electronics

Article

Dynamic Data Integrity Auditing Based on Hierarchical Merkle Hash Tree in Cloud Storage

Zhenpeng Liu [1,2], Shuo Wang [1], Sichen Duan [1], Lele Ren [1] and Jianhang Wei [2,3,*]

1 School of Cyberspace Security and Computer, Hebei University, Baoding 071002, China
2 Information Technology Center, Hebei University, Baoding 071002, China
3 Network and Experiment Management Center, Xinjiang University of Science & Technology, Korla 841000, China
* Correspondence: wei@hbu.edu.cn

Abstract: In cloud storage mode, users lose physical control over their data. To enhance the security of outsourced data, it is vital to audit the data integrity of the data owners. However, most of the current audit protocols have a single application scenario and cannot accommodate the actual needs of individuals and enterprises. In this research, a safe and efficient auditing scheme is proposed that is based on a hierarchical Merkle tree. On the one hand, we use a hierarchical authentication data structure and local signature aggregation technique to reduce the scale of the Merkle tree. In addition, authoritative nodes are introduced to reduce the length of the authentication path and improve the update efficiency. On the other hand, we introduce a monitoring mechanism that is based on the original data integrity auditing model to analyze the cloud data, which improves the transparency and credibility of cloud service providers. In addition, we achieve incomplete data recovery through log analysis, which greatly reduces the number of replicas of files under the premise of multi-copy auditing, reduces the burden on cloud service providers, and improves the fairness of audit protocols. The theoretical analysis and experimental comparison prove that the method is secure and efficient. It can effectively reduce the computational overhead and storage overhead in integrity auditing.

Keywords: data integrity; hierarchical Merkle tree; monitoring mechanism; data recovery; integrity auditing

check for updates

Citation: Liu, Z.; Wang, S.; Duan, S.; Ren, L.; Wei, J. Dynamic Data Integrity Auditing Based on Hierarchical Merkle Hash Tree in Cloud Storage. *Electronics* **2023**, *12*, 717. https://doi.org/10.3390/electronics12030717

Academic Editors: Jie Li, Xianfu Chen, Lei Shi, Yangjie Cao and Bo Zhang

Received: 4 January 2023
Revised: 26 January 2023
Accepted: 30 January 2023
Published: 1 February 2023

1. Introduction

With the rapid rise in cloud computing, cloud storage has increasingly taken over as the main method of storage in the big data era [1]. Cloud service providers (CSPs) enjoy very efficient computing power and huge storage capacity, meaning that they can provide services with strong scalability, low price, and access anytime and anywhere, which can significantly reduce the local storage pressure and data management burden of enterprises or individuals [2]. However, CSPs are not always trustworthy. While CSPs provide considerable convenience to users and corporations, they also pose potential threats to data storage security, resulting in some unavoidable security problems, such as network attacks, hardware failures, management errors, etc. [3]. When data owners (DOs) outsource data to the CSP, this also means that they lose physical control power over the data [4]. First, due to software failure or hardware failure, causing the data of users to be damaged or lost, the CSP might purposefully hide this data loss in order to protect its reputation. Secondly, the CSP may delete files that have never been accessed or that have a very low probability of access to reduce pressure on storage capacity. Lastly, unauthorized users maliciously tamper with or illegally process the data of users through network attacks [5]. Data-outsourcing security incidents have occurred on several occasions in recent years. Therefore, it is critical to adopt related technologies to verify the integrity of the outsourced data in the cloud uploaded by users [6]. For CSPs, how to obtain the trust of users is also a pressing issue that needs to be resolved.

