Hindawi
Security and Communication Networks
Volume 2022, Article ID 3186112, 8 pages
https://doi.org/10.1155/2022/3186112

Research Article

Blockchain-Based Proof of Retrievability Scheme

Yan Ren,[1] **Haipeng Guan**,[1] **Qiuxia Zhao**,[1] **and Zongxiang Yi**[2,3,4]

[1]*School of Mathematics and Information Technology, Yuncheng University, Yuncheng 044000, China*
[2]*School of Mathematics and Systems Science, Guangdong Polytechnic Normal University, Guangzhou 510665, China*
[3]*Guangdong Provincial Key Laboratory of Information Security Technology, Guangzhou 510006, China*
[4]*School of Mathematics and Information Science, Guangzhou University, Guangzhou 510006, China*

Correspondence should be addressed to Zongxiang Yi; tpu01yzx@gmail.com

Received 17 October 2021; Revised 15 November 2021; Accepted 24 December 2021; Published 3 February 2022

Academic Editor: Yuling Chen

In the internet of things, user information is usually collected by all kinds of smart devices. The collected user information is stored in the cloud storage, and there is a risk of information leakage. In order to protect the security and the privacy of user information, the user and cloud provider will periodically execute a protocol called proof of retrievability scheme. A proof of retrievability scheme ensures the security of the data by generating proof to convince the user that the cloud provider does correctly store the user information. In this paper, we construct a proof of retrievability scheme using the blockchain technology. Using the advantage that the stored data cannot be tampered with in blockchain, this ensures the integrity of the data. Specifically, some related definitions, security models, and a blockchain-based construction of a proof of retrievability scheme are given. Then the validity and security of the scheme are proved later. As a result, user information can be protected by our scheme.

1. Introduction

1.1. Background. With the information systems coming into our life, there are many user private information appliances such as surveillance cameras, smartwatches, smart door locks, and the online supermarket. They provide a lot of convenience for our life. However, these providers will collect user information and store it in the cloud where new technologies are widely used [1–7]. Due to the vulnerability of the cloud, user information could be attacked by hackers in information systems and can be easily stolen if the cloud storage provider is compromised. Among the problems and challenges of cloud storage [8–10], only the problem of how to ensure the security and integrity of the information is considered in the paper. In order to solve it, three kinds of methods are used [11]: proof of ownership (PoW), provable data possession (PDP), and proof of retrievability (PoR). We focus on the PoR and for the state-of-the-art of PoR, the reader is referred to [11–25].

Generally, the schemes of PoR are under different settings and security models. On the one hand, some schemes [11–14] are for static data. Some schemes [15–17] discussed the multiserver setting. In these schemes, the client can identify machines and recover the data from the others by using the audit mechanism. Other schemes [18–25] are for dynamic data. On the other hand, works in [18–21] are about security. The authors of [22–24] researched on memory checking and study how to authenticate remotely stored dynamic data. The scheme in [25] is for the multiserver and dynamic data setting.

Recently, blockchain is used to eliminate a trusted third party in many protocols [26]. However, it is still unknown how to utilize blockchain in PoR schemes, which is also a new challenge in constructing a PoR scheme.

1.2. Motivation and Contribution. The concept of blockchain was first proposed in 2008 in "Bitcoin: a peer-to-peer electronic cash system" [27] published by the cryptography mailing group by a scholar known by the pseudonym "Satoshi Nakamoto." The verification, bookkeeping, storage, maintenance, and transmission of the data in blockchain are all based on the distributed system structure, and the trust relationship between distributed nodes is established by the

pure mathematical method instead of the central mechanism. Thus, a decentralized and reliable distributed system can be formed. The goal of blockchain is to provide trusty for transactions between untrusted entities, without the need for a trusted third party. At present, many institutions have combined the industry conditions with the characteristics of blockchain and made beneficial attempts in many industries, including payment, Internet of things, credit investigation, transaction settlement and clearing, crowdfunding, equity transaction, audit, supply chain, digital asset management, notarization, and other fields [28–33]. We consider using blockchain technology to solve the problem of the trusted third party in the verification of the PoR scheme.

In this paper, we first define a security model for the blockchain-based proof of retrievability by modifying the model in [14, 34, 35]. Secondly, we propose the first concrete PoR scheme based on blockchain. Finally, we demonstrate that the proposed scheme is provably secure in the new model.

1.3. Organization. The rest of the paper is organized as follows. Preliminaries are given in Section 2. In Section 3, we formally define the framework and security model for blockchain-based PoR schemes. Then a concrete construction of a blockchain-based scheme is presented in Section 4. We analyze the security of the proposed scheme in Section 5. Finally, conclusions are made in Section 6.

2. Preliminaries

In this section, some notions are introduced such as hash function, Merkle tree, blockchain, and bilinear pairing.

