally, Section 5 summarizes this paper and points out our future plan.

1. Related work

Currently, there is no direct research on cloud-edge computing medical systems data auditing. Therefore, this section will focus on the related work of cloud computing data auditing and cloud-edge computing data auditing.

1.1. Cloud computing data auditing

The first data auditing mechanism called Proofs Of Retrievability (POR) is proposed in 2007 [7]. This scheme allows users to verify the integrity of outsourced data to ensure safe storage of data, but its disadvantage is that it cannot be publicly audited. Ateniese et al. [4] propose Provable Data Possession (PDP). A third-party auditors verify the integrity of data on behalf of users, reducing the cost of users. In practical applications, the demand for dynamic operation is more common. Ateniese et al. [8] propose a scalable PDP scheme, but this method only supports part of dynamic operation. Shacham and Waters [9] improve the POR model, combined with homomorphic authentication technology to reduce communication overhead. Erway et al. [10] introduce a rank-based authentication skip table and propose the first data integrity audit scheme that supports full dynamic updates. Wang et al. [11] propose a new data dynamic update integrity verification scheme based on Merkel Hash Tree (MHT). This scheme supports public auditing and reduces some overhead, but cannot resist replacement attacks. Mao et al. [12] improve MHT and propose a position-aware Merkle Tree (PMT), with position information in each node to resist replacement attacks. Rao et al. [13] design a dynamic audit scheme based on batch authentication MHT to solve the anti-collusion problem. Bowers et al. [14] design a scheme based on POR. Its authentication does not need to decrypt the file. Yuen [15] propose an auditable consortium chain. Users participate in transactions after being authenticated by the authorized party, preventing malicious users from abusing privacy attributes, and can track malicious users. Gao et al. [16] propose a cloud data audit scheme that supports data deduplication to achieve low-entropy security. Malicious clouds cannot forge validators to pass audits, and users do not need to interact with third-party auditors. Xu et al. [17] propose a public cloud auditing scheme that periodically updates the audit authorization code to prevent malicious clouds from tampering with file keys.

1.2. Cloud-edge computing data auditing

For edge servers, a collaborative model has to be established. Li et al. propose an app vendor's cache data such as VR video integrity for the edge computing environment [18]. They divide the treats into unexpected failures and cheating attacks. They give their scheme around correctness, lightweight, and security objectives, including a light sampling-based probabilistic approach and a new data structure named variable Merkle hash tree. Then, in [19], they propose a cooperative scheme. A distributed consensus is employed to form an edge caching system. All the edge servers are responsible for guaranteeing data integrity. In addition, in [20] and [21], the privacy and efficiency are improved respectively. Li et al. have achieved remarkable results by focusing on solving the edge data integrity problem of application manufacturers. However, the above scenarios all focus on a single application vendor's data integrity verification needs. Each edge server is responsible for storing many application vendors' data. Application vendor servers have more computing power and storage space than single-person servers and edge servers. Therefore, these methods are unsuitable for heterogeneous edge servers to provide personal services.

In [22], a blockchain-based framework is proposed without TPA for data integrity verification. Merkle tree-based validation methods and sampling strategies contribute to the framework. In addition, a prototype is implemented on Ethereum [23] by deploying smart contracts [24], which may not be suitable for resource-limited edge servers. Fan et al. propose a consensus for a blockchain-based data integrity framework [25]. They divide the framework into two-layer: the private part and the edge part. The private part maintains a private chain for IoT devices, while the edge part is a public chain constructed by edge servers. Their verification method is similar to Yue et al. [22], based on

the Merkle tree [26]. Two blockchain-based data integrity verification frameworks in [22] and [25] adopt a similar way based on the Merkle tree and sampling. The success rate of verification cannot be guaranteed. To this end, we design novel scheme to improve the success rate.

Therefore, it is urgent to design a blockchain-powered distributed data auditing scheme for the healthcare system in this paper.

2. Blockchain-powered distributed data auditing scheme

2.1. Scheme overview

The proposed system framework has three roles: users, hospital server, and cloud server. Users, such as doctors and patients, can be trusted. They upload their data to the nearest hospital servers and request the service of data integrity verification. Hospital servers cannot be trusted. They save the data for the users, conduct the process of data integrity verification, and attend to the blockchain consensus. The malicious edge servers try to hide the fact of losing data. The cloud server is semi-trust. It is only responsible for setting the initial system parameters and generating the genesis block. Besides being distributed without a trusted third party, blockchain can also endue the advantages of tamper-proofing and anti-collusion.

The notation used in this section is shown in Table 1.

