IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 17, 2021, accepted May 7, 2021, date of publication May 11, 2021, date of current version May 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3079143

Integrity Auditing for Multi-Copy in Cloud Storage Based on Red-Black Tree

ZHENPENG LIU[1,2], YI LIU[1], XIANWEI YANG[1], AND XIAOFEI LI[1,2]
[1]School of Cyberspace Security and Computer, Hebei University, Baoding 071002, China
[2]Information Technology Center, Hebei University, Baoding 071002, China

Corresponding author: Xiaofei Li (lixiaofei@hbu.edu.cn)

This work was supported in part by the National Natural Science Foundation of Hebei Province under Grant F2019201427, and in part by the Ministry of Education of China under Project 2017A20004.

ABSTRACT With the rapid development of cloud storage, cloud users are willing to store data in the cloud storage system, and at the same time, the requirements for the security, integrity, and availability of data storage are getting higher and higher. Although many cloud audit schemes have been proposed, the data storage overhead is too large and the data cannot be dynamically updated efficiently when most of the schemes are in use. In order to solve these problems, a cloud audit scheme for multi-copy dynamic data integrity based on red-black tree full nodes is proposed. This scheme uses ID-based key authentication, and improves the classic Merkel hash tree MHT to achieve multi-copy storage and dynamic data manipulation, which improves the efficiency of real-time dynamic data update (insertion, deletion, modification). The third-party audit organization replaces users to verify the integrity of data stored on remote cloud servers, which reduces the computing overhead and system communication overhead. The security analysis proves that the security model based on the CDH problem and the DL problem is safe. Judging from the results of the simulation experiment, the scheme is safe and efficient.

INDEX TERMS Cloud storage, data integrity auditing, red-black tree, dynamic data update.

I. INTRODUCTION

In the face of increasing user management and sharing needs, the emergence of cloud computing and cloud storage provides a new scheme for it. Users can obtain sufficient storage capacity at a lower price, and at the same time, highly concentrated computing resources greatly improve computing power [1]–[3]. Usually when people use cloud storage services, they upload their data to the cloud and store it on a remote cloud server. In order to save local storage resources, the local copy will be deleted. There are two hidden dangers in this way. One is the lack of control over the confidentiality and integrity of the data and the other is that it is difficult to recover the data if the local copy is deleted [4]–[6]. In order to solve these problems, the researchers proposed that users can encrypt data before outsourcing and sending it to a remote cloud server [7]–[9]. People also think of improving the availability and recoverability of data by storing multiple copies of the original data. Suppose that part of the users' data is damaged, only one copy of the data is needed to restore the data correctly, and it remains unchanged with the data in the cloud [10]–[12].

Ateniese *et al.* [18] proposed the concept of provable data possession (PDP). Users can effectively verify the integrity of cloud server data without retrieving the entire file, and based on homomorphic linear verification, they proposed an effective and proved a secure PDP scheme, but the scheme it proposes is only for static data. In the case of increasing data, dynamic operation of data is also a key research point of later researchers [16], [17]. Wang *et al.* [19] proposed a dynamic data scheme based on Merkle hash tree. Luo *et al.* [13] used the Shamir secret sharing concept to improve the authentication tag based on polynomials. Barsoum *et al.* [20] extended the PDP model and proposed a map-based provable multi-copy dynamic data possession (MB-PMDDP) scheme. Users can dynamically manipulate data and store fewer copies. Security is guaranteed, but any insert and delete operations will result in the need to recalculate the label and the position of the operation block, which will incur high calculation costs. Subsequently, the scheme [21] was proposed, which greatly improved the method of dynamic update efficiency. At the same time,

The associate editor coordinating the review of this manuscript and approving it for publication was Fan-Hsun Tseng.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

Yang *et al.* [22] proposed a public cloud audit scheme for dynamic update of user data and revocation of user data, but it did not solve the problem of dynamic revocation at any time. Min *et al.* [23] proposed an integrity verification scheme based on spatiotemporal chaos, which supports dynamic data analysis, blinding information, and preventing third parties from leaking user data privacy. Min *et al.* [24] proposed a data integrity verification scheme based on a binary balanced tree. The efficiency of data update has been greatly improved, and it also provides us with a research direction.

In order to realize the public audit of multi-copy storage, Curtmola *et al.* [25] proposed an audit scheme based on RSA signatures, but it cannot support analyzing dynamic data. In order to solve this problem, Barsoum and A. Barsoum and Hasan [26] proposed a DPDP structure, which supports the dynamic behavior of data copy, block modification, fast deletion and fast addition on the cloud server. Then, based on the pseudo-random function and BLS signature, Shacham and Waters [27] proposed a remote public data integrity verification scheme and a private data integrity verification scheme. Shen *et al.* [28] proposed a dynamic structure of a doubly linked list, which can effectively verify whether the data stored by the cloud service provider is safe. Initially, the generation of the public and private key of most schemes based on the certificate issuing authority PKI, which was a huge overhead for certificate management, so later schemes began to use ID-based signatures to reduce the overhead of certificate management [14], [15]. Shen *et al.* [29] proposed an identity-based remote data integrity audit scheme, which realizes data sharing by hiding sensitive information and simplifies complex certificate management. And then, Zhang *et al.* [30] proposed a multi-copy full dynamic session protocol (MR-DPDP). In this protocol, they used the rank characteristics of the Merkel hash tree to verify the data, but all its copies are existing on a cloud storage server, the performance of multiple copies is meaningless. To solve this problem, Li *et al.* [31] proposed to deliver all copies to different cloud storage servers, and audit the integrity of all copies through homomorphic verifiable tags.

Other aspects, such as privacy protection [32] and data deduplication [33], have also been studied in remote data integrity auditing. Although the batch update storage of multiple copies can be achieved, the audit cost has always been high and the efficiency is not high. Therefore, in this article, we explore how to optimize the storage structure and reduce dynamic operation overhead, so as to design an effective dynamic multi-copy integrity audit scheme.

Specifically, the main contributions of this paper can be summarized as follows:

1) The red-black tree data structure is designed to store data, which is conducive to efficient storage of data, improves the efficiency of data update, reduces the consumption of data storage, and more standardizes data management. Greatly avoid the possibility of TPA and CSP spoofing attacks.

TABLE 1. Symbols and their meanings.

Symbol	Meaning
p	One large prime
G_1, G_2	Multiplicative cyclic groups with order p
t	Number of copies
n	Number of file blocks per copy
L	Number of characters per file block
$F = \{m_1^{'}, m_2^{'}, ..., m_n^{'}\}$	The original file F
$F^* = \{m_1, m_2, ..., m_n\}$	The encrypted file F^*
e	A bilinear pairing map $e: G_1 \times G_1 \rightarrow G_2$
filename	The name of the file stored on the cloud
$r_{i,j}$	A random number
$b_{i,j}$	The j-th block of the i-th file copy, where $b_{i,j} = m_j + r_{i,j}$
F_i	set of i copies
$\sigma_{i,j}$	The signature of the j-th block of the i-th file copy
θ_j	The signature of the j-th block of all file copies
H	A cryptographic hash function: $H: \{0,1\}^* \rightarrow G_1$

2) A digital signature scheme based on ID is designed, which improves the security of the private key and the secret key pair, and avoids the resource consumption caused by the use of PKI as the basic public and private key generation, and effectively reduces the cost.
3) The design allows the user to determine the number of copies to be generated, and to encrypt the copies, and then to generate the corresponding signatures on the encrypted files. If the user wants to get the original file, he can easily convert the encrypted file back to the original valid original file. This method effectively protects the security of data sharing and data storage, and also realizes remote data integrity auditing.

The safety analysis of the scheme was carried out through general experiments, and its performance was proved through concrete realization. The results showed that the proposed scheme had achieved the expected results in terms of safety and efficiency.

II. SYMBOLS AND PRELIMINARIES

In this section, the system model, security model, and system composition will be introduced. Before the introduction, some symbols used in our scheme are shown in Table 1.

A. SYSTEM MODEL

The system model involves four different entities: cloud server (CSP), users, key pair generator (PKG) and third-party audit agency (TPA), as shown in Figure 1:

1) CSP: Cloud servers provide users with huge data storage space. Users store data in cloud servers, which can save local storage space, use their powerful cloud computing capabilities, and share data with others.

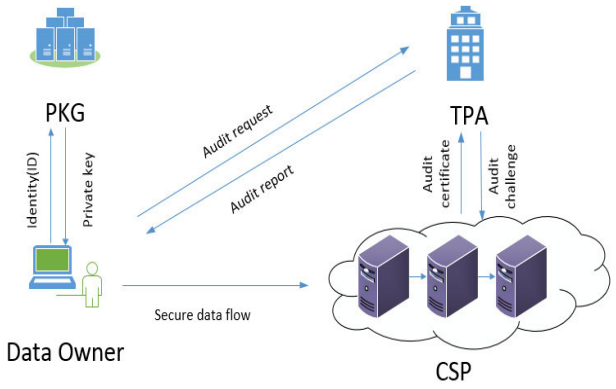

FIGURE 1. The system model of the data integrity auditing.

2) User: A user is a member of an organization, and a large number of files are stored in the cloud server.
3) PKG: PKG is trusted by other entities. It is responsible for generating system public parameters and signature key pairs.
4) TPA: TPA is a public verifier. It can audit the integrity of the data stored on the cloud server on behalf of the user.

The user outsources his data to the CSP. The user and the CSP do not trust each other, so the data flow between them must be encrypted, and the encrypted data flow can be considered safer. TPA is selected by the user and can verify data integrity on behalf of the user. When the user wants to verify the integrity of a certain data file in the CSP, he will delegate the task to the TPA. After receiving the task, the TPA will send the challenge information to the CSP. TPA will return the final audit result to the user based on the proof returned by the CSP.

B. SYSTEM COMPOSITION

A multi-copy dynamic data integrity audit protocol includes the following algorithms:

1) KeyGen: This algorithm is executed by the user. It accepts a security parameter x as input, and generates a key pair (sk, pk) and system parameters, which will be used in the following algorithm.
2) ReplicaGen: This algorithm is executed by the user. It accepts a file F as input and outputs $F_i = \{b_{i,j}\}_{1\le i\le t, 1\le j\le n}$.
3) TagGen: The algorithm is executed by the user. It takes different copy files and private key sk as input, and outputs aggregate label $\theta_j = \prod_{i=1}^{t} \sigma_{i,j}$.
4) Store: This algorithm mainly runs on users, CSP and TPA. The TPA verification output result is 1 or 0, and the meaning of 1 or 0 indicates whether you agree to upload data.
5) ChalGen: This algorithm is mainly run by users and TPA to generate a random challenge $chal$.
6) ProofGen: This algorithm mainly runs on CSP to generate an integrity audit certificate P.
7) VerifyProof: This algorithm mainly runs in the user and TPA to verify the proof P. The algorithm will output 1 or 0 after the verification is completed, and 1 or 0 indicates whether the CSP is allowed to pass the user or TPA verification.
8) DynaGen: This algorithm mainly runs on the user to generate a dynamic request $updateop$.
9) VerifyDyna: This algorithm mainly runs in the user and TPA to verify the update request. The algorithm will output 1 or 0 after the verification is completed, and 1 or 0 indicates whether the CSP is allowed to pass the update request of the user or the TPA.