2.1. Hash Function. The hash function H is used to map data x of an arbitrary length (input) to data $y = H(x)$ of fixed length (output). y is called the hash of x. Many Hash functions [36] are widely publicly available and can be selected based on the context.

$$H\colon \{0,1\}^* \longrightarrow \{0,1\}^n. \tag{1}$$

This transformation is a compression mapping, which has the following properties:

(i) The space of the hash value is usually much smaller than the space of the input.

(ii) Different inputs may hash into the same output, but it is hard to find two different inputs x, x' such that $H(x) = H(x')$.

(iii) It is infeasible to determine the input value x from the hash value y.

Assumption 1 (hash function preimage assumption). Given $y = H(x)$, it is hard to compute x.

Assumption 2 (hash function collision assumption). Given x, it is hard to compute x' such that $H(x) = H(x')$.

2.2. Merkle Tree. Merkle tree, also known as a Hash tree, as the name implies, is a tree that stores hash values. A leaf node of a Merkle tree is attached to the hash value for a data block. A nonleaf node is attached to the cryptographic hash of its corresponding child nodes.

Figure 1 presents a simple example of a Merkle tree with 4 pieces of data. Let f be a hash function and $X = \{x_0, x_1, x_2, x_3\}$ denotes the set of data used to generate the Merkle tree. A Merkel tree is generated as follows: firstly, for all leaf nodes, $y_{\mathrm{bin}(i)} = f(x_i)$ where $i = 1, 2, 3, 4$ and $\mathrm{bin}(i)$ is the binary form of i; secondly, for all inside nodes, the value of the node is $f(y_l \| y_r)$ where y_l and y_r are the value of left child and right child, respectively. An Merkle tree is **valid** if and only if the value of each inside node equals to $f(y_l \| y_r)$. As a result, this example outputs the following:

$$\begin{aligned} &y_{0,0} = f(x_0), y_{0,1} = f(x_1), y_{1,0} = f(x_2), y_{1,1} = f(x_3),\\ &y_0 = f\left(y_{0,0} \| y_{0,1}\right), y_1 = f\left(y_{1,0} \| y_{1,1}\right),\\ &y = f(y_0 \| y_1). \end{aligned} \tag{2}$$

In a Merkle tree, the value of the root node is called the hash of the Merkle tree. For the example in Figure 1, the hash of that tree with data X is

$$y = f(f(f(x_0) \| f(x_1)) \| f(f(x_2) \| f(x_3))). \tag{3}$$

In the rest of this paper, we use $\mathrm{Merkel}(X)$ to denote the Merkle tree created by the data set X and use $H(T)$ to denote the hash of a Merkle tree T, where H is the underlying hash function. For example, the hash of the Merkle tree created by the data set X can be denoted by $H(\mathrm{Merkel}(X))$.

2.3. Blockchain. Within a blockchain, the hash function is used to determine the state of the blockchain and Figure 2 shows the structure of blockchain which can be viewed as a linked list of blocks. Every block has four basic objects: the hash of the previous block, the timestamp of generation, the random number of security, and the hash of a Merkle tree. Usually, the corresponding Merkel tree is linked with the block too. Two neighbor blocks are linked by a hash pointer that points from the previous block and thus it creates a chain of connected blocks, hence the name blockchain. By linking blocks in this manner, the ordered hashes of all the n blocks represent the entire state of the blockchain, namely,

$$f(f)(\mathrm{Block}(0) \| f(\mathrm{Block}(1)) \| \cdots \| f(\mathrm{Block}(n))), \tag{4}$$

where f is a hash function. A blockchain is **valid** if $f(\mathrm{Block}(i-1))$ equal to the value of the field *hash of* $\mathrm{Block}(i-1)$ in the structure of the block $\mathrm{Block}(i)$, for all $1 \le i \le n$.

To utilize blockchain for a data set X (see the example in Subsection 2.2), a corresponding Merkel tree T will be constructed by the data set X. Then a new block B denoted by $B(X)$ can be generated with the help of a timestamp provider. Adding more parameters, we use $B(X; ts)$ to denote a block where X is the data set to generate the hash of the Merkle tree, ts is the timestamp of the current time, and

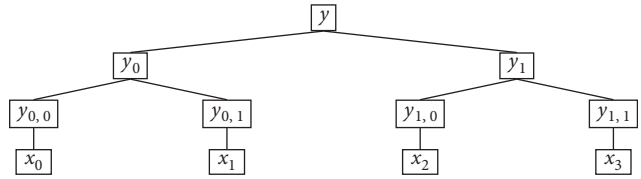

Figure 1: The structure of the Merkle tree.