The scheme includes 3 phase, as shown as follows:

- Phase 1: Hospital servers and the cloud server generate their public keys and private keys. Then they hash their public keys as their addresses and broadcast the public keys and addresses to the other servers. Once the cloud server receives all the public keys of hospital servers, it will assign an order to the hospital servers and generate the first block of the blockchain, in other words, the genesis block. The genesis block includes the timestamp, edge servers order, addresses, and public keys. Then the cloud server sends the genesis block to all hospital servers.
- Phase 2: The user (doctor or patient) first generates the public and private keys and hash the public key as his address. Then the user processes the data for the verification information. After that, the user uploads the data to the nearest hospital server and sends the verification information to the blockchain. After consensus, the verification information is stored on the blockchain between all the edge servers.
- Phase 3: The user (doctor or patient) requests the service of data integrity verification. The hospital servers validate the data integrity in a distributed way according to the verification information stored on the blockchain. After consensus between the edge servers, the user will get the result of the data integrity verification.

2.2. Distributed data integrity verification method

We design a distributed verification method including three main phases: Tag Generation, Verification, and Tag Updating.

Table 1
Notation.

Symbols	Meaning
$F = \{f_1, f_2, \ldots, f_i, \ldots, f_N\}$	The data file with m data blocks
N	The order of cyclic group
g	Generator of cyclic group
$H_1()$ and $H_2()$	Two hash functions
$PRF(k)$	The pseudo random function with key k
β_1 and β_2	The length of the digest of the two hash functions
VT	The matrix of verification message
S	The secret string

2.2.1. Tag generation

Due to the resource limitation of the edge servers and blockchain, we construct the verification tag of data based on hash function and pseudo rand function. First, the data is partitioned into N data blocks, represented by $F = \{f_1, f_2, \ldots, f_i, \ldots, f_m\}$. Then, the VT matrix is constructed with a size of $(\beta_1 \times (\beta_2))$. The construction of VT is shown in Algorithm 1.

Algorithm 1 Tag construction for data F.

1: INPUT: $F = \{f_1, f_2, \ldots, f_i, \ldots, f_m\}, g, N, \beta_1, \beta_2, H_1(), H_2(), PRF(), k$
2: OUTPUT: VT
3: $S = PRF(k)$
4: Initial $hashTable$ as a binary matrix with a size of $m \times \beta_1$
5: **for** $i = 1$ to m **do**
6: Set the ith row of $hashTable$ as $H_1(g^{s_i + f_i} mod\ N)$
7: Initial VT as a binary matrix with a size of $\beta_1 \times \beta_2$, elements 0
8: **for** $i = 1$ to β_1 **do**
9: Set the $temp$ as the ith column of$hashTable$
10: Set the ith row of VT as $H_2(temp)$
11: **Return** VT

As shown in Algorithm 1, the explanation of inputs and outputs can be found in Table 1. In line 3 to 6, the $hashTable$ is generated according to each data block and secret string s_i of F, $H_1(g^{s_i + f_i} mod\ N$. Then, in line 7 to 11 is for constructing VT : first, retrieve the ith column of $hashTable$ as $temp$. Second, calculate $H_2(temp)$. Third, set the ith row of VT as $H_2(temp)$.

2.2.2. Verification

After phase 1 - tag generation, verification can be carried out by the other hospital servers distributively. The procedure is shown in Algorithm 2.

Algorithm 2 Verification for data F.

1: INPUT: $VT, N, m, H_2(), \beta_1, \beta_2, PRF(), k$
2: OUTPUT: True of False
3: The hospital server calculates the $S = PRF(k)$
4: The hospital server reconstruct the $hashTable'$
5: The validator (other hospital server) decides the times for sampling t
6: **for** $i = 1$ to t **do**
7: The validator generates a random number ($rand$) from $[1, \beta_1]$
8: The validator requests the ith column of $hashTable'$
9: The validator calculates $H_2(hashTable'_{rand})$and compares with the VT
10: **if** $H_2(hashTable'_{rand}) \neq VT_{rand}$ **do**
11: **Return** False
12: **Return** $Ture$

As shown in Algorithm 2, the explanation of inputs can be found in Table 1. The output is the verification results: True or False. In line 3 to 4, the hospital server calculates the secret string and re constructs the $hashTable'$ by the current data file stored in it. In line 5 to 12, the validator verify the data according the $hashTable'$ and the VT. First, the decides the times for sampling. Second, it request the $rand$th column of $hashTable'$, hashes this value, and compares with the $rand$th row of VT. Third, if the t times sampling all pass the verification, it returns $True$, otherwise returns $False$. The Algorithm 2 can be carried out by different hospital servers in parallel, that is, a distributed manner.

2.2.3. Tag updating

After carrying out verification, the hospital server knows the secret string kept by the user. As a result, the old VT is invalid, and the user has to update the verification tag. Due to the communication overhead,

it is impractical to resend the file to the user. Thus, we design a Tag Updating method without resenting the file, shown in Algorithm 3.