C. SECURITY MODEL

A secure PDP protocol should be complete, which means that if the prover can generate a valid certificate and pass the verification, the data must be stored. In order to verify the security, we have extended the security model to some copies of the file. We set up a game between the challenger C and the adversary $\mathcal{A}$ to show that the adversary $\mathcal{A}$ poses a threat to the security of the entire data integrity audit program. In this data model, the data owner is regarded as the challenger C, and the untrusted cloud server is regarded as the adversary $\mathcal{A}$. This game includes the following stages:

1) Setup phase: The challenger C initializes the system to obtain the public parameters, master key msk and signature key pair.
2) Queries phase: Adversary $\mathcal{A}$ and challenger C do three queries next.
    1) Hash Queries: The adversary $\mathcal{A}$ makes a series of hash queries to the challenger C. The challenger C returns the hash value to the adversary $\mathcal{A}$.
    2) Extract Queries: The adversary $\mathcal{A}$ queries the private key for the authentication ID. The challenger C runs the $Extract$ algorithm to generate the private key sk_{ID} and sends it to the adversary $\mathcal{A}$.
    3) SigGen Queries: The adversary $\mathcal{A}$ queries the signature of file F. The challenger C runs the $SigGen$ algorithm and sends the signature to the adversary $\mathcal{A}$.
    3) Challenge phase: In this phase, the adversary $\mathcal{A}$ is the audit object, and the challenger C sends a challenge $chal = \{j, v_j\}_{\Upsilon_1\le j\le\Upsilon_c}$ to the adversary. At the same time, the adversary $\mathcal{A}$ is requested to provide a proof of data possession P based on the challenge $chal$.
    4) Forgery phase: After accepting the request of the challenger C, the adversary $\mathcal{A}$ generates a proof of possession P of the data block. If the proof can pass the challenge of the challenger C with a probability that cannot be ignored, it can say that the adversary $\mathcal{A}$ has succeeded in the above game.

D. DESIGN GOAL

In order to achieve safe and effective dynamic audit of cloud data, our schema design should achieve the following goals:

1) Public audit: TPA can verify the integrity of the data stored in the cloud server on behalf of the user, and will not cause other problems to the user.
2) Storage accuracy: Only when the CSP can completely save the data stored by the user, the audit certificate *chal* generated by the CSP can pass the TPA audit successfully.
3) Dynamic operation support: Allow users to perform reasonable dynamic update operations (insert, delete, and modify) the data stored on the CSP, and ensure the efficiency of the entire cloud storage system.
4) Privacy protection: Design a random factor to encrypt the original file, and only calculate the signature of the encrypted file. The CSP and TPA cannot get the valid and correct original file.

III. OUR PROPOSED SCHEME

A. DYNAMIC STORAGE STRUCTURE RED-BLACK TREE

In order to establish a more complete cloud storage service system, we use the red-black tree data structure for data storage. Compared with the balanced binary tree, although the red-black tree algorithm has the same time complexity, its statistical performance is higher. For dynamic data update, the traditional tree storage structure requires a lot of queries and adjustment operations in the worst case. If the data is stored in the data structure of the red-black tree, because it is not strictly balanced, its query ability is slightly weaker, but its insertion and deletion capabilities are completely stronger than the balanced binary tree. In our *scheme*, each copy corresponds to a red-black tree with a complete node and is stored in the CSP, and the data block copy in each copy corresponds to a node value, which is stored in the TPA. According to the node value of the binary tree, the location of the data block can be verified, which greatly reduces the verification path and improves the system efficiency. Here, the red-black tree stored in the CSP is abbreviated as C-RBTree, and the red-black tree stored in the TPA is abbreviated as T-RBTree.

In T-RBTree, each node should store the node value $N(b_{i,j})$ and the serial number of the data block. As shown in figure 2.

The leaf node values and non-leaf node values can be calculated by the following equations (1) and (2).

$$X(m_{i,j}) = \frac{\sum_{k=1}^{L} A_{i,j,k} \times \frac{1}{256}}{L} \tag{1}$$

$$\begin{cases} K(m_{i,j}) = 1 - aN_L^2 + bN_R \\ T(m_{i,j}) = \dfrac{\sum_{k=1}^{L} A_{i,j,k} \times \frac{1}{256}}{L} \\ X(m_{i,j}) = 1 - aK(m_{i,j})^2 + bT(m_{i,j}) \end{cases} \tag{2}$$

$A_{i,j,k} \in [0, 255](1 \le i \le t, 1 \le j \le n, 1 \le k \le L)$ represents the ASCLL of the k-th character of the j-th block of the i-th copy. N_L and N_R are the left and right child nodes of a non-leaf node. In non-leaf nodes, first using the henon map to calculate the value $K(m_{i,j})$ by N_L and N_R, then calculate the value $T(m_{i,j})$ according to the leaf node calculation method, and perform henon mapping on the two values to obtain the non-leaf node value $X(m_{i,j})$. In this way, once a node changes, the change of the node value can be improved, and the integrity of the data can be better judged. The data file stored in the cloud server generally has multiple copies. Especially when the number of copies is relatively large, checking the binary value of the root node of each red-black tree will bring great difficulties to verification. Therefore, by aggregating the root node values of all file copies into the metadata M, the position of the data block of the changed file copy can be determined by checking the metadata. The calculation method of metadata M is equation (3):

$$M = h(R_1 + R_2 + \ldots + R_t) \tag{3}$$

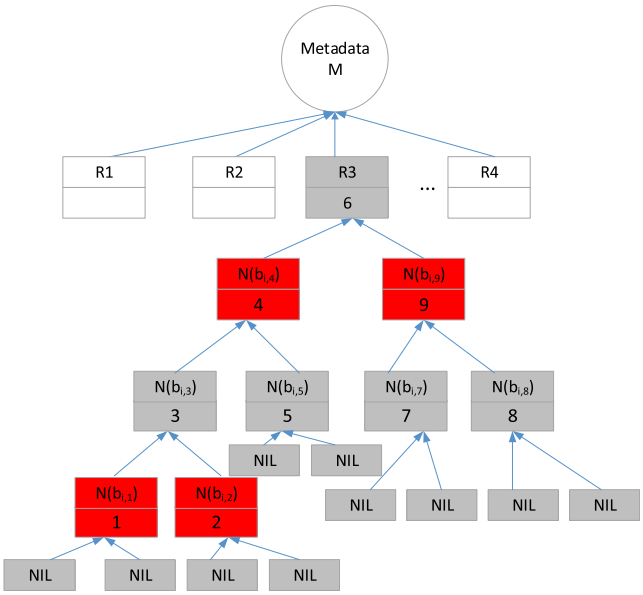

FIGURE 2. The full-node red-black tree T-RBTree.

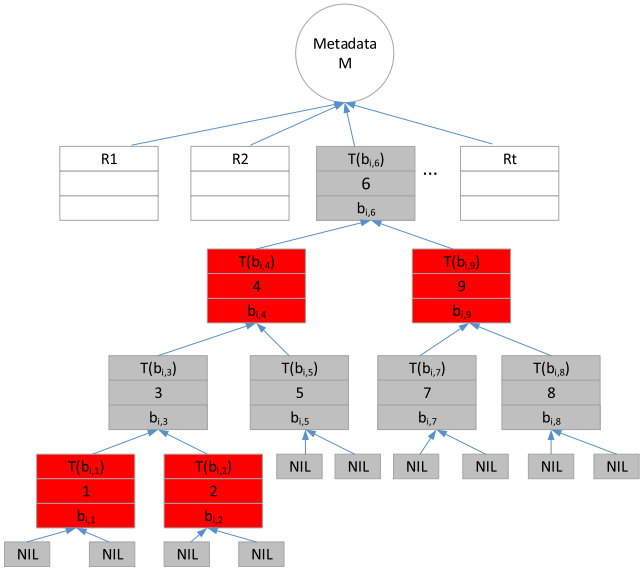

FIGURE 3. The full-node red-black tree C-RBTree.

In C-RBTree, each node should store the node value and the serial number of the data block shown in Figure 3.

The verifier wants to verify the integrity of the data stored in the cloud. It does not need all the data to participate in the verification. It only needs to calculate the node value of the data block according to the verification path of the verified data block. The integrity of the block can be known based on the comparison of the node value, which also greatly reduces the waste of resources.

B. CONSTRUCTION OF OUR PROPOSAL

In the program, the review process is divided into two phases: the setup phase and the verification phase. The previous phase is some preparations, including: KeyGen, ReplicaGen and TagGen. The system setting is mainly for the user. The user generates a public key and a private key pair through the KeyGen algorithm, and then performs other pre-processing tasks through ReplicaGen and TagGen. The latter phase includes three algorithms: ChalGen, ProofGen and VerifyProof. At this stage, users, CSP, and TPA will participate to perform data verification. In the ChalGen algorithm, the TPA sends verification challenge information to the CSP, and then the CSP responds in the ProofGen algorithm to prove the integrity of the stored data. In the final VerifyProof algorithm, TPA audits the proof and sends the audit results to the user by TPA. The following are the specific implementation details:

1) KeyGen algorithm (1^k)
The user runs KeyGen to generate a public key and signature key pair. The PKG selects two large prime number p-order cyclic multiplicative groups G_1 and G_2, in which g and u are generating element of G_1. A bilinear map $e : G_1 \times G_1 \rightarrow G_2$. Z_p^* is a set of non-negative integers less than p, and a pseudorandom function $f : Z_p^* \times Z_p^* \rightarrow Z_p^*$. Hash encryption function is $H : \{0,1\}^* \rightarrow G_1, H_1 : G_1 \rightarrow Z_p^*, H_2 : \{0,1\}^* \rightarrow G_1$. Firstly, the user randomly selects a random number $msk = x \in Z_p^*$ as a part of the system key, and then calculates a system public key $mpk = g^x \in G_1$. The public parameter is $\{G_1, G_2, p, e, g, u, H, H_1, H_2, mpk\}$ and msk is only kept by the user. After given the public parameters, the user sends his identity $ID \in \{0,1\}^*$ to the key management center to apply for a random signature key pair (sk, pk), in which sk is the other part of the system key. After receiving the request, the key management center calculates $ssk = H(ID)^x$ and returns it to the user. After the user receives the signature key pair, verify its correctness:

$$e(ssk, g) = e(H(ID), mpk) \tag{4}$$

If the verification fails, the user resends the request to the key management center; otherwise, the user chooses a random number as a secret value, and then calculates:

$$sk = \{sk_1, sk_2\} = \{ssk, (s + H_1(ssk)) \bmod p\}$$
$$pk = g^{sk_2}$$

2) ReplicaGen algorithm $(\cdot)$
This process is illustrated in Figure 4 as an example. The user generates multiple copies of the original file by running the ReplicaGen algorithm. The user first selects *filename* $\in\{0,1\}^*$ for the copies, and divides it into n blocks. Each block has the same size, if the size of the last block is smaller than the other blocks, add 0 to fill it. We select a random number $r_{i,j} \in Z_p^*$. The original file is encrypted, and the encrypted file is $F^* = \{m_1, m_2, ..., m_n\}$ and the data block $b_{i,j} = m_j + r_{i,j}$. If you want to get the original file, you only need to calculate $m_j = b_{i,j} - r_{i,j}$ to get the contents of the original file. Finally, t different copy files $F_i = \{b_{i,j}\}_{1\le i\le t, 1\le j\le n}$ can be obtained.