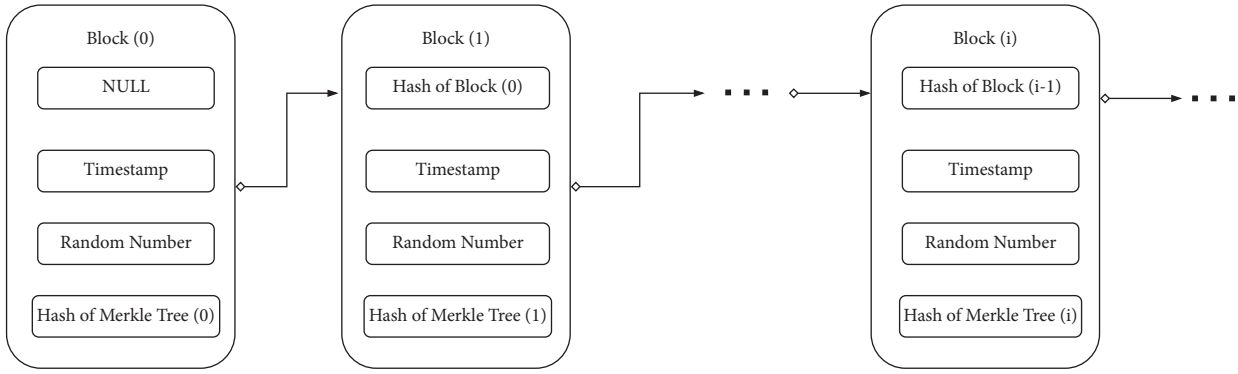

Figure 2: The structure of blockchain.

rand is the random number. Moreover, a blockchain provides the following operations:

(i) NewBlock(X;ts): create a valid block $B(X; ts)$.

(ii) AppendBlock(B): append the block B to the blockchain by filling a suitable random number in the block.

(iii) FetchBlock(ts): return the block at a time ts in the blockchain. If there are no blocks at that time, then *NULL* is returned.

Recently, there are issues in maintaining a blockchain, such as generating blocks [37, 38] and updating with efficiency [39]. Anyway, to summarize the characteristic of blockchain, we have the following assumption.

Assumption 3 (blockchain assumption). All the state and blocks of blockchain is hard to modify after they were generated.

2.4. Bilinear Pairing. Bilinear pairing is also called bilinear mapping, which was first used to construct tripartite key exchange protocol [40]. It involves three multiplicative cyclic groups G_1, G_2, and G_T which have a prime order p. Bilinear pairing is a mapping e: $G_1 \times G_2 \longrightarrow G_T$ satisfying the following conditions:

(1) For any $g_1 \in G_1, g_2 \in G_2$, and $a, b \in \mathbb{Z}_p$, it always has $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$

(2) There exists two elements $g_1 \in G_1$ and $g_2 \in G_2$ such that $e(g_1, g_2) \neq 1_{G_T}$ where 1_{G_T} is the identity in G_T

(3) For any $g_1 \in G_1$, $g_2 \in G_2$, it is feasible to compute $e(g_1, g_2)$

Let $c_1, c_2, c_3 \in \mathbb{Z}_p$ and g_1, g_2, g be the generators of G_1, G_2, G_T, respectively. There are two security assumptions related to bilinear pairing.

Assumption 4 (bilinear decisional Diffie–Hellman). Given a bilinear pairing e, $g_1^{c_1} \in G_1, g_2^{c_2} \in G_2, g^{c_3} \in G_T, e(g_1, g_2)^{c_1 c_2 c_3}$ and a randomly selected element $T \in G_T$, it is hard to distinguish $e(g_1, g_2)^{c_1 c_2 c_3}$ from T.

Assumption 5 (bilinear computational Diffie–Hellman). Given a bilinear pairing e, $g_1^{c_1} \in G_1, g_2^{c_2} \in G_2, g^{c_3} \in G_T$, it is hard to compute $e(g_1, g_2)^{c_1 c_2 c_3}$.

3. Security Model

3.1. System Setting. Our system has three entities, the user, the cloud storage provider where user information is stored, and a blockchain where several timestamp providers are available to all entities. The structure of the system setting is shown in Figure 3.

(i) *The User.* The user is the entity who wants to store the data on the cloud storage. Whenever the user wants to check whether the data is correctly stored on the cloud storage, then a request of PoR will be generated and sent to the cloud storage. With the help of blockchain, the user can verify the retrievability of stored data by the proof received from the cloud storage provider.

(ii) *The Cloud Storage Provider.* A Cloud storage provider is an entity who exactly stores the data for the user. Besides, the cloud storage provider generates and sends the proof of retrievability after receiving the request from the user.