Algorithm 3 Tag Updating.

1: INPUT: $g, N, m, H_1(), H_2(), \beta_1, \beta_2$
2: OUTPUT: VT'
3: The hospital server sends the $g^{f_1} \ldots g^{f_m} mod\ N$ to user
4: The user chooses an new k' and calculates $S' = PRF(k')$
5: **for** $i = 1$ to m **do**
6: Set the ith row of $hashTable$ as $H_1(g^{s'_i + f_i} mod\ N)$
7: Initial VT' as a binary matrix with a size of $\beta_1 \times \beta_2$, elements 0
8: **for** $i = 1$ to β_1 **do**
9: Set the $temp$ as the$i$th column of$hashTable$
10: Set the ith row of VT' as $H_2(temp)$
11: **Return** VT'

As shown in Algorithm 3, the explanation of inputs and output can be found in Table 1. In line 3, the hospital server sends $.g^{f_m}\ mod\ N$ to user. Then in line 4, the user chooses an new key k' and generates new secret string S'. After that, the user constructs a new VT and sends to blockchain.

2.3. Blockchain design

Based on the proposed distributed verification method, the detailed blockchain design for could-edge computing is given out in this section. There are three phases: setup, consensus, upload, and verification.

2.3.1. Setup phase

In this stage, the cloud server has to determine the initialization, including the hash functions, the pseudo random function, the number of data blocks, the blockchain nodes, and the block structure. proposed scheme has three block structures: genesis block, data upload block, and data verification block. The genesis block is used for storing the public keys of blockchain nodes. The data upload block is for recording the data verification message. The data verification block is for aggregating the verification results generated by the blockchain nodes.

2.3.2. Proof of auditing frequency

To carry out the consensus efficiently for Blockchain, we design a novel protocol called Proof of Auditing Frequency (PoAF). The detailed illustration is shown as follows.

Step 1: Each hospital server verifies the data file for t times.
Step 2: Hospital server sends the verification results to the other hospital servers.
Step 3: The other hospital servers receive all the verification results and choose the hospital server with the most verification results as the ledger node for this round consensus.
Step 4: The hospital server with the most verification results generates the block and sends to the other hospital servers.
Step 5: All the hospital servers update and sync the blockchain.

2.3.3. Data upload phase

After the setup phase, the cloud server first initializes the genesis block to start the blockchain network among edge servers. Then the user processes the data, uploads it to the nearest hospital server, and sends the tag to blockchain networks. The procedure is shown as follows:

Step 1: All the edge servers generate their private and public keys and send the public keys to the cloud server.
Step 2: The cloud server initializes the genesis block. The block header includes block height, Merkle root, and timestamp. The block body consists of all public keys of the edge servers.The Merkle root is calculated by the cloud server.
Step 3: The user generates an m-bit secret strings by $PRF(k)$ and attaches the m-bit secret string to the data file F.
Step 4: The user processes the F by the method in Tag Generation phase subsection and generates the tag VT.
Step 6: The user sends the VT to blockchain. For the upload block structure, the block header of the upload block includes hash-value of the previous block, the block height, the merkle root, the block type and timestamp. the block body consists of all columns of VT. The Merkle root of the upload block is calculated determined by the PoAF among the hospital server.
Step 7: The user uploads the data F to the nearest edge server. Without the key k of $PRF()$, the hospital server cannot construct the $hashTable$ without the k of $PRF()$.

2.3.4. Data verification phase

After uploading the data to the nearest hospital server, the user only keeps the key k of $PRF()$. When the user needs to verify the data stored in the hospital server, he only has to send the key k of $PRF()$ to the nearest hospital server and the blockchain maintained by the hospital servers returns the auditing result.The procedure is shown as follows:

Step 1: The user sends the validation request to the blockchain maintained by the hospital servers.
Step 2: The user sends his key k of $PRF()$ to the nearest edge server to reconstruct the $hashTable$. Based on the current data and the the key k of $PRF()$, the edge server calculates the $hashTable'$.
Step 3: The blockchain nodes generate a random number $rand$ between [1,β_1] and request the $rand$th column of $hashTable'$.
Step 4: The blockchain nodes verify the sample according to the method introduced in section 4.2 and send the verification result to the user.
Step 5: The user aggregates the verification results from all the blockchain nodes and send to the blockchain.
Step 6: The user requests the $(g^{f_1} \ldots g^{f_m}\ mod\ N)$ from the nearest hospital server, changes the key k' of $PRF()$, and regenerates new verification tag VT' to blockchain for the next-time auditing.