3) TagGen algorithm $(\cdot)$
The user generates aggregated tags and metadata by running the TagGen algorithm. The owner of the data selects a random element $U_{i,j}$, and generates a signature $\sigma_{i,j} = sk_1 \cdot [H_2(U_{i,j}^{b_{i,j}}) \cdot \mathrm{u}^{b_{i,j}}]^{sk_2} \in G_1$ for each random element, where $U_{i,j} = \{filename \,||\varepsilon\, ||\eta\}$ is the identification code of data block $b_{i,j}$, *filename* is the name of file F_i, ε is the virtual index of data block $b_{i,j}$, and η is the use of random function f to encrypt *filename* and ε. The public parameters selected by $U_{i,j}$ from G_1 correspond to the corresponding management cycle. Then an aggregate signature $\theta_j = \prod_{i=1}^{t} \sigma_{i,j}$ is generated. The user initializes the storage into two red-black trees. One red-black tree is used to store the node value and the serial number of the data block, and the other one stores the node value, data block $b_{i,j}$ and the serial number of the data block. The metadata of each copy is calculated by equation 3. Then, the user sends $\{pk, M,$ T-RBTree$\}$ to the TPA and sends $\{F_i, \theta_j,$ C-RBTree$\}$ to the CSP. After receiving the data, the CSP needs to verify the consistency between the data block and the signature through the following equation:

$$e(\theta_j, g) = e(H(ID), mpk) \cdot e(\prod_{i=1}^{t} H_2(\mathrm{U}_{i,j}^{b_{i,j}}) \cdot u^{b_{i,j}}, pk) \tag{5}$$

If the verification fails, the CSP refuses to store the data sent by the user, and the algorithm outputs 0. Otherwise, CSP will store relevant data. Finally, the user deletes the local copy and file label stored on the local storage, leaving only the system key and metadata M.

4) ChalGen algorithm $(\cdot)$
This part of the process is illustrated in Figure 5 as example.
TPA runs the ChalGen algorithm to issue an integrity audit certification challenge to the CSP on behalf of the user. TPA will also periodically send verification challenges to the CSP to confirm whether the CSP has all the copies and whether these are complete. The user entrusts a specific file verification task to the TPA.

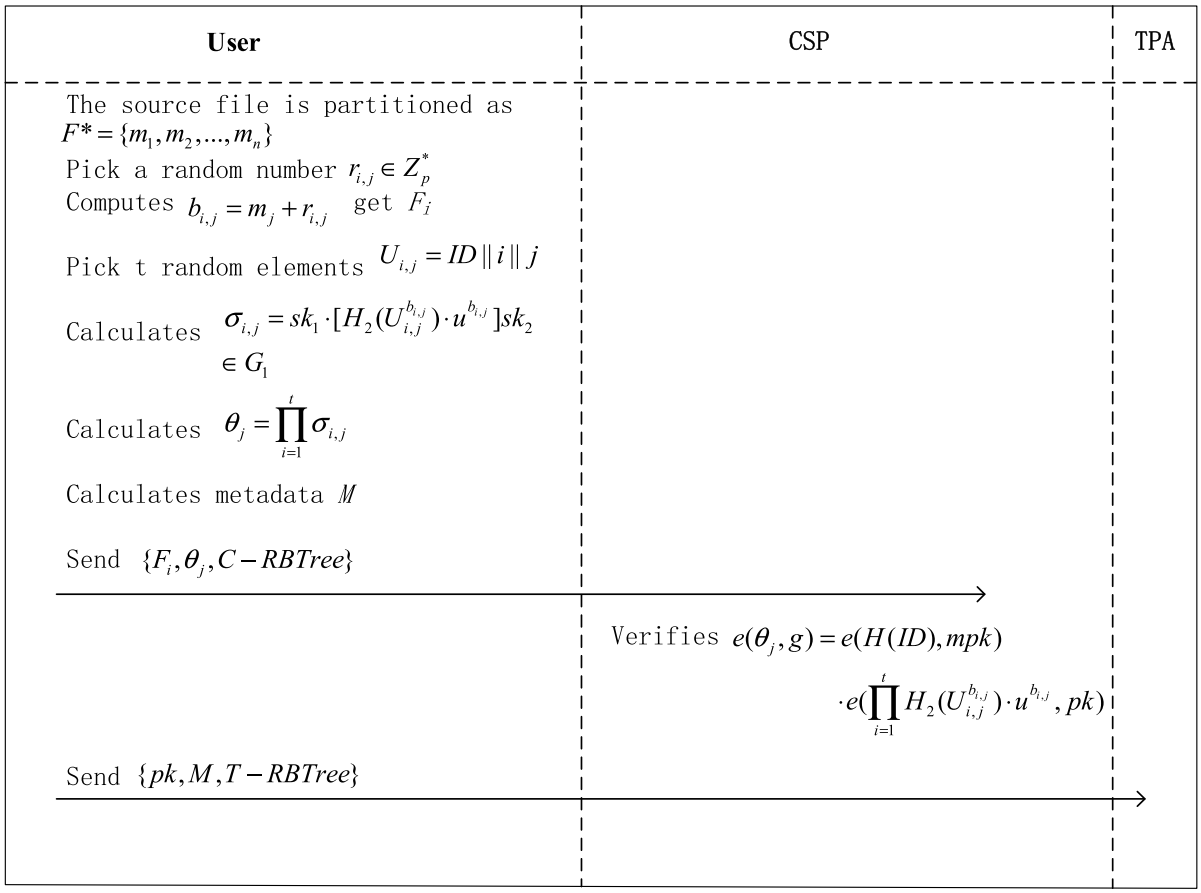

FIGURE 4. The copy generation and label generation process.

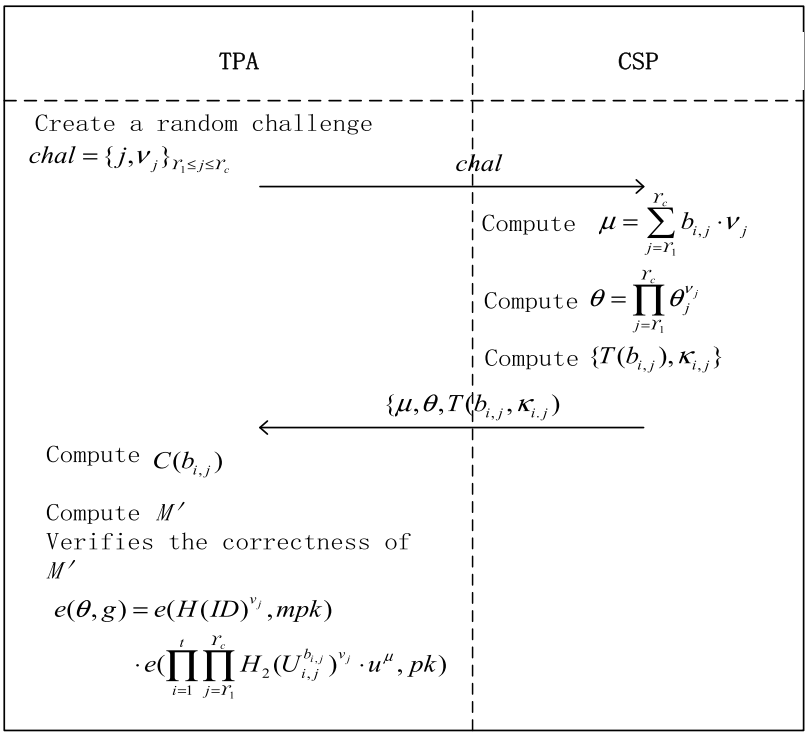

FIGURE 5. The challenge generation and evidence verification process.

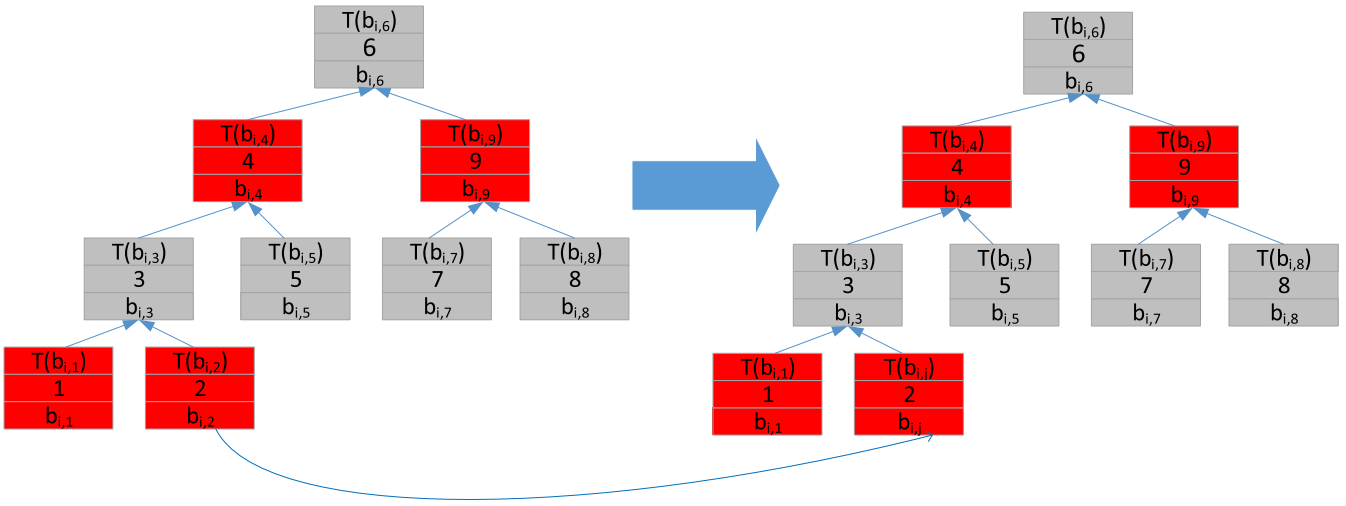

FIGURE 6. Data block $b_{i,2}$ is modified to $b_{i,j}$ in the red-black tree.