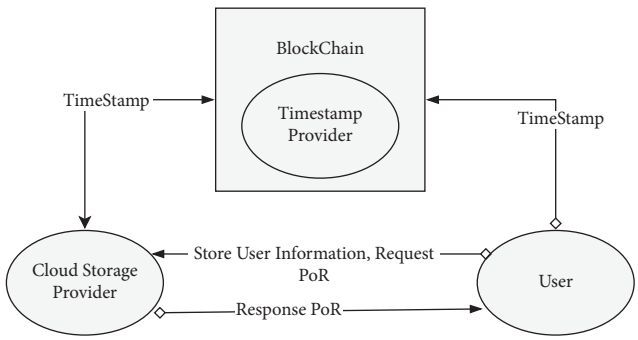

Figure 3: System setting.

(iii) *BlockChain.* BlockChain is mainly for keeping the transcripts of PoR scheme constant. Moreover, timestamp providers in a blockchain can help the cloud storage providers to generate PoR and the user to verify the generated PoR.

3.2. Timestamp Usage. The timestamps are provided to both the user and the cloud storage provider. The existence of the data is guaranteed by timestamp through computing the hash value which is included in the next timestamp. In our scheme, we will modify the traditional timestamp computation. At the end of every proof generation, the timestamp provider proceeds to compute a timestamp on the current time and makes the timestamp published on the blockchain. The timestamp is used to compute the hash value in blockchain by the cloud storage provider.

We benefit the security from the usage of timestamps. On the one hand, running a PoR scheme twice at two different moments would be the PoR for the duration between the two moments. On the other hand, it gives a timeline of PoR records which can be used to analyze the efficiency.

3.3. Definition. There are five algorithms in the blockchain-based PoR which are described as follows:

(i) Keygen: The input of the algorithm is the security parameter, and the output is the public key and private key of the system and the user.

(ii) Outsource: In this stage, it inputs the private key and user data M, and outputs a data set Y with n blocks and one tag σ for each block. For the blockchain, also generates new blocks for the data.

(iii) RequestChallenge: The user randomly selects a challenge r and sends it to the cloud storage provider.

(iv) ResponseProof: The proof process is an interactive protocol. The input is a public key, the file name and tag of the file and the output is *proof* for a proof response.

(v) VerifyProof: The input is a system parameter and *proof*, the output of the algorithm is *accepted* or *rejected*.

Remark 1. Note that system parameter includes the structure and the state of selected blockchain, as well as another luxury public information such as the hash function implementations and bilinear pairing implementations.

3.4. Security Model. Under the assumptions mentioned in Section 2, a blockchain-based PoR scheme is secure if it satisfies the following two properties.

(1) *Correctness.* If all the effective proofs generated by the algorithm (*KeyGen*, *outsourcing*, *Request Challenge*, *Response Proof*, *and Verify Proof*) are defined above, the verification algorithm outputs accept, then a blockchain-based PoR scheme is correct.

(2) *Reasonableness.* For reasonableness, if any malicious cloud storage provider can generate proof such that the *Verify Proof* outputs *accept*. That is, the user believes that the cloud storage provider can generate the proof only if it correctly stores the user data.

If the probability that an adversary with arbitrary probabilistic polynomial-time wins the game described below is negligible, then a blockchain-based PoR scheme is reasonableness.

(a) Setup: The challenger runs the *Keygen* algorithm to obtain the public key and private. Then the public key is sent to the adversary.

(b) Outsource: The adversary selects a data set and sends it to the challenger, who runs the *Outsourcing* algorithm and responds with the output.

(c) ChallengeProof:

(1) In the *Request Challenge* algorithm, the challenger randomly generates a challenge message and sends it to the adversary.

(2) The adversary generates a data set first by running an arbitrary algorithm that returns a proof. The proof will be sent to the challenger in the *Response Proof* algorithm.

(d) Verify: The challenger runs the *VerifyProof* algorithm to verify the proof received from the adversary. It outputs *accept* if and only if the proof is accepted by the challenger.

The adversary wins the game if *accept* is outputted in the last *Verify* step.

4. Our PoR Scheme

4.1. High Description. In this section, we will propose a blockchain-based PoR scheme. To cut costs, the cloud storage provider only needs to generate a Merkle tree for a data set and store the hash of the Merkle tree in the blockchain. The data set can be stored anywhere by the cloud storage provider. When the user requests a challenge of PoR, the cloud storage provider fetches back the Merkle tree and generates a PoR to the user with the help of blockchain.

4.2. Blockchain-Based Proof of Retrievability Scheme. Our scheme consists of five algorithms, namely, *Keygen*, *Outsource*, *RequestChallenge*, *ResponseProof,* and *VerifyProof*.