For the verification block, the block header includes the previous block's hash value, the block height, the hash value of the cascade of the block body, the block type, and the timestamp. The block body consists of the validated result of each hospital server. The hash value of the concatenation of the verification results of each blockchain nodes is determined by the PoAF. The data is stored on the nearest edge server by the users. The blockchain nodes (hospital servers) only store the verification tag and verification results.

2.4. Security analysis

The malicious edge servers try to cover up the facts that the data has been lost or modified. Detailed security analysis for five malicious behavior is given out.

2.4.1. Sybil attack

In proposed scheme, the malicious nodes try to forge identities to attend the data verification. However, all the true identities are bound with their public key and saved on the blockchain. The malicious node cannot tamper with the information saved on the blockchain. Therefore, proposed scheme is resistant to the Sybil attack [27].

2.4.2. Forgery attack

The forgery attack may happen on the data-held edge server in proposed scheme. The malicious node tries to forge a valid without key k of $PRF()$ kept in the user. Due to the hash function's one-way property and the collision-free property, a malicious node can't forge a valid. So, proposed scheme is resistant to the forgery attack [28].

2.4.3. Replay attack

In proposed scheme, the malicious data-holder server tries to use the verified $hashTable$ for the next-time verification. This attack does not work to the proposed scheme because the secret string is changed after one verification. In addition, all the communication messages have embedded the timestamp. Therefore, proposed scheme is resistant to replay attack [29].

2.4.4. Collusion attack

In proposed scheme, the adversaries can hide the fact of data loss or modification. In the experimental part, we show that the verification rate is near 100%, even a part of the malicious node exists. So, proposed scheme is resistant to the collusion attack [30].

2.4.5. Man-in-the-middle attack

The man-in-the-middle attack occurs when two parties exchange their public keys. In proposed scheme, all the public keys are stored in the genesis block and cannot be tampered. Therefore, proposed scheme is resistant to the man-in-the-middle attack [31].

3. Evaluation

To verify the effectiveness of proposed scheme, we built up a prototype to evaluate its performance in a simulated cloud-edge environment.

3.1. Experiment environment and performance metrics

The simulation is implemented with Python 3.6 [32]. An SHA-256 hash function [33] is used for $H_1()$ and $H_2()$. β_1 and β_2 are 256. The cloud-edge healthcare system environment includes 7 hospital servers and a cloud server with 2 malicious hospital server. Each edge server is equipped with 2 vCPUs, 2 GB memory, and 40 GB storage. The cloud server has 8 vCPUs, 16 GB memory, and 100 GB storage.

Two distributed data integrity verification methods for cloud-edge computing are compared [19,22]. For schemes in [22], the authors verify the data integrity through a Merkle hash tree by sampling mode. For the scheme in [19], the authors hash the entire file to verify the data integrity. In the following part, scheme 1 and scheme 2 are used to represent these two kinds of methods. We use NumPy, a site package of Python, to generate the data file with a matrix structure. In NumPy, an element in the matrix is represented by 64 bits. The minimum data modification is 8 Bytes. The data varied form 128 MB to 2048 MB, which is sufficient for general healthcare system [34].

The following three metrics are used to evaluate schemes performance:

1. Success rate of corruptiong detection:
   We excute the scheme for 1000 times to evaluate the success rate.
2. Time consumption:
   The time consumption reflects the computing overhead, in which times for generating the tag generation and data verification are compared.
3. Communication overhead:
   Communication overhead is the size of transmitted data respectively for tag generation, verification, and consensus.

The parameter settins are shown in Table 2.

Table 2
Parameter settings.

Parameter	Varified value				
Data size (MB)	128	256	512	1024	2048
Data block size (KB)	128	256	512	1024	2048
Corruption size (Byte)	8		40	72	
Corruption ratio	0.001		0.004	0.016	

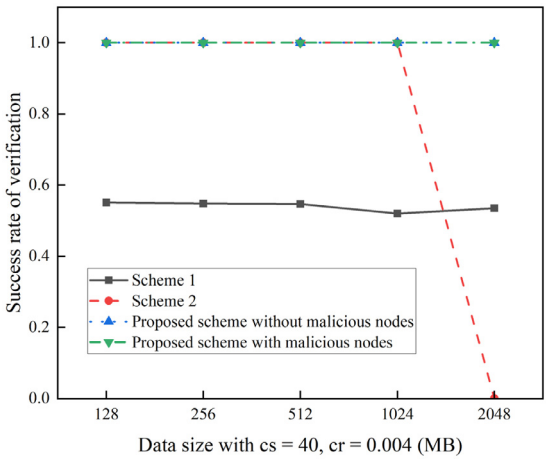


Fig. 2. Success rate of verification with data size varying.