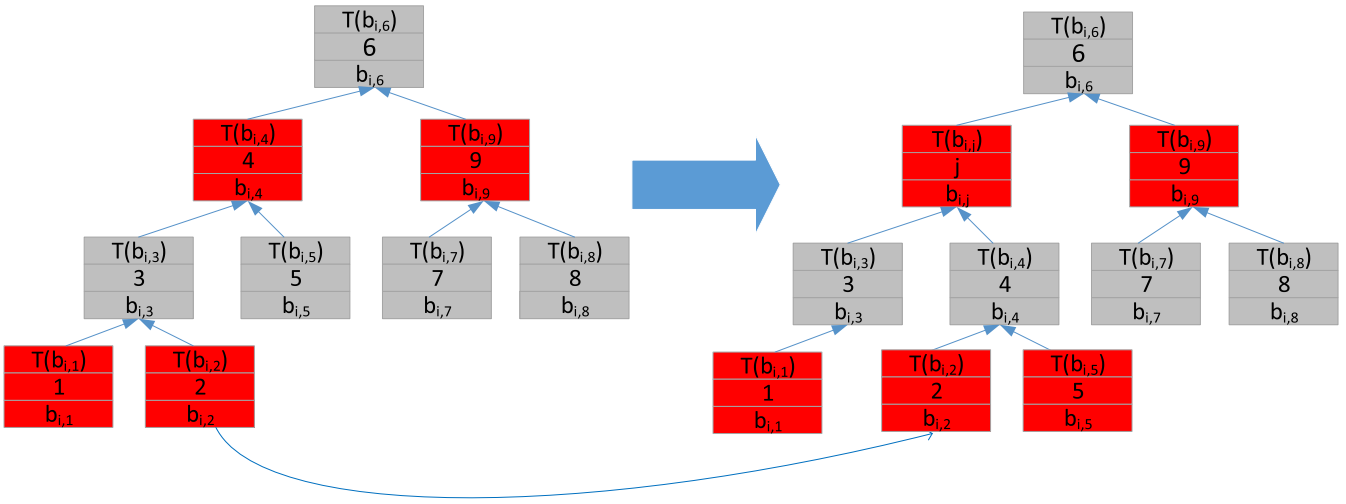

FIGURE 7. Data block $b_{i,j}$ is inserted into the data block $b_{i,2}$ position in the red-black tree.

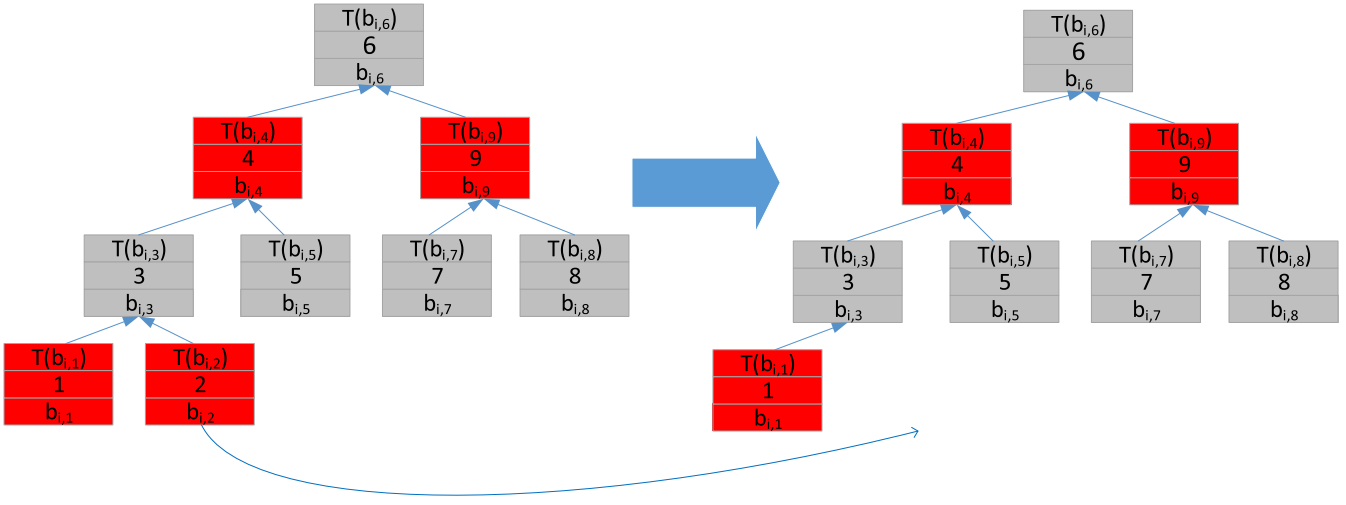

FIGURE 8. Data block $b_{i,2}$ is deleted in the red-black tree.

Then, the TPA randomly selects $c(c \in [1, n])$ elements to form a block index integer set $I = \{\gamma_1, \gamma_2, \ldots, \gamma_c\}$, and selects a random element $v_j \in Z_p^*$ for each block index $j \in I$ to form an audit challenge $chal = \{j, v_j\}_{\gamma_1 \le j \le \gamma_c}$ and sends it to the CSP.

5) ProofGen algorithm $(\cdot)$
CSP runs ProofGen algorithm to generate corresponding audit certificate for data block. After receiving the verification challenge *chal*, the CSP finds the storage directory server according to the sent command, and then sends $\{j\}_{\gamma_1 \le j \le \gamma_c}$ to the corresponding storage server. The storage server returns $\{T(b_{i,j}), \kappa_{i,j}\}_{1 \le i \le t, \gamma_1 \le j \le \gamma_c}$ to the main storage server after receiving it. The $\kappa_{i,j}$ represents the verification path of data block $b_{i,j}$. If the verifier wants to verify the integrity of $b_{i,2}$, the primary storage server will send the authentication path $\kappa_{i,2} = \{\lambda_{i,1}, \lambda_{i,2}, \lambda_{i,3}, \lambda_{i,5}, \lambda_{i,4}, \lambda_{i,9}, \lambda_{i,6}\}$ of $b_{i,2}$, where $\lambda_{i,1} = \{T(b_{i,1})\}$, $\lambda_{i,4} = \{T(b_{i,1})||T(b_{i,2})||T(b_{i,3})||T(b_{i,5})||T(b_{i,4})\}$,and $T(b_{i,j})_{1 \le i \le t, \gamma_1 \le j \le \gamma c}$ represents the node value of a node in the verification path, and then calculates the linear combination $\mu = \sum_{j=\gamma_1}^{\gamma_c} b_{i,j} \cdot v_j$, $\theta = \prod_{j=\gamma_1}^{\gamma_c} \theta_j^{v_j}$ of the data block $b_{i,j}$. The CSP returns the audit certificate $P = \{\mu, \theta, \{b_{i,j}, \kappa_{i,j}\}_{1 \le i \le t, \gamma_1 \le j \le \gamma_c}\}$ to the TPA.

6) VerifyProof algorithm $(\cdot)$
After receiving the audit certificate P, TPA calculates the node value $C(b_{i,j})$ based on $\kappa_{i,j}$, and then calculates M' based on $\{C(b_{i,j}), \kappa_{i,j}\}_{1 \le i \le t, \gamma_1 \le j \le \gamma_c}$. TPA checks $M' = M$, if it is not equal, output "Failure", otherwise continue to calculate:

$$e(\theta, g) = e(H(ID)^{v_j}, mpk) \cdot e(\prod_{i=1}^{t} \prod_{j=\gamma_1}^{\gamma_c} H_2(U_{i,j}^{b_{i,j}})^{v_j} \cdot u^{\mu}, pk) \quad (6)$$

C. DYNAMIC OPERATION VERIFICATION

For each data change operation, the user will send a request to the CSP to run the dynamic operation. After receiving the request, the CSP will dynamically update the data. According to the user's request, it will implement operations such as modification, insertion, and deletion.

When the user needs to update the data, he should send a request $(F_i, U_{i,j}, \sigma_{i,j}, UpdateOp)$ to the CSP. Here it is assumed that *UpdateOp* is *DM* for modification, *UpdateOp* is *DI* for insertion, and *UpdateOp* is *DD* for deletion. After the CSP receives it, it generates an update certificate V_{update} and returns it to the user. The user sends (P_{update}, V_{update}) to the TPA, where V_{update} is the user update request. The TPA can determine if the update request operation is complete according to the user. The TPA returns "TRUE" to the user, otherwise, return "FALSE". Different operations for different data blocks, as follows:

1) DATA MODIFICATION (DM)

During the data modification, if the user wants to replace the data block $b_{i,j}$ to $b'_{i,j}$, he should encrypt the original file to obtain m'_j, and the $r'_{i,j}$ is added to m'_j to generate $b'_{i,j}$.Then, the new signature is $\sigma'_{i,j} = sk_1 \cdot [H_2(U_{i,j}^{b'_{i,j}}) \cdot u^{b'_{i,j}}]^{sk_2} \in G_1$ and the corresponding node value $C(b'_{i,j})$ is generated for this new data block. The user send $\{filename, \varepsilon, b'_{i,j}, \sigma'_{i,j}\}$ to CSP, send $\{filename, \varepsilon\}$ to TPA.

After receiving the request, the CSP obtains the authentication path $\kappa_{i,j}$ in accordance with the *filename* and block index ε, and then finds the node in C- RBTree, replaces $b_{i,j}$ with $b'_{i,j}$, replaces $\sigma_{i,j}$ with $\sigma'_{i,j}$, calculates the node value $C(b'_{i,j})$ and replaces $C(b_{i,j})$, calculates new aggregate tag $\theta'_j = \prod_{i=1}^{t} \sigma_{i,j}$ and the authentication path is unchanged. After receiving the request, the TPA obtains the corresponding node according to *filename* and ε, and then finds the corresponding node in the T-RBTree, and calculates the new node value $T(b'_{i,j})$, and replaces $T(b_{i,j})$ to $T(b'_{i,j})$, and recalculates the metadata M' at the same time.

2) DATA INSERTION (DI)

During the data insertion, once the data is inserted, the file storage structure will be changed, which means the constructed storage tree will be changed, so it needs to be adjusted. It is assumed that after inserting a block $b'_{i,j}$ into the data block $b_{i,j}$, the user first encrypts the original file to obtain m'_j, and the $r'_{i,j}$ is added to the m'_j to obtain $b'_{i,j}$. The new signature $\sigma'_{i,j} = sk_1 \cdot [H_2(U_{i,j}^{b'_{i,j}}) \cdot u^{b'_{i,j}}]^{sk_2} \in G_1$ and the corresponding node value $C(b'_{i,j})$ are generated for this new data block. The user sends {filename, ε, $b'_{i,j}$, $\sigma'_{i,j}$} to the CSP, and sends {filename, ε} to the TPA.

After receiving the request, the CSP obtains the verification path $\kappa_{i,j}$ according to the *filename* and block index ε, and then finds the node $b_{i,j}$ in C-RBTree. After inserting the $b'_{i,j}$ into $b_{i,j}$, the node value $C(b'_{i,j})$ is calculated, and then the new aggregate signature is $\theta'_j = \prod_{i=1}^{t} \sigma_{i,j}$. The C-RBTree verification path and node value of each node of are also updated. After receiving the request, the T-RBTree value is also updated, and the metadata M' is recalculated.