4.2.1. Keygen. Both the user and the cloud storage provider make consensus on public system parameters: a hash function H, a blockchain BC, a block size t, a prime number p, a generator g of the cyclic multiplicative group $(\mathbb{Z}_p, \times)$, and a bilinear pairing e on $\mathbb{Z}_p$.

The user chooses a nonzero element $s \in \mathbb{Z}_p$ randomly as a private key and computes and publics $g^s \in \mathbb{Z}_p$ as a public key.

4.2.2. Outsource. When a user wants to store a file on the cloud storage, the interactive algorithm is run between them.

(1) Given a data set $X = \{x_1, x_2, \ldots, x_m\}$, the user uses an error correction code to get the encoded data Y. In the case that some blocks $Y' \subset Y$ may be lost by the cloud storage, an error correction code is used to reconstruct the original data set X [41].

(2) Divide the encoded data Y into n blocks, $Y = \{y_1, y_2, \ldots, y_n\}$, where $y_i \in \{0,1\}^t$.

(3) For each data block y_i, the user computes the authentication tag σ_i as follows:

(i) Randomly choose a nonzero element $r_i \in Z_p^*$ called block nonce.

(ii) $\sigma_i = y_i^{H(r_i \| i)}$

(4) The user outsources Y and $\Sigma = \{\sigma_i | 1 \le i \le n\}$ to the storage server.

(5) The cloud storage provider creates a Merkle tree $\mathrm{Merkel}(\Sigma)$ by Σ, and stores the hash of the Merkle tree $H(\mathrm{Merkel}(\Sigma))$ into the blockchain by doing an operation $\mathrm{AppendBlock}(\mathrm{Merkel}(\Sigma); ts, \mathrm{NULL})$.

Remark 2. When we compute $y_i^{H(r_i \| i)}$, y_i and $H(r_i \| i)$ are treated as a big integer number.

4.2.3. RequestChallenge. To verify that the provider has stored the data correctly, the user randomly selects an integer $1 \le k \le n$ indicating which block should be checked. Then k and r_k are sent to the provider for requesting challenge.

4.2.4. ResponseProof. For the cloud storage provider, there are n blocks of data and the k-th block is requested to be checked. Now when the provider receives a request challenge, a PoR can be generated as follows:

(1) Randomly select a nonzero element $x \in Z_p$.

(2) Compute $\sigma = \sigma_k^x$ and g^x.

(3) Fetch out Y and Σ from storage devices to retain the hash of the Merkle tree $\mathrm{Merkel}(\Sigma)$.

(4) Send $H(\mathrm{Merkel}(\Sigma))$ to the timestamp provider in the blockchain.

(5) The timestamp provider verifies that $H(\mathrm{Merkel}(\Sigma))$ is valid when received it. If it is valid, then a timestamp ts is generated to run

$$\mathrm{AppendBlock}(\mathrm{NewBlock}(\mathrm{Merkel}(\Sigma), ts)), \tag{5}$$

and is sent back to the cloud storage provider. Otherwise, the algorithm is terminated.

(6) The cloud storage provider generates the proof

$$\mathrm{proof}_k = (\sigma, \sigma_k, g^x, ts, H_1, H_2), \tag{6}$$

where $H_1 = H(\mathrm{Merkel}(\Sigma))$ and $H_2 = H(ts \| \mathrm{Merkel}(\Sigma))$. Then proof_k is sent back to the user.

4.2.5. VerifyProof. After receiving the proof, the user does the following operations in order:

(1) Send ts to the timestamp provider in the blockchain. If no *accept* is returned, then the algorithm is terminated with a *reject*.

(2) If the blockchain is invalid (See Section 2.3), then the algorithm is terminated with a *reject*.

(3) Run $\mathrm{FetchBlock}(ts)$ to obtain the corresponding Merkle tree $\mathrm{Merkel}(\Sigma)$ and the hash H_0 of that Merkle tree from the blockchain.

(4) If $H_0 \neq H(\mathrm{Merkel}(\Sigma))$, then the algorithm is terminated with a *reject*.

(5) If $\mathrm{Merkel}(\Sigma)$ is invalid (See Section 2.2), then the algorithm is terminated with a *reject*.

(6) If $H(\sigma_k)$ does not equal the value of the corresponding leaf node in the Merkle tree, then the algorithm is terminated with a *reject*.

(7) If $e(\sigma, g^s) \neq e(\sigma_k^s, g^x)$, then the algorithm is terminated with a *reject*.

(8) If $H_1 \neq H(\mathrm{Merkel}(\Sigma))$, then the algorithm is terminated with a *reject*.

(9) If $H_2 \neq H(ts \| \mathrm{Merkel}(\Sigma))$, then the algorithm is terminated with a *reject*.

(10) Return *accept*.

Remark 3. Firstly, the above operations first check that the blockchain (without Merkle trees) and the Merkle tree related to the last block are valid. Secondly, the existence of the k-th block is checked.