3.2. Results of success verification rate

Figure 2 shows the success rate of verification achieved by 3 competing schemes with the data size varying. The corruption size is 40 Bytes, and the corruption ratio is 0.004. The proposed scheme is performed in situations with and without malicious nodes.

As shown in Fig. 2, the success verification rate of the three schemes does not change with the data size increasing. Due to the similar corruption size and ratio, the detection rate is the same.The verification success rate of scheme 1 is a little higher than 50% because of a 200-times sampling. Besides, scheme 2 and proposed schemes achieve a 100% success rate. The existence of malicious nodes does not result in adverse effects. When the data size is not more than 1024 MB, the hash operation for a whole data file can guarantee a 100% success rate for scheme 2. A hash operation on a 2 GB file reported a memory error because the memory of the experimental virtual machine is 2 GB, and the Ubuntu system occupies a part of the memory. Therefore, scheme2's experimental result is blank when the data file is 2 GB.

Figure 3 illustrates the impact of the corruption ratio (from 0.001 to 0.016) and corruption size (from 8 Bytes to 72 Bytes) on the success rate of verification with data size fixed at 512 MB. It can be seen that the corruption ratio impacts the success rate more significantly than the corruption size. An overall tendency is that the corruption size does not influence the success rate much for all schemes. Main reason is the one-way and collision-free properties of the hash function. For scheme 1, the success rate of verification increases when the corruption ratio increases. A higher corruption ratio makes it easier to sample the corruption data block. For scheme 2, the success rate is 100% due to the direct comparison of the hash value of the data file. For proposed schemes, the success rate is near 100% when the corruption ratio is 0.001. Because of malicious behavior hiding data loss or modification, the success rate of proposed scheme with the malicious nodes is a bit lower than that without malicious nodes. However, the rate is still over 90%.

3.3. Results of time consumption

Figure 4 (a) shows that the time consumption for generating a tag is increasing as the data size. The data block size increases, resulting in the time consumption of calculating the hash value. Mainly, due to the memory limitation, scheme 2 cannot be applied with a data size of 2 GB. From Fig. 4(a), the overall time consumption of proposed scheme is much lower than the other two schemes. In Fig. 4(b), because the verification process is reconstructing the Merkle hash root, the time consumption of verification for scheme 1 remains almost the same. For scheme 2, the verification process is recalculating the hash value of the entire data file. So, the time consumption is increasing with more data size.