3) DATA DELETION (DD)

During data deletion, once the data is deleted, the file storage structure will be changed that the storage tree of our scheme will be changed, and it need to be adjusted. Suppose user want to delete block $b'_{i,j}$, the user sends {filename, ε} to the CSP, and sends {filename, ε} to TPA.

After the CSP receives the request, it obtains the path $\kappa_{i,j}$ based on the *filename* and block index ε to find the node and deletes it in C-RBTree. The verification path value of each node of C-RBTree is also updated. After the TPA receives the request, the node will be found according to *filename* and ε and it will be deleted. The TPA adjusts the T-RBTree and calculates the metadata M'.

IV. SECURITY ANALYSIS

This section is analyzed for the proposed scheme and the scheme will be described by the following theorem.

Theorem 1 (Correctness): The proposed program contents the following properties:

1) (Signing secret key correctness) When the PKG sends the correct signature key to the user, this signature key pair can be verified by the user.
2) (Correctness of the original file and its corresponding signatures) The original file and its corresponding signature can be verified by CSP.
3) (Authentication correctness) When the cloud server can completely store the file, the proof of its generation can pass the TPA verification.

Prove.

1) The PKG sends the correct to the user, and the user verifies whether the formula (4) is established in the Keygen algorithm. Based on the nature of the bilinear map, the final derivation of the equation can be proved to be correct:

$$\begin{aligned} & e(ssk, g) \\ & = e(H\ (ID)^{x}, g) \\ & = e(H\ (ID), mpk) \end{aligned}$$

2) The CSP determines the original file F_i and its corresponding tag $\sigma_{i,j}$, which will verify the formula (5) in the TagGen algorithm. Based on the nature of bilinear map, it can be derived to obtain the results prove that it is correct:

$$\begin{aligned} & e(\theta_j, g) \\ & = e(\prod_{i=1}^{t} \sigma_{i,j}, g) \\ & = e(sk_1, g) \cdot e((H_2(U_{i,j}^{b_{i,j}}) \cdot u^{b_{i,j}}) sk_2, g) \\ & = e(H(ID), g^{x}) \cdot e(\prod_{i=1}^{t} H_2(U_{i,j}^{b_{i,j}}) \cdot u^{b_{i,j}}, g^{sk_2}) \\ & = e(H(ID), mpk) \cdot e(\prod_{i=1}^{t} H_2(U_{i,j}^{b_{i,j}}) \cdot u^{b_{i,j}}, pk) \end{aligned}$$

3) CSP has been verified after receiving a valid audit challenge $chal = \{j, \upsilon_j\}$. Depending on the nature of the bilinear map, the verification equation (6) is derived from the left to the right and then is derived from the right side to the left, and finally the result is proved correct:

$$\begin{aligned} & e(\theta, g) \\ & = e(\prod_{j=\gamma_1}^{\gamma_c} \theta_j^{v_j}, g) \\ & = e(\prod_{j=\gamma_1}^{\gamma_c} (\prod_{i=1}^{t} \sigma_{i,j})^{v_j}, g) \\ & = e(\prod_{j=\gamma_1}^{\gamma_c} (\prod_{i=1}^{t} sk_1 \cdot [H_2(U_{i,j}^{b_{i,j}}) \cdot \mathrm{u}^{b_{i,j}}]^{sk_2})^{v_j}, g) \\ & = e(sk_1^{v_j}, g) \cdot e(\prod_{j=\gamma_1}^{\gamma_c} (\prod_{i=1}^{t} [H_2(U_{i,j}^{b_{i,j}}) \cdot \mathrm{u}^{b_{i,j}}]^{sk_2})^{v_j}, g) \\ & = e(ssk^{v_j}, g) \cdot e(\prod_{j=\gamma_1}^{\gamma_c} (\prod_{i=1}^{t} H_2(U_{i,j}^{b_{i,j}}) \cdot \mathrm{u}^{b_{i,j}})^{v_j}, g^{sk_2}) \\ & = e(H(ID)^{x \cdot v_j}, g) \cdot e(\prod_{j=\gamma_1}^{\gamma_c} \prod_{i=1}^{t} H_2(U_{i,j}^{b_{i,j}})^{v_j} \\ & \quad \cdot \prod_{j=\gamma_1}^{\gamma_c} \prod_{i=1}^{t} \mathrm{u}^{b_{i,j} \cdot v_j}, g^{sk_2}) \\ & = e(H(ID)^{v_j}, \mathrm{mpk}) \cdot e(\prod_{i=1}^{t} \prod_{j=\gamma_1}^{\gamma_c} H(U_{i,j}^{b_{i,j}})^{v_j} \cdot u^{\mu}, pk) \end{aligned}$$

Theorem 2 (Audit Security): Assume that the CDH problem is difficult to solve in the bilinear group and the file signature generated in the signature scheme is fundamentally difficult to forge. In the scheme proposed in Section 4, if users face an untrusted CSP, the data stored in the user may be damaged, missing or even modified. At this time, when the CSP is audited, it cannot be calculated a forged evidence that can be verified by TPA.

Prove. We will use the zero-knowledge theorem to prove the theorem. If the CSP can generate a forged audit certificate that can be verified by the TPA when the data is incorrect (such as the data has been tampered with by the CSP privately or the data is lost), then a knowledge extractor is constructed in our proposed scheme. By continuously extracting the challenged data block information, a complete challenge data block can finally be obtained. Our proof will be completed through a series of games.

Game 1. In Game 1, the challenger C and adversary will follow the steps in the proposed scheme. The challenger C first runs the algorithm to obtain the master key *msk*, system public key *mpk* and public parameters. After obtaining this information, it is necessary to request the most important signature key pair (*sk*, *pk*). After the challenger C obtains the information, the public parameters shall be disclosed so that the adversary can obtain and generate the corresponding audit certificate to challenge. The adversary queries a series of data block signatures that need to be challenged and then the challenger C runs the signature generation algorithm to generate the corresponding signatures of these data blocks. The challenger C sends the generated data block signature to the adversary and issues a challenge to the adversary. Finally, the adversary generates a data ownership certificate P based on the public parameters and the data block signature and returns it to the challenger C. If the proof of this data can pass the challenger's verification with a non-negligible probability, it can be said that the adversary has won the Game.

Game 2. In Game 2, after obtaining the master key *msk*, system public key *mpk*, public parameters, and signature key pair, the challenger C receives the signature of the data block that the adversary wants to query and record it. Then the challenger C will analyze each subsequent challenge and response process with the adversary. If the adversary can generate a proof that is verified by the challenger C, but the aggregate signature generated by the adversary is not equal to that generated by the challenger, C then the challenger C can only declare failure and abort the challenge.

Analysis. Assuming that the adversary can win Game 1 with a non-negligible probability, a simulator can be constructed to solve the CDH problem. This simulator gives $g, g^{\alpha}, g^{\beta} \in G_1$ and its ultimate goal is to generate $g^{\alpha\beta}$, where the behavior of the simulator is the same as the role of the challenger C in Game 1.

1) The simulator randomly selects a generator g in G_1, and then randomly selects an element $\alpha \in Z_p^*$ and sets $mpk = g^{\alpha} \in G_1$.
2) The simulator programs random H, where H is regarded as a random prediction model in the evidence. It stores a list of queries and responds to them in a unified way. The adversary arbitrarily selects a user ID to adaptively execute H-queries, assuming that in the case of probability p, this ID belongs to the query list $L = (ID, h, ssk)$ of H. The simulator extracts the corresponding information, where $ssk = H(ID) = g^h$. The probability of not belonging to the query list in H is naturally $1- p$, and the simulator randomly selects a value $h' \in Z_p^*, ssk' = H(ID) = g^{\beta h'}$.Then the simulator returns ssk' to the adversary and adds the new information to the query list.
3) The adversary queries the signature key pair based on the selected ID. At this time, the simulator maintains a new query list $L_1 = (ID, h_1, ssk_{ID}, sk_{ID}, pk_{ID})$. First, the simulator queries the adversary's ID and ssk. If it is not stored in the H query list, it will return to the second step. If the relevant information is found in the storage list of H, the simulator selects a random value $x \in Z_p^*$ and then orders $x_{ID'} = x$. Next, the simulator sets $sk_{ID'} = x_{ID'} + h_1' \bmod p, pk_{ID'} = g^{sk_{ID'}}$ and updates the query list L_1, and finally sends $pk_{ID'}$ to the adversary.
4) The adversary runs H query on the data block $b_{i,j}$ by itself. The simulator maintains a query list $L_2 = (U_{i,j}, h_2)$ here. Assuming that in the case of probability p, the $U_{i,j}'$ sent by the adversary belongs to the query list L_2, then the simulator sends $g^{h_2'}$ to the adversary. The probability of not belonging to L_2 is 1-p. At this time, the simulator randomly selects a $h_2' \in Z_p^*$ and sends $g^{h_2'}$ to the adversary and at the same time updates $(U_{i,j}', h_2')$ to the query list L_2. The simulator can calculate $\sigma_{i,j}$ for the data block $b_{i,j}$ that

$$\begin{aligned}\sigma_{i,j} &= sk_1 \cdot [H_2(U_{i,j}^{b_{i,j}}) \cdot u^{b_{i,j}}]^{sk_2} \\ &= ssk \cdot [g^{h_2'} \cdot u^{b_{i,j}}]^{x_{ID'}+h_{1'} \bmod p} \\ &= g^{\beta h'} \cdot [g^{h_{2'}} \cdot u^{b_{i,j}}]^{x_{ID'}+h_{1'} \bmod p}\end{aligned}$$

5) The simulator continues to interact with the adversary to conduct a remote data integrity audit program. In Game 2, if the adversary succeeds, but the generated sum signature θ_j' is not equal to the expected aggregation label, the game is aborted.