5. Security Analysis

5.1. Correctness

Theorem 1. *The verify process is correct. It means that*

$$e(\sigma, g^s) = e(\sigma_k^s, g^x), \tag{7}$$

holds where $\sigma = \sigma_k^x$.

Proof. It follows from the property of bilinear pairing that

$$e(\sigma, g^s) = e(\sigma_k^x, g^s) = e(\sigma_k, g)^{sx} = e(\sigma_k^s, g^x). \quad (8)$$

□

Remark 4. Due to Assumptions 4 and 5, the private key s is still secure even the result of bilinear pairing computation is public.

5.2. Reasonableness

Theorem 2. *If the cloud storage provider is honest, the final proof must be*

$$\text{proof}_k = (\sigma, \sigma_k, g^x, ts, H_1, H_2), \quad (9)$$

where $H_1 = H(Merkel(\Sigma))$, $H_2 = H(ts \| Merkel(\Sigma))$ *and* ts *is the current timestamp.*

Proof. If the cloud storage provider is honest, the following points hold true:

(i) σ and σ_k guarantee that at least the cloud storage provider stores the k-th block which is not revealed to the public in the bilinear pairing computation (See Remark 5.1).

(ii) The Merkle tree is created by the cloud storage provider at the time ts was required, and the leaf nodes of the tree are all part of the data set Y. It follows from Assumptions 1 and 2 that these hashes cannot be found without knowing the original data set Y.

(iii) ts generated by blockchain is trusted according to Assumption 3.

(iv) The consistency of the Merkle tree and timestamp are assured by H_1 and H_2, respectively.

To sum up, the cloud storage provider must store the data set correctly if *VerifyProof* return is *accepted*. □

5.3. Traceability

Theorem 3. *The blockchain-based PoR scheme in Section 4 is traceable.*

Proof. If the cloud server is dishonest, that is, the server modifies, deletes, or tampers with a piece of file without authorization of the user, S cannot compute the value of the root node correctly, so it cannot prove that he has completely stored the data. By verifying the Merkle tree, it will get which piece of file S has been modified finally.

For example, to verify whether the fifth block file has been modified, the following procedure can be followed and the structure as shown in Figure 4:

(i) *Verify Node 1.* Verify that the calculated value of node 1 is correct through the values of node 2 and node 3.

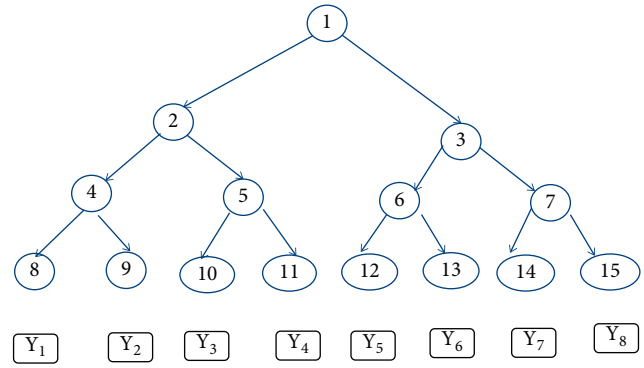

Figure 4: The example of traceability.

(ii) *Verify the Value of Node 3.* The receiver computes the value of node 3 through the values of node 6 and node 7 that he has received and verifies whether the calculated value of node 3 is correct.

(iii) *Compute the Value of Node 6.* The receiver computes the value of node 6 through the values of node 12 and node 13 that he has received and verifies whether the calculated value of node 6 is correct.

(iv) The receiver computes the value of node 12 from the value of Y_5 and verifies that the calculated value of node 6 is correct.

The correct value can be determined by whether the value of node 6 is consistent. This allows you to track down blocks of files that have been modified. □

5.4. Resistance to Two Kinds of Attacks. In this subsection, two kinds of attacks are considered, i.e., replay attacks and collusion attacks.

5.4.1. Resistance to Replay Attack. In Section 4.2.4, note that there is a timestamp ts attached to the proof_k, where

$$\text{proof}_k = (\sigma, \sigma_k, g^x, ts, H(\text{Merkel}(\Sigma)), \quad H(ts \| \text{Merkel}(\Sigma))). \quad (10)$$

(i) If a data storage provider uses an old timestamp ts, then it would be *rejected* in the first step (1) in Section 4.2.5 since the timestamp provider can easily find that such ts is expired. In other words, such ts may be valid in a short time. However, the user could not run this protocol twice in such a short time.

(ii) If a data storage provider uses an old proof proof_k, then it would be *rejected* in the seventh step (7) in Section 4.2.5 since g^x is attached with a challenge x that is randomly generated by the user. x should be different in two runs of this protocol.