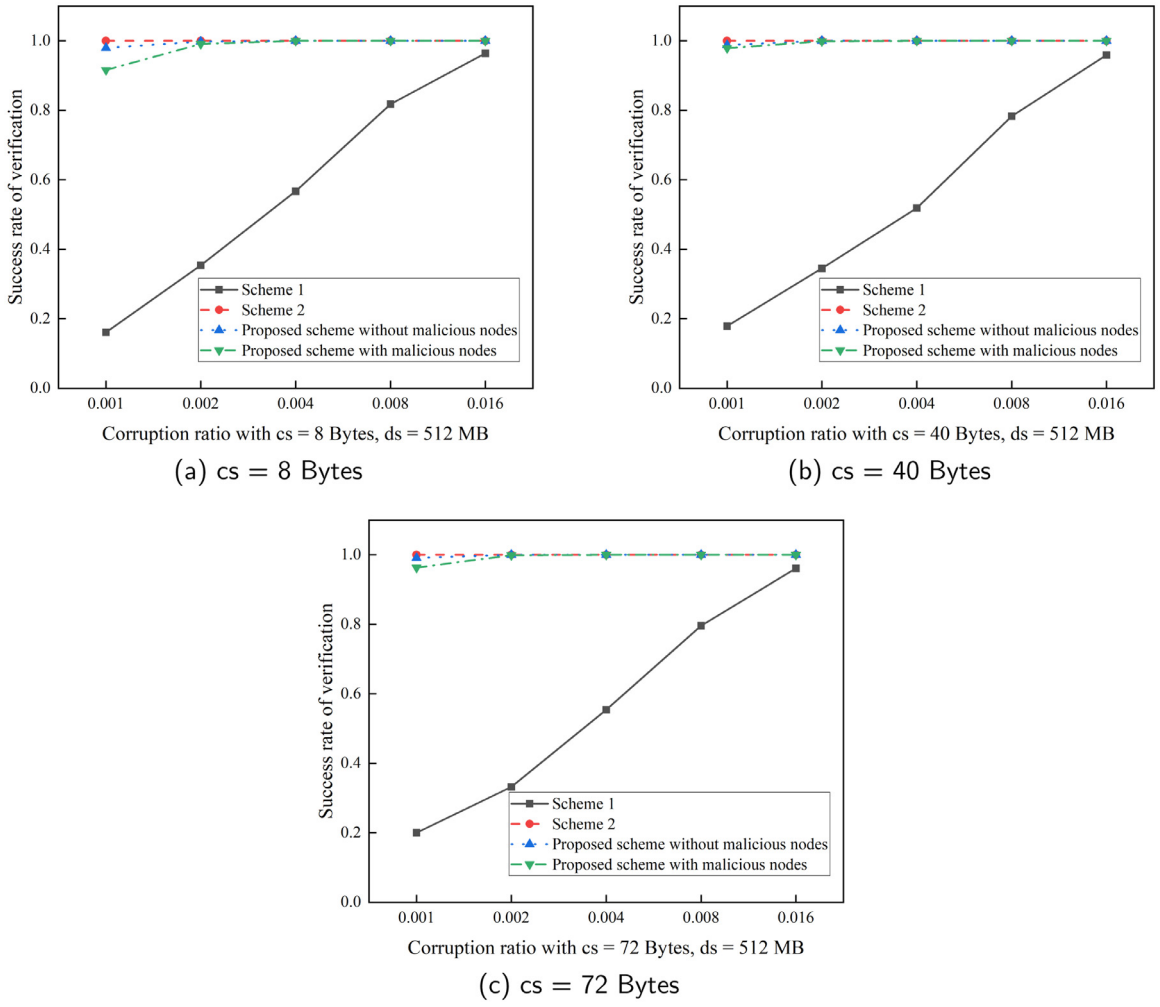


Fig. 3. Success rate of verification with corruption ratio varifying.

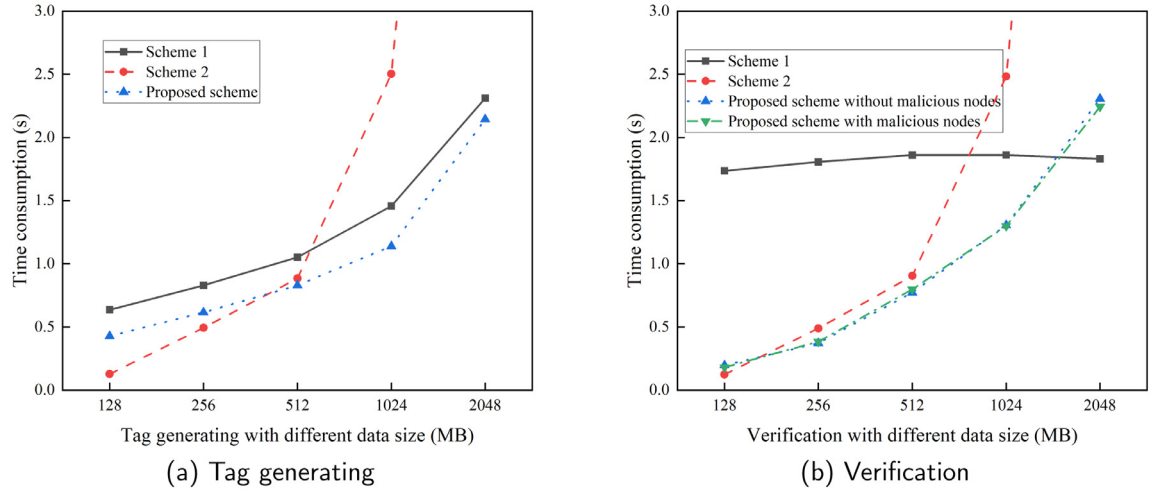


Fig. 4. Time consumption for different stage.

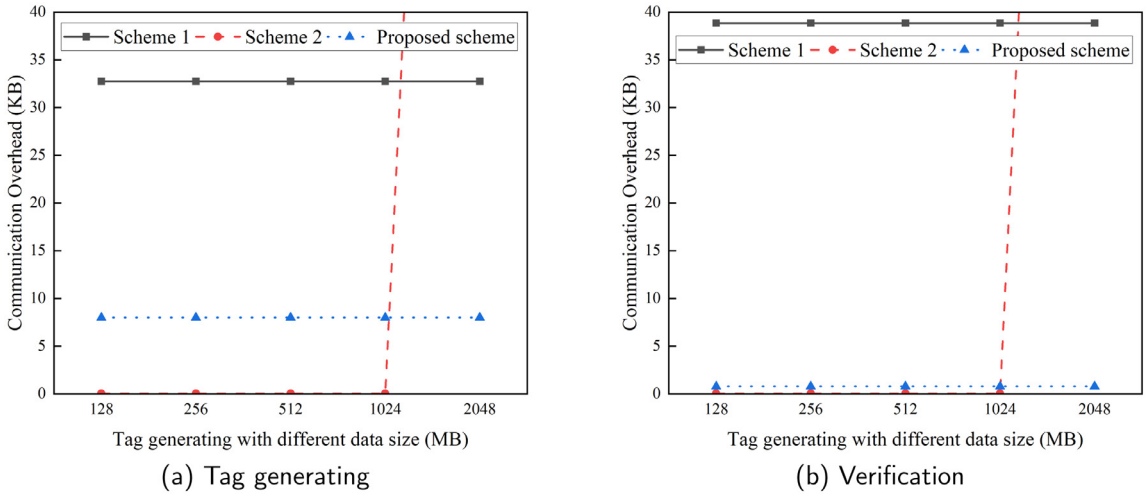


Fig. 5. Communication overhead for different stage.