Suppose that the adversary has at most Q_L user ID queries, Q_{L_1} signature key pair queries, and Q_{L_2} data block queries forged the tuple $(\theta_j^*, U_{i,j}^{b_{i,j}^*}, b_{i,j}^*, ID^*, pk_{\mathrm{ID}^*})$, which is different from the expected tuple. The adversary sends it to the simulator, and the simulator can get:

$$e(\theta_j^*, g) = e(H(ID^*),\mathrm{mpk}) \cdot e(\prod_{i=1}^{t} H_2(U_{i,j}^{b_{i,j}^*}) \cdot u^{b_{i,j}^*}, \mathrm{pk}_{ID^*})$$

In other words, although $\theta_j^* \neq \theta_j$, it can still pass the verification. Then the simulator extracts $H(ID^*) = g^{\beta h^*}$ from the query list of L, and extracts $H_2(U_{i,j}^{b_{i,j}^*}) = g^{h_2^*}$ from the query list of L_2, and then can get that

$$\begin{aligned}e(\theta_j^*, g) &= e(g^{\beta h^*}, g^{\alpha}) \cdot e(g^{h_2^*} \cdot u^{b_{i,j}^*}, g^{sk_{ID}^*}) \\ e(\theta_j^*, g) &= e(g^{h^*}, g)^{\alpha\beta} \cdot e(g^{h_2^*} \cdot u^{b_{i,j}^*}, g)^{sk_{ID}^*} \\ e(g^{h^*}, g)^{\alpha\beta} &= e(\theta_j^*, g)/e(g^{h_2^* \cdot sk_{ID}^*} \cdot u^{b_{i,j}^*}, g)\end{aligned}$$

Obviously, according to the nature of bilinear pairing, and can further imply that

$$g^{\alpha\beta} = (\theta_j^* / pk_{ID^*}^{h_2^*} \cdot u^{b_{i,j}^*})^{-h^*}$$

Next, the probability can be calculated that the simulator will get the correct result. Suppose here that the adversary can win in Game 2 with a non-negligible probability advantage ε in time t, then the success probability of the interaction between the simulator and the adversary is better than $(1-\rho)^{Q_L Q_{L_1} Q_{L_2}}$. At this time, the probability of success of the simulator is $\varepsilon' = \varepsilon \cdot (1-\rho)^{Q_L Q_{L_1} Q_{L_2}} \rho$, when $\rho = 1/(Q_L Q_{L_1} Q_{L_2} + 1)$, ε' reaches the maximum value: $\varepsilon' = \varepsilon \cdot (Q_L Q_{L_1} Q_{L_2}/(Q_L Q_{L_1} Q_{L_2}+1))^{Q_L Q_{L_1} Q_{L_2}} \cdot 1/(Q_L Q_{L_1} Q_{L_2}+1)$ When $Q_L Q_{L_1} Q_{L_2}$ is large enough, $(Q_L Q_{L_1} Q_{L_2}/(Q_L Q_{L_1} Q_{L_2}+1))^{Q_L Q_{L_1} Q_{L_2}}$ is close to e^{-1}, so the probability of success of the simulator is $\varepsilon' \geq \varepsilon/e(Q_L Q_{L_1} Q_{L_2} + 1)$. Next, it need to calculate the total time required for the simulator to perform calculations, including the time required for the adversary to query the simulator to answer $Q_L cG_1 + Q_{L_1} cG_1 + Q_{L_2} cG_1$, where cG_1 represents the time spent in exponentiation in G_1 and the adversary forges the time it takes to sign t. The simulator can solve the CDH problem within time $t' \leq t + Q_L cG_1 + Q_{L_1} cG_1 + Q_{L_2} cG_1$ with probability $\varepsilon' \geq \varepsilon/e(Q_L Q_{L_1} Q_{L_2} + 1)$.

Game 3. The Game 3 and the Game 2 are the same, only the way the emulator sets the public key is different, and the audit scheme for each Game 2 is still retained and analyzed.

Analysis. Assuming that the adversary can win Game 2 with a non-negligible probability, another simulator can be constructed to solve the CDH problem. This simulator gives $g, g^{\alpha}, g^{\beta} \in G_1$, and the simulator calculates $g^{\alpha\beta}$ according to each instance.

1) The simulator randomly selects a generator g in G_1, and then randomly selects an element $x \in Z_p^*$, and sets it to msk, and calculates the corresponding system public key g^x.
2) The adversary selects user ID to adaptively execute H query. If this ID belongs to H-query list $L = \{ID, h, ssk\}$, the simulator extracts the corresponding information, including $ssk = H(ID) = g^h$. Otherwise, the simulator randomly chooses a value $h' \in Z_p^*$, $ssk' = H(ID') = g^{h'}$. Then the simulator returns ssk' to the adversary and adds (ID', h', ssk') to the query list.
3) The adversary runs and maintains the query list L_1. If the ID' already belongs to L_1., the simulator extracts the tuple $(ID', h_1', ssk_{ID}', sk_{ID}', pk_{ID}')$. If the relevant information is not found in L_1, the simulator selects a $h_1' \in Z_p^*$ and $x \in Z_p^*$, and then sets $x_{ID'} = x$. The simulator calculates $sk_{ID'} = x_{ID'} + h_1' \bmod p$ and $pk_{ID'} = (g^\alpha)^{(x_{ID'}+h_1') \bmod p}$, then adds these corresponding parameters to the tuple, and finally sends $pk_{ID'}$ to the adversary.
4) The adversary maintains the query list L_2. The simulator also maintains a query list $L_2 = (U_{i,j}, h_2)$. If the simulator wants to verify the data block $b_{i,j}$, it should query the corresponding block from the list and extract the relevant parameters first. The probability of successful query is p. Then the probability of not belonging to L_2 is 1-p. At this time, the simulator randomly selects a $h_2' \in Z_p^*$, then sends $g^{h_2'}$ to the adversary, and at the same time updates $(U_{i,j}', h_2')$ to the query list L_2. The simulator can calculate $\sigma_{i,j}$ for the data block $b_{i,j}$.

$$\begin{aligned}\sigma_{i,j} &= sk_1 \cdot [H_2(U_{i,j}^{b_{i,j}}) \cdot u^{b_{i,j}}]^{sk_2} \\ &= ssk \cdot [g^{\beta h_2'} \cdot u^{b_{i,j}}]^{x_{ID'}+h_1' \bmod p} \\ &= g^{h'} \cdot [g^{\beta h_2'} \cdot u^{b_{i,j}}]^{x_{\mathrm{ID}'}+h_1' \bmod p}\end{aligned}$$

5) The simulator continues to interact with the adversary to conduct a remote data integrity audit scheme. In Game 3, if the adversary succeeds and the generated sum signature θ_j' is not equal to the expected aggregation signature, the game is aborted.
Suppose that the adversary has at most T_L user ID queries, T_{L_1} signature key pair queries, and T_{L_2} data block queries forged tuples $(\theta_j^*, U_{i,j}^{b_{i,j}^*}, b_{i,j}^*, ID^*, pk_{ID^*})$. The adversary sends it to the simulator, and the simulator can get:

$$e(\theta_j^*, g) = e(H(ID^*), mpk) \cdot e(\prod_{i=1}^{t} H_2(U_{i,j}^{b_{i,j}^*}) \cdot u^{b_{i,j}^*}, pk_{ID^*})$$

Although $\theta_j^* \neq \theta_j$, it can still be verified. The simulator extracts $H(ID^*) = g^{h^*}$ from the query list of L, extracts $pk_{ID'} = (g^\alpha)^{(x_{ID'}+h_1') \bmod p}$ and $H_2(U_{i,j}^{b_{i,j}^*}) = g^{\beta h_2^*}$ from the query list of L_2. From the correctness of the scheme, the following verification equation can be obtained:

$$\begin{aligned}e(\theta_j^*, g) &= e(g^{h^*}, g^x) \\ &\quad \cdot e(g^{\beta h_2^*} \cdot u^{b_{i,j}^*}, g^{sk_{ID}^*}) \\ e(\theta_j^*, g) &= e(g^{h^*}, g^x) \\ &\quad \cdot e(g^{\beta h_2^*} \cdot u^{b_{i,j}^*}, \\ &\quad (g^\alpha)^{(x_{ID'}+h_1') \bmod p}) \\ e(g^{\alpha\beta h_2^* \cdot (x_{ID'}+h_1') \bmod p} \cdot u^{b_{i,j}^*}, g) &= e(\theta_j^*, g)/e(g^{xh^*}, g)\end{aligned}$$

According to the nature of bilinear pairing, $g^{\alpha\beta} = (\theta_j^*/g^{xh^*} \cdot u^{b_{i,j}^*})^{-h_2^* \cdot (x_{ID'}+h_1') \bmod p}$ can be obtained. Assuming that the adversary can win game 3 with a non-negligible probability advantage ε within time t, then the success probability of the interaction between the simulator and the adversary is better than $(1-\rho)^{T_L T_{L_1} T_{L_2}}$. At this time, the probability of success of the simulator is $\varepsilon' = \varepsilon \cdot (1-\rho)^{T_L T_{L_1} T_{L_2}} \rho$, when $\rho = 1/(T_L T_{L_1} T_{L_2} + 1)$, ε' reaches the maximum value:

$$\varepsilon' = \varepsilon \cdot (T_L T_{L_1} T_{L_2}/(T_L T_{L_1} T_{L_2}+1))^{T_L T_{L_1} T_{L_2}} \cdot 1/(T_L T_{L_1} T_{L_2}+1)$$

When $T_L T_{L_1} T_{L_2}$ is large enough, $(T_L T_{L_1} T_{L_2}/(T_L T_{L_1} T_{L_2} + 1))^{T_L T_{L_1} T_{L_2}}$ is close to e^{-1}, so the probability of success of the simulator is $\varepsilon' \geq \varepsilon/e(T_L T_{L_1} T_{L_2} + 1)$. It calculate the total time spent by the simulator in the calculation, which includes: the time required for the adversary to query the simulator to answer $T_L cG_1 + T_{L_1} cG_1 + T_{L_2} cG_1$ and the time t spent by the adversary forging the signature

In other words, the simulator can solve the CDH problem with probability $\varepsilon' \geq \varepsilon/e(T_L T_{L_1} T_{L_2} + 1)$ and time $t' \leq t + T_L cG_1 + T_{L_1} cG_1 + T_{L_2} cG_1$.

From the perspective of Game 1, Game 2 and Game 3, as long as the adversary forges a legal signature, an algorithm is needed to solve the CDH problem with a non-negligible probability. The calculation of the assumption of the CDH problem is a difficult problem, so it is impossible to forge a legal signature in this scheme.

V. EFFICIENCY ANALYSIS

In this section, our scheme is compared with other data integrity verification schemes such as schemes [20], [23], [31]. Scheme [20] is a dynamic operation scheme based on MHT, scheme [23] is a multi-copy data integrity verification based on spatiotemporal chaos and scheme [31] is a multi-copy data integrity verification based on ID signature. Therefore, it is a good contrast to compare these schemes we chose with our proposed scheme.

A. PERFORMANCE ANALYSIS AND COMPARISON

Table 2 shows the symbols and meanings used in the following analysis.

Suppose m as the number of copies, c as the number of data blocks in the audit challenge, n as the total number of data blocks, s representing the number of sectors in each block, $|F|$ representing the size of the copy file, δ representing the number of user update database blocks. In keeping with the

TABLE 2. Symbols and their meanings.

Symbol	Meaning	Symbol	Meaning
$HashG_1$	$H:\{0,1\}^* \rightarrow G_1$	$Add_{Z_p^*}$	Addition operation
$ExpG_1$	Exponential operation on G_1	$Pair$	Bilinear pairing
$MulG_1$	Multiply operation on G_1	$\|q\|$	Bit length of elements in G_1
$\|p\|$	Bit length of elements in Z_p^*		

TABLE 3. Calculation time of each data block node.