In a word, our protocol is resistant to replay attacks with old timestamps ts or old proof proof_k.

5.4.2. Resistance to Collusion Attack. If we consider the case that the timestamp provider (and by extension the

blockchain provider) colludes with the data storage provider, then, in other words, the data storage provider would also play as a timestamp provider in the blockchain context. However, due to the security analysis of blockchain [42, 43], such malicious timestamp providers could be detected by the nodes in the blockchain network. Under Assumption 3 (BlockChain assumption), our protocol is resistant to such collusion attacks which can be reduced to an attack in a blockchain context.

6. Conclusion

In order to protect the security and integrity of user data, we formally defined a novel security model for a blockchain-based PoR scheme and proposed a secure scheme under the defined security model. The properties of the PoR scheme and the characteristics of blockchain, ensure the security and the integrity of data, respectively. Furthermore, we prove the correctness and reasonableness of our scheme. Our scheme makes user data more secure. In our scheme, blockchain plays an irreplaceable role in the privacy and security of user data. It is believed that as a blockchain improves the PoR scheme, it will continue to promote the progress of technology.

However, there are still many attacks not being considered, such as reset attacks and malicious attacks. To improve the performance, it is interesting to remove the bilinear mapping while reserving the same security level.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest in this work.

Acknowledgments

This work was supported by the Higher Education Technology Innovation Projects Foundation of Shanxi (Grant nos. 2021L467and 2020L0560), National Statistical Science Research Program of China(Grant no. 2021LY047), Research Project of Yuncheng University(Grant nos. YQ-2020020 and XK-2020036), Key Discipline Project of Yuncheng University, Young Innovative Talents Project of General Colleges and Universities in Guangdong Province (Grant no. 2019KQNCX112), Talent Special Project of Research Project of Guangdong Polytechnic Normal University (Grant no. 2021SDKYA051), Opening Project of Guangdong Provincial Key Laboratory of Information Security Technology (Grant no. 2020B1212060078), and the Guangdong Basic and Applied Basic Research Foundation (Grant no. 2021A1515011954).

References

[1] M. Armbrust, A. Fox, R. Griffith et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] R. Buyya, C. Shin Yeo, S. Venugopal, B. James, and I. Brandic, "Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[3] L. M. Kaufman, "Data security in the world of cloud computing," *IEEE Security & Privacy*, vol. 7, no. 4, pp. 61–64, 2009.

[4] R. L. Grossman, "The case for cloud computing," *IT professional*, vol. 11, no. 2, pp. 23–27, 2009.

[5] C. Stergiou, K. E. Psannis, B. G. Kim, and B. Gupta, "Secure integration of iot and cloud computing," *Future Generation Computer Systems*, vol. 78, pp. 964–975, 2018.

[6] A. Mishra, N. Gupta, and B. B. Gupta, "Defense mechanisms against ddos attack based on entropy in sdn-cloud using pox controller," *Telecommunication Systems*, vol. 77, pp. 1–16, 2021.

[7] Y. Chen, J. Sun, Y. Yang, T. Li, X. Niu, and H. Zhou, "Psspr: a source location privacy protection scheme based on sector phantom routing in wsns," *International Journal of Intelligent Systems*, vol. 37, no. 2, 2021.

[8] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 583–592, 2012.

[9] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[10] Y. Jadeja and K. Modi, "Cloud computing-concepts, architecture and challenges," in *Proceedings of theInternational Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, pp. 877–880, Nagercoil, India, March 2012.

[11] C. B. Tan, M. H. A. Hijazi, Y. Lim, and A. Gani, "A survey on proof of retrievability for cloud data integrity and availability: cloud storage state-of-the-art, issues, solutions and future trends," *Journal of Network and Computer Applications*, vol. 110, pp. 75–86, 2018.

[12] L. Jin, X. Tan, X. Chen, D. S. Wong, and F. Xhafa, "Opor: enabling proof of retrievability in cloud computing with resource-constrained devices," *IEEE Transactions on cloud computing*, vol. 3, no. 2, pp. 195–205, 2014.

[13] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Proceedings of theTheory of Cryptography Conference*, pp. 109–127, CA, USA, March 2009.

[14] H. Shacham and B. Waters, "Compact proofs of retrievability," *Journal of Cryptology*, vol. 26, no. 3, pp. 442–483, 2013.

[15] K. D. Bowers, A. Juels, and A. Oprea, "Hail: a high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 187–198, IL, USA, November 2009.

[16] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pp. 31–42, IL, USA, October 2010.

[17] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "Mr-pdp: multiple-replica provable data possession," in *Proceedings of the 28th international conference on distributed computing systems*, pp. 411–420, Beijing, China, July 2008.

[18] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, Istanbul Turkey, September 2008.