Similarly, the result of scheme 2 with a 2 GB data file is blank due to the memory limitation.

3.4. Results of communication overhead

Figure 5 (a) shows that the communication overhead for tag generating is similar when data size varifies. 8-KB communication overhead is acceptable for proposed scheme. In Fig. 5(b), the communication overhead for verification of scheme 1 is much higher than scheme 2 and proposed scheme. The reason is that scheme 1 adopt a 200-times samplings. A communication overhead of less than 1KB for verification is acceptable for proposed scheme. It can also be seen from these two figures that the communication overhead of scheme 2 is lower than ours when the data size is less than 2 GB. The main reason that leads to these experimental results is that the tag information of scheme 1 is a 256-bits hash value, but proposed scheme uses 256 bits × 256 bits hash table. In addition, the communication overhead of scheme 1 is much higher than scheme 2 and ours due to the tag information of scheme 1 being a Merkle hash tree, whose space occupancy is much higher.

4. Conclusion

A blockchain-based scheme without a third-party auditor is proposed to accomplish data integrity auditing for the cloud-edge healthcare system. First, a distributed data integrity verification method is designed. A verification tag constructed by the hash table of the data file and secret string guarantees a verification rate close to 100%. Then, a specific description of the proposed blockchain-based scheme is given, consisting of the PoAF consensus protocol and related block construction. Finally, the proposed scheme is compared with two state-of-art schemes. The results show that our scheme can verify the data integrity effectively and efficiently for the cloud-edge healthcare system. In the future, we will further improve the distributed data auditing method and deploy it in a real blockchain environment.

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.

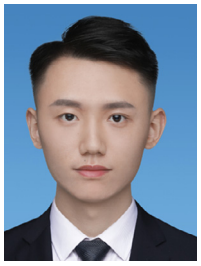
CRediT authorship contribution statement

Yi Li: Conceptualization, Methodology, Software. **Meiqin Tang:** Data curation, Writing – original draft.

References

[1] C. Lin, D. He, X. Huang, K.-K.R. Choo, OBFP: optimized blockchain-based fair payment for outsourcing computations in cloud computing, IEEE Trans. Inf. Forensics Secur. 16 (2021) 3241–3253.
[2] J. Shen, J. Shen, X. Chen, X. Huang, W. Susilo, An efficient public auditing protocol with novel dynamic structure for cloud data, IEEE Trans. Inf. Forensics Secur. 12 (10) (2017) 2402–2415.
[3] X. Jiang, F.R. Yu, T. Song, V.C.M. Leung, A survey on multi-access edge computing applied to video streaming: some research issues and challenges, IEEE Commun. Surv. Tutor. 23 (2) (2021) 871–903.
[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song, Provable data possession at untrusted stores, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007, pp. 598–609.
[5] A. Juels, B.S. Kaliski Jr., PORs: proofs of retrievability for large files, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007, pp. 584–597.
[6] M. Nofer, P. Gomber, O. Hinz, D. Schiereck, Blockchain, Bus. Inf. Syst. Eng. 59 (3) (2017) 183–187.
[7] K. JuelsA, Proofs of retrievability for large files, in: Proc of Computer and Communications Security, Vol. 584, AcMPress, Alexandria, 2007, p. 597.
[8] G. Ateniese, R. Di Pietro, L.V. Mancini, G. Tsudik, Scalable and efficient provable data possession, in: Proceedings of the 4th International Conference on Security and Privacy in Communication Networks, 2008, pp. 1–10.
[9] H. Shacham, B. Waters, Compact proofs of retrievability, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2008, pp. 90–107.
[10] C.C. Erway, A. Küpçü, C. Papamanthou, R. Tamassia, Dynamic provable data possession, ACM Trans. Inf. Syst. Secur.(TISSEC) 17 (4) (2015) 1–29.
[11] Q. Wang, C. Wang, J. Li, K. Ren, W. Lou, Enabling public verifiability and data dynamics for storage security in cloud computing, in: European Symposium on Research in Computer Security, Springer, 2009, pp. 355–370.
[12] J. Mao, Y. Zhang, P. Li, T. Li, Q. Wu, J. Liu, A position-aware Merkle tree for dynamic cloud data integrity verification, .Soft Comput 21 (8) (2017) 2151–2164.
[13] L. Rao, H. Zhang, T. Tu, Dynamic outsourced auditing services for cloud storage based on batch-leaves-authenticated Merkle hash tree, IEEE Trans. Serv. Comput. 13 (3) (2017) 451–463.
[14] K.D. Bowers, A. Juels, A. Oprea, Proofs of retrievability: theory and implementation, in: Proceedings of the 2009 ACM workshop on Cloud Computing Security, 2009, pp. 43–54.
[15] T.H. Yuen, PAChain: private, authenticated & auditable consortium blockchain and its implementation, Future Gen. Comput. Syst. 112 (2020) 913–929.
[16] X. Gao, J. Yu, W.-T. Shen, Y. Chang, S.-B. Zhang, M. Yang, B. Wu, Achieving low-entropy secure cloud data auditing with file and authenticator deduplication, Inf. Sci. 546 (2021) 177–191.
[17] Y. Xu, S. Sun, J. Cui, H. Zhong, Intrusion-resilient public cloud auditing scheme with authenticator update, Inf. Sci. 512 (2020) 616–628.
[18] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, Y. Yang, Auditing cache data integrity in the edge computing environment, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2020) 1210–1223.
[19] B. Li, Q. He, F. Chen, H. Dai, H. Jin, Y. Xiang, Y. Yang, Cooperative assurance of cache data integrity for mobile edge computing, IEEE Trans. Inf. Forensics Secur. 16 (2021) 4648–4662.
[20] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, Y. Yang, Inspecting edge data integrity with aggregated signature in distributed edge computing environment, IEEE Trans. Cloud Comput. (2021). early access