Number of blocks	[20]	[23]	[31]	*Our scheme*
100	0.0434	0.0398	0.0315	0.0287
200	0.0882	0.0812	0.0628	0.0568
1000	0.4358	0.3971	0.3157	0.2868

principle of generality, it is assumed that the required security level is 128 bits. We use the elliptic curve of $|p| = 256$ bits defined by the Galois field. The points on this curve can be represented by 257 bits, and the hash function h_{SHA} used in this scheme is SHA-1, and the length of its hash value is 160 bits.

1) COMPUTATIONAL OVERHEAD

It is listed the time consumption of computing nodes and the computing overhead of metadata in Table 3. It can be seen that the computing overhead of our scheme is far less than other schemes. As the number of copies increases, the gap will gradually increase.

We compared the computational cost in the three stages of label generation, proof generation, and evidence verification. These stages consume the most resources in the scheme and are the most comparative. The comparison results of the three schemes will show in Table 4.

2) STORAGE OVERHEAD

The storage cost mainly includes the storage cost of file copies, the storage cost of corresponding signature of file copies and the cost of information storage such as verification cost. Among them, the storage cost of file copies and the corresponding signature cost of file copies account for a large proportion. Scheme [20] uses Merkle tree to store data files. Each tree stores a file copy, and each leaf node stores a data block. Therefore, there are as many leaf nodes as there are blocks in a copy. In the case of large data blocks, the structure of the tree will be very huge, and its size is larger than $m|F|$. In our scheme, the red-black tree is used to store data files and signature files. Each leaf node and non-leaf node can store data blocks, so the overall tree structure is much smaller and the efficiency improves greatly. Although it is necessary to construct two red-black trees, in one of the trees, only the data block serial number and node value are stored, and the storage overhead is very small and can be ignored. The size of the two trees is basically equal to $m|F|$. Our scheme needs to generate n file tags, each of which is an element on G_1, so its storage cost is $n|G_1|$. Here it use a file F with a size of 100MB for analysis. Its block size is 4KB, which can be divided into 25600 blocks, and then the element size $|q|$ of the G_1 elliptic curve group is selected as 257 bits. From the above calculation, the overhead required to store the tag is 803KB, and then it can be calculated that the generated tag to the entire file account for 803KB/100MB ≈0.7841%, which is a small proportion for the entire file.

3) COMMUNICATION OVERHEAD

The communication overhead of the integrity verification scheme generally includes: passing a random challenge *chal*, responding to a challenge, data integrity certification, and communication overhead in the update phase. For the number c of data blocks users want to challenge, it challenges $chal = \{j, v_j\}_{\gamma_1 \le j \le \gamma_c}$, where j is the $log_2(c)$ calculated from the pseudo-random sequence and $v_j \in Z_p^*$. In the response challenge phase $P = \{\mu, \theta, \{b_{i,j}, \kappa_{i,j}\}_{1 \le i \le t, \gamma_1 \le j \le \gamma_c}\}$ of our proposed scheme, where $\mu, b_{i,j} \in Z_p^*$, $\theta \in G_1$ and $\kappa_{i,j}$ is a cryptographic hash value with a length of $log_2(n)$. The result compared with other schemes are shown in Table 5.

B. EXPERIMENTAL RESULT

In this section, we will evaluate our scheme through experiments. Most of the experiments are run on Vmware Workstation Pro, and its configuration is 4G RAM and 20G ROM. The operating system selected on the workstation is ubuntu 20.04.1. Vmware Workstation Pro is configured on a Dell computer, its core is i5-8300H@2.8GHz CPU and its RAM is 8G. The network used is the school campus network, the code mainly uses the programming language C language with the free Pairing-Based Cryptography (PBC) library [34] and GNU Multiple Precision Arithmetic (GMP) [35]. In the experiment, we set the basic field size to 512 bits, the bit length of element is $|p| = 256$ bits. The bit length of the element in G_1 is $|q| = 257$ bits. The data file size is 20MB, which is divided into 1×10^6 blocks. The bit length of the identification code of each data block is 80 bits.

The computational cost of signature generation will be evaluated. In the experiment, different signatures are generated for different data blocks with an interval of 100 from 0 to 1000, and each block has 5 copies, which are divided into 20 sectors. As shown in Figure 9, the proposed scheme is compared with other schemes, and the signature generation increases with the increase of data blocks, and it grows linearly.

Our scheme also compared with other schemes the integrity audit overhead of CSP for 100 to 1000 challenged

TABLE 4. Computational overhead comparison.

	[20]	[23]	[31]	*Our scheme*
Tag – Gen	$(s+1)mnExp_{G_1}$ $+(sm+m-1)nMul_{G_1}+mnH_{G_1}$	$(s+1)mnExp_{G_1}$ $+mnMul_{G_1}+mnH_{G_1}$	$(s+1)mnExp_{G_1}$ $+2mnMul_{G_1}+mnH_{G_1}$	$2mnExp_{G_1}$ $+2mnMul_{G_1}+2mnH_{G_1}$
Proof -gen	$cExp_{G_1}+(c-1)Mul_{G_1}+csmMul_{Z_p^*}$ $+(c-1)smAdd+cmH_{G_1}$	$cExp_{G_1}+(c-1)Mul_{G_1}$ $+csmMul_{Z_p^*}+csmAdd$	$cExp_{G_1}+(c-1)Mul_{G_1}$ $+csmMul_{Z_p^*}+(c-1)smAdd$	$cExp_{G_1}+(c-1)Mul_{G_1}$ $+csmMul_{Z_p^*}+(c-1)smAdd$
Proof -verify	$2pair+(mc+s)Exp_{G_1}$ $+(mc+s-1)Mul_{G_1}$ $+(s-1)Add+cH_{G_1}$	$2pair+(2m+1)cExp_{G_1}$ $+(2mc+c+s+1)Mul_{G_1}$ $+sAdd+(c+1)mH_{G_1}$	$3pair+(mc+s+3)Exp_{G_1}$ $+(mc+s)Mul_{G_1}$ $+sAdd+cmH_{G_1}$	$3pair+(mc+2)Exp_{G_1}$ $+(mc+s)Mul_{G_1}$ $+(s-1)Add$ $+(cm+1)H_{G_1}$
Multi-copy	*Yes*	*Yes*	*Yes*	*Yes*
ID-Based	*No*	*No*	*Yes*	*Yes*

TABLE 5. Communication overhead comparison.

	[20]	[23]	[31]	*Our scheme*
Challenge	$\|q\|+log_2(c)$	$\|q\|+log_2(c)$	$\|q\|+log_2(c)$	$\|q\|+log_2(c)$
Response	$ms\|q\|+(1+cm)\|p\|$ $+log_2(n)$	$(1+ms)\|q\|+(1+cm)\|p\|$ $+log_2(n)$	$2ms\|q\|+3ms\|p\|$ $+log_2(n)$	$m\|q\|+(1+cm)\|p\|$ $+log_2(n)$
Total	$\|q\|+log_2(c)$ $+ms\|q\|$ $+(1+cm)\|p\|$ $+log_2(n)$	$\|q\|+log_2(c)$ $(1+ms)\|q\|+(1+cm)\|p\|$ $+log_2(n)$	$\|q\|+log_2(c)$ $+2ms\|q\|+3ms\|p\|$ $+log_2(n)$	$\|q\|+log_2(c)$ $+m\|q\|+(1+cm)\|p\|$ $+log_2(n)$
Update	$\delta\|p\|(log_2(c)+1)$ $+\|p\|$	$\delta\|p\|(log_2(c)+1)$ $+\|p\|$	-	$\delta\|p\|log_2(c)$ $+3log_2(c)+\|p\|$

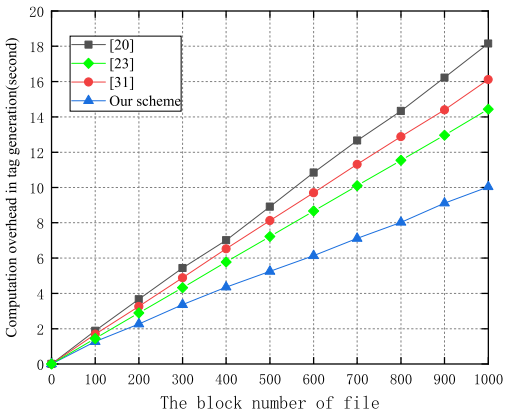

FIGURE 9. The computational overhead of label generation.

data blocks. Figure 10 shows the calculation cost of evidence generation. It can be seen from the figure that our scheme, scheme [20], scheme [23] and scheme [31] have a linear relationship with the number of challenged data blocks in the evidence generation stage, but our scheme overall more efficient.

After the calculation cost is evaluated, the communication cost in the certification phase and the update phase will be evaluated. As mentioned by Shen *et al.* in the scheme [28], the communication overhead in the challenge generation, proof generation and integrity verification phases are all linearly related to the challenge request c. As c gets larger, the accuracy of the data gets higher. But the increase will also cause the system to increase the communication overhead. Therefore, the value of c is selected on the basis of balancing the verification accuracy and the communication overhead. According to experiments, when $c = 460$, the verification accuracy can reach 99% that the communication overhead is not large. As shown in Figure 11, when c = 460 is selected, the communication overhead of each scheme is compared.

Then the communication overhead in the update phase will be evaluated. Barsoum mentioned in scheme [20] that update operations such as inserting and deleting data are engineering issues. For different problems, there are different

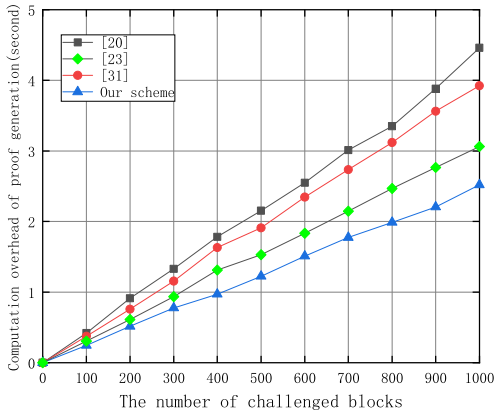

FIGURE 10. The computational overhead of evidence generation.

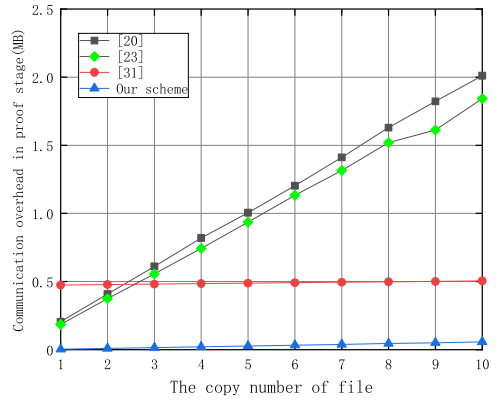

FIGURE 11. The communication overhead of integrity verification.