[19] M. Bellare and O. Goldreich, "On defining proofs of knowledge," in *Proceedings of theAnnual International Cryptology Conference*, pp. 390–420, August 1992.

[20] Q. Wang, C. Wang, L. Jin, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proceedings of the European symposium on research in computer security*, pp. 355–370, Saint-Malo, France, September 2009.

[21] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor, "Checking the correctness of memories," *Algorithmica*, vol. 12, no. 2-3, pp. 225–244, 1994.

[22] A. Fu, Y. Li, S. Yu, Y. Yan, and G. Zhang, "Dipor: an ida-based dynamic proof of retrievability scheme for cloud storage systems," *Journal of Network and Computer Applications*, vol. 104, pp. 97–106, 2018.

[23] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," *Journal of Cryptology*, vol. 30, no. 1, pp. 22–57, 2017.

[24] M. Etemad and A. Küpçü, "Transparent, distributed, and replicated dynamic provable data possession," in *Proceedings of the International Conference on Applied Cryptography and Network Security*, Saint-Malo, France, June 2013.

[25] E. Stefanov, M. V. Dijk, A. Juels, and A. Oprea, "Iris: A scalable cloud file system with efficient integrity checks," in *Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 229–238, Orlando, FL, USA, January 2011.

[26] E. Staff, "Blockchains: the great chain of being sure about things," *Economist*, vol. 18, no. 7, 2016.

[27] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," *Decentralized Business Review*, Article ID 21260, 2008, https://www.debr.io/article/21260.

[28] A. C. Chow, M. W. C. Paul, P. A. J. Haldenby et al., "Automated implementation of provisioned services based on captured sensor data," US Patent App, 2018.

[29] C. Xie, Y. Sun, and H. Luo, "Secured data storage scheme based on block chain for agricultural products tracking," in *Proceedings of the Third International Conference on Big Data Computing and Communications (BIGCOM),*, pp. 45–50, Chengdu, China, August2017.

[30] H. Y. Chen, "An e-government model design based on block chain," in *Proceedings of the International Conference on Manufacturing, Construction and Energy Engineering*, Hong Kong, China, November 2017.

[31] A. Kiayias, A. Russell, B. David, and R. O. Ouroboros, "A provably secure proof-of-stake blockchain protocol," in *Proceedings of the Annual International Cryptology Conference*, pp. 357–388, Santa Barbara, USA, July2017.

[32] T. Li, Y. Chen, Y. Wang et al., "Rational protocols and attacks in blockchain system," *Security and Communication Networks*, vol. 2020, Article ID 8839047, 11 pages, 2020.

[33] Y. Wang, G. Yang, T. Li et al., "Optimal mixed block withholding attacks based on reinforcement learning," *International Journal of Intelligent Systems*, vol. 35, no. 12, pp. 2032–2048, 2020.

[34] C. Wang, K. Ren, W. Lou, and L. Jin, "Toward publicly auditable secure cloud data storage services," *IEEE network*, vol. 24, no. 4, pp. 19–24, 2010.

[35] A. Juels and B. S. Kaliski, "Pors: proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 584–597, Alexandria, VA, USA, October2007.

[36] "Wikipedia contributors. Comparison of cryptographic hash functions," 2021, https://en.wikipedia.org/wiki/Comparison_of_cryptographic_hash_functions.

[37] T. Li, Z. Wang, G. Yang, Y. Cui, Y. Chen, and X. Yu, "Semi-selfish mining based on hidden Markov decision process," *International Journal of Intelligent Systems*, vol. 36, no. 7, pp. 3596–3612, 2021.

[38] T. Li, Z. Wang, Y. Chen, C. Li, Y. Jia, and Y. Yang, "Is semi-selfish mining available without being detected?" *International Journal of Intelligent Systems*, 2021.

[39] X. Yu, Y. W. Zhaojie, F. Li et al., "Impsuic: a quality updating rule in mixing coins with maximum utilities," *International Journal of Intelligent Systems*, vol. 36, no. 3, pp. 1182–1198, 2020.

[40] J. Antoine, "A one round protocol for tripartite diffie–hellman," *Journal of Cryptology*, vol. 17, no. 4, pp. 263–276, 2004.

[41] A. Juels, J. Kelley, R. Tamassia, and N. Triandopoulos, "Falcon codes: fast, authenticated lt codes (or: making rapid tornadoes unstoppable)," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1032–1047, Denver, CO, USA, October2015.

[42] A. Kircanski and T. Tarvis, "Coinbugs: enumerating common blockchain implementation-level vulnerabilities," 2021, https://arxiv.org/abs/2104.06540.

[43] D. Goldman, "The Verge Hack," explained, 2018, https://blog.theabacus.io/the-verge-hack-explained-7942f63a3017.