[21] G. Cui, Q. He, B. Li, X. Xia, F. Chen, H. Jin, Y. Xiang, Y. Yang, Efficient verification of edge data integrity in edge computing environment, IEEE Trans. Serv. Comput. (2021).
[22] D. Yue, R. Li, Y. Zhang, W. Tian, Y. Huang, Blockchain-based verification framework for data integrity in edge-cloud storage, J. Parallel Distrib. Comput. 146 (2020) 1–14.
[23] A. Gervais, G.O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, S. Capkun, On the security and performance of proof of work blockchains, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 3–16.
[24] G. Wood, et al., Ethereum: a secure decentralised generalised transaction ledger, Ethereum Project Yellow Paper 151 (2014) (2014) 1–32.
[25] Y. Fan, H. Wu, H.-Y. Paik, DR-BFT: a consensus algorithm for blockchain-based multi-layer data integrity framework in dynamic edge computing system, Future Gen. Comput. Syst. 124 (2021) 33–48.
[26] M. Szydlo, Merkle tree traversal in log space and time, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 541–554.
[27] J.R. Douceur, The sybil attack, in: International Workshop on Peer-to-Peer Systems, Springer, 2002, pp. 251–260.
[28] N. Jovanovic, E. Kirda, C. Kruegel, Preventing cross site request forgery attacks, in: 2006 Securecomm and Workshops, IEEE, 2006, pp. 1–10.
[29] R. Pries, W. Yu, X. Fu, W. Zhao, A new replay attack against anonymous communication networks, in: 2008 IEEE International Conference on Communications, IEEE, 2008, pp. 1578–1582.
[30] B. Kannhavong, H. Nakayama, N. Kato, Y. Nemoto, A. Jamalipour, NIS01-2: A collusion attack against OLSR-based mobile ad hoc networks, in: IEEE Globecom 2006, IEEE, 2006, pp. 1–5.
[31] F. Callegati, W. Cerroni, M. Ramilli, Man-in-the-middle attack to the HTTPS protocol, IEEE Secur. Privacy 7 (1) (2009) 78–81.
[32] G. vanRossum, Python reference manual, Department of Computer Science [CS] R 9525, 1995.
[33] S. Gueron, S. Johnson, J. Walker, Sha-512/256, in: 2011 Eighth International Conference on Information Technology: New Generations, IEEE, 2011, pp. 354–358.
[34] M.K. Hassan, A.I. El Desouky, S.M. Elghamrawy, A.M. Sarhan, Big data challenges and opportunities in healthcare informatics and smart hospitals, in: Security in Smart Cities: Models, Applications, and Challenges, 2019, pp. 3–26.



Yi Li is a PhD student in the School of Computer Science at Nanjing University of Information Science and Technology, Nanjing, China. His current research interests include information security, edge computing, data auditing, and blockchain.

Meiqin Tang is working at Internet of Things Technology College at Wuxi Vocational College of Science and Technology. Her current research interests include Internet of Things, blockchain, and information security.