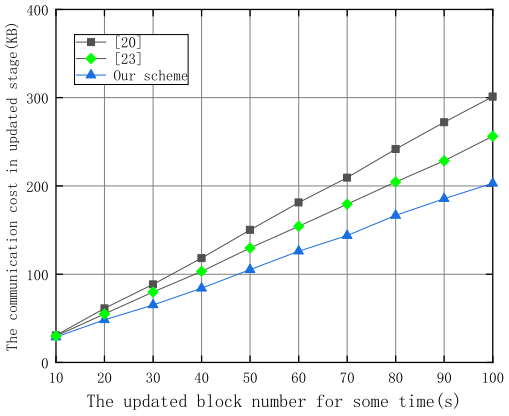

FIGURE 12. The communication overhead of integrity verification.

insertion operations such as inserting and deleting data are engineering issues. For different problems, there are different insertion schemes, and the most standard balanced binary tree mode is used. Our scheme adopts the red-black tree data structure, after inserting the data node, both of them only need two rotation operations at most. When deleting data nodes, the scheme [20] and scheme [23] requires $O(logN)$ times in the worst case, but the red-black tree in our scheme only needs three times. As shown in Figure 12, our scheme is obviously more efficient than the scheme [20] and scheme [23] when a large number of insert and delete operations are required.

VI. CONCLUSION

This paper proposes an effective multiple copies data integrity verification scheme based on red-black tree in cloud storage. It supports dynamic operations on multiple copies, which can improve efficiency. In terms of data storage, the red-black tree data structure storage is adopted to effectively improve data storage efficiency and simplify data update operations. The theoretical analysis of our scheme also proves the security of the scheme. The experimental results also show that our scheme is superior to other comparative schemes in terms of computational cost, storage cost, and communication cost. The experimental analysis demonstrate that our scheme achieves desirable security and efficiency.

REFERENCES

[1] W. Shen, J. Yu, H. Xia, H. Zhang, X. Lu, and R. Hao, "Light-weight and privacy-preserving secure cloud auditing scheme for group users via the third party medium," *J. Netw. Comput. Appl.*, vol. 82, pp. 56–64, Mar. 2017.

[2] M. Alshehri, A. Bhardwaj, M. Kumar, S. Mishra, and J. Gyani, "Cloud and IoT based smart architecture for desalination water treatment," *Environ. Res.*, vol. 195, Apr. 2021, Art. no. 110812, doi: 10.1016/j.envres.2021.110812.

[3] S. Srivastava, S. Saxena, R. Buyya, M. Kumar, A. Shankar, and B. Bhushan, "CGP: Cluster-based gossip protocol for dynamic resource environment in cloud," *Simul. Model. Pract. Theory*, vol. 108, Apr. 2021, Art. no. 102275, doi: 10.1016/j.simpat.2021.102275.

[4] K. He, J. Chen, Q. Yuan, S. Ji, D. He, and R. Du, "Dynamic group-oriented provable data possession in the cloud," *IEEE Trans. Dependable Secure Comput.*, early access, Jul. 2, 2019, doi: 10.1109/TDSC.2019.2925800.

[5] M. Kumar, M. Alshehri, R. AlGhamdi, P. Sharma, and V. Deep, "A DE-ANN inspired skin cancer detection approach using fuzzy C-means clustering," *Mobile Netw. Appl.*, vol. 25, no. 4, pp. 1319–1329, Aug. 2020, doi: 10.1007/s11036-020-01550-2.

[6] L. Zhou, A. Fu, G. Yang, H. Wang, and Y. Zhang, "Efficient certificateless multi-copy integrity auditing scheme supporting data dynamics," *IEEE Trans. Dependable Secure Comput.*, early access, Aug. 4, 2020, doi: 10.1109/TDSC.2020.3013927.

[7] C. Dhasarathan, M. Kumar, A. K. Srivastava, F. Al-Turjman, A. Shankar, and M. Kumar, "A bio-inspired privacy-preserving framework for healthcare systems," *J. Supercomput.*, early access, Mar. 19, 2021, doi: 10.1007/s11227-021-03720-9.

[8] L. Krithikashree and S. Manisha, "Audit cloud: Ensuring data integrity for mobile devices in cloud storage," *IEEE Trans. Depend. Sec. Comput.*, pp. 1–5, Sep. 2018, doi: 10.1109/ICCCNT.2018.8493963.

[9] L. Deng, B. Yang, and X. Wang, "A lightweight identity-based remote data auditing scheme for cloud storage," *IEEE Access*, vol. 8, pp. 206396–206405, 2020, doi: 10.1109/ACCESS.2020.3037696.

[10] S. Peng, F. Zhou, Q. Wang, Z. Xu, and J. Xu, "Identity-based public multi-replica provable data possession," *IEEE Access*, vol. 5, pp. 26990–27001, 2017, doi: 10.1109/ACCESS.2017.2776275.

[11] A. Bhardwaj, S. B. H. Shah, A. Shankar, M. Alazab, M. Kumar, and T. R. Gadekallu, "Penetration testing framework for smart contract blockchain," *Peer-Peer Netw. Appl.*, early access, Sep. 5, 2020, doi: 10.1007/s12083-020-00991-6.

[12] P. Shen, C. Li, and Z. Zhang, "Research on integrity check method of cloud storage multi-copy data based on multi-agent," *IEEE Transl. Content Mining*, vol. 4, no. 8, pp. 17170–17178, 2020, doi: 10.1109/ACCESS.2020.2966803.

[13] Y. Luo, M. Xu, S. Fu, D. Wang, and J. Deng, "Efficient integrity auditing for shared data in the cloud with secure user revocation," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2015, pp. 434–442, doi: 10.1109/Trustcom.2015.404.

[14] R. Rabaninejad, S. M. Sedaghat, M. Ahmadian Attari, and M. R. Aref, "An ID-based privacy-preserving integrity verification of shared data over untrusted cloud," in *Proc. 25th Int. Comput. Conf., Comput. Soc. Iran (CSICC)*, Jan. 2020, pp. 1–6, doi: 10.1109/CSICC49403.2020.9050098.

[15] C.-T. Hunag, C.-Y. Yang, C.-Y. Weng, Y.-W. Chen, and S.-J. Wang, "Secure protocol for identity-based provable data possession in cloud storage," in *Proc. IEEE 4th Int. Conf. Comput. Commun. Syst. (ICCCS)*, Feb. 2019, pp. 327–331, doi: 10.1109/CCOMS.2019.8821766.
[16] C. Liu, J. Chen, L. Yang, X. Zhang, and R. Kotagiri, "Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2234–2244, Sep. 2014.
[17] H. Jin, H. Jiang, and K. Zhou, "Dynamic and public auditing with fair arbitration for cloud data," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 680–693, Jul. 2018, doi: 10.1109/TCC.2016.2525998.
[18] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. 14th ACM Conf. Comput. Commun. Secur. (CCS)*, 2007, pp. 598–609.
[19] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 847–859, May 2011.
[20] A. F. Barsoum and M. A. Hasan, "Provable multicopy dynamic data possession in cloud computing systems," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 3, pp. 485–497, Mar. 2015.
[21] M. Sookhak, F. R. Yu, and A. Y. Zomaya, "Auditing big data storage in cloud computing using divide and conquer tables," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 5, pp. 999–1012, May 2018.
[22] X. Yang, T. Liu, P. Yang, F. An, M. Yang, and L. Xiao, "Public auditing scheme for cloud data with user revocation and data dynamics," in *Proc. IEEE 2nd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Dec. 2017, pp. 813–817, doi: 10.1109/ITNEC.2017.8284847.
[23] M. Long, Y. Li, and F. Peng, "Integrity verification for multiple data copies in cloud storage based on spatiotemporal chaos," *Int. J. Bifurcation Chaos*, vol. 27, no. 4, Apr. 2017, Art. no. 1750054.
[24] M. Long, Y. Li, and F. Peng, "Dynamic provable data possession of multiple copies in cloud storage based on full-node of AVL tree," *Int. J. Digit. Crime Forensics*, vol. 11, no. 1, pp. 126–137, Jan. 2019.
[25] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Proc. 28th Int. Conf. Distrib. Comput. Syst.*, Jun. 2008, pp. 411–420.
[26] A. Barsoum and M. Hasan, "On verifying dynamic multiple data copies over cloud servers," Dept. Elect. Comput. Eng., Tech. Rep., 2011.
[27] H. Shacham and B. Waters, "Compact proofs of retrievability," *J. Cryptol.*, vol. 26, no. 3, pp. 442–483, Jul. 2013.
[28] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 10, pp. 2402–2415, Oct. 2017.
[29] W. Shen, J. Qin, J. Yu, R. Hao, and J. Hu, "Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 2, pp. 331–346, Feb. 2019, doi: 10.1109/TIFS.2018.2850312.
[30] Y. Zhang, J. Ni, X. Tao, Y. Wang, and Y. Yu, "Provable multiple replication data possession with full dynamics for secure cloud storage," *Concurrency Comput., Pract. Exper.*, vol. 28, no. 4, pp. 1161–1173, Mar. 2016, doi: 10.1002/cpe.3573.
[31] J. Li, H. Yan, and Y. Zhang, "Efficient identity-based provable multi-copy data possession in multi-cloud storage," *IEEE Trans. Cloud Comput.*, early access, Jul. 16, 2019, doi: 10.1109/TCC.2019.2929045.
[32] T. Wu, G. Yang, Y. Mu, F. Guo, and R. H. Deng, "Privacy-preserving proof of storage for the pay-as-you-go business model," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 2, pp. 563–575, Mar. 2021, doi: 10.1109/TDSC.2019.2931193.
[33] J. Li, J. Li, D. Xie, and Z. Cai, "Secure auditing and deduplicating data in cloud," *IEEE Trans. Comput.*, vol. 65, no. 8, pp. 2386–2396, Aug. 2016.
[34] *The Pairing-Based Cryptography(PBC) Library*. Accessed: 2020. [Online]. Available: https://crpto.stanford.edu/pbc/download.html
[35] *The GNU Multiple Precision Arithmetic Library (GMP)*. Accessed: 2020. [Online]. Available: http://gmplib.org/

ZHENPENG LIU received the B.S. and M.S. degrees from the School of Cyberspace Security and Computer, Hebei University, and the Ph.D. degree from Tianjin University. He is currently a Professor with the School of Cyberspace Security and Computer, Hebei University. His research interests include big data, cloud computing, and information security research.

YI LIU is currently pursuing the M.S. degree with the School of Cyberspace Security and Computer, Hebei University, Baoding, Hebei. His research interests include cloud computing security, privacy protection, and data integrity verification.

XIANWEI YANG is currently pursuing the M.S. degree with the School of Cyberspace Security and Computer, Hebei University, Baoding, Hebei. His research interests include privacy protection, redundant variant research, and data integrity verification.

XIAOFEI LI received the M.S. degree from the School of Cyberspace Security and Computer, Hebei University. She is currently an Engineer with the Information Technology Center, Hebei University. Her main research interests include big data, cloud computing, and information security.