To better protect the cloud data of users, domestic and foreign researchers have carried out a series of studies and have proposed a variety of data integrity auditing schemes, with functions such as dynamic update, privacy protection, and multi-copy verification [7]. However, with the continuous refinement of business requirements, the existing auditing schemes have become less and less able to satisfy the needs of users. Additionally, the majority of current schemes ignore the fact that the resources of CSPs are likewise constrained and instead concentrate more on saving the computing resources and storage capacity of users [8]. Nowadays, as the number of users and documents grows exponentially, cloud services also require more cost to maintain, which may create a bottleneck in the performance of cloud servers [9]. To reduce their maintenance costs, CSPs may betray the agreements signed with users and may adopt illegal means to achieve their goals, such as deleting part of the user data and reducing data copy storage. It also indirectly increases the distrust between users and CSPs [10]. Currently, most CSPs can only rely on copies for data recovery. If the CSP removes the replicas to save storage space, once the equipment fails or suffers from external attacks, the CSP will not be able to recover the data of users in time to prevent problems, causing huge losses to both parties in the agreement.

Because of the single and inefficient application scenarios of many integrity-auditing protocols, it is essential to improve the security of cloud data and promote mutual trust between users and CSPs. Therefore, we propose an integrity audit solution that is secure and efficient in terms of generic scenarios. The main work of this paper can be summarized in the following four points:

1. We designed a hierarchical Merkle hash tree (HMHT) structure with authoritative nodes. Based on the hierarchical principle, the HMHT can significantly decrease the number of invalid retrievals, shorten the authentication path of the nodes, and improve auditing efficiency and dynamic update efficiency.
2. We have introduced a monitoring mechanism to improve the transparency and credibility of the CSP, which can offer timely conduct violation analysis and the traceability of illegal operations.
3. We used monitoring logs to recover the damaged data, and to reduce the number of cloud data copies and redundancy. This can save a great deal in terms of storage resources and reduce the complexity of computation.
4. We analyzed the security and performance of this scheme through a series of experiments; the results prove that the solution is safe and effective in terms of communication cost and computational overhead.

The remainder of this essay is structured as follows: we provide a summary of other scholars' research in Section 2. In Section 3, we introduce some preliminaries and notations. We establish the system model and security objectives in Section 4. We describe in great depth the implementation of the solution in Section 5. In Section 6, we provide evidence of the scheme's security. In Section 7, we analyze the scheme's effectiveness by contrasting it with alternative schemes. Finally, we offer our conclusions for the entire study in Section 8.

2. Related Work

Data integrity auditing has been intensively investigated in recent years as users have become increasingly concerned about the privacy and security of their data in the cloud. Therefore, numerous auditing protocols have been set up. These solutions can be broadly divided into two categories: proofs of retrievability (POR) and provable data possession (PDP).

The POR mechanism was first proposed by Jules et al. [11]. In practical applications, the POR mechanism is mostly adopted in the integrity auditing of some essential information, such as confidential data from the military, government, biomedical and scientific research institutes, and other confidential units that are stored [12]. Because the POR mechanism can not only give access if the data in the cloud have already been corrupted or altered, but it can also partially recover the damaged data. The effectiveness of the POR mechanism is largely dependent on specific preprocessing operations performed by the data owner before uploading the data, such as inserting some "sentinels" into the sequence of the original data blocks and then fixing the corruption through error correction codes. However, the POR method does not allow for dynamic updating; in addition, the number of challenges that users can launch is also limited due to the limited number of "sentries". Shacham et al. [13] developed a novel POR mechanism for public verification, which can initiate any number of challenges. However, it does not permit the dynamic manipulation of data. Although the POR can recover data, it must download a great deal of data from the cloud to local storage for comparison and verification during the auditing process, which significantly increases the communication cost, computational overheads, storage burden, and network bandwidth resource consumption of the data owners.

The PDP mechanism was first suggested by Ateniese et al. [14] in 2007. This mechanism avoids data downloading and comparison by calculating tags or signatures on separate blocks of a file before they are uploaded. The user then uploads the signatures, together with the file, to the cloud, and only the signatures need to be verified during the auditing process. The approach also defines the concept of "probabilistic auditing", which can greatly reduce the computing burden and communication costs. Specifically, when 1% of the blocks are corrupted, only 4.6% of the total data blocks can be sampled to detect data corruption with a 99% probability, which considerably increases auditing efficiency. Based on the efficiency of the PDP mechanism, Wang et al. [15] proposed a public checking scheme, based on a third-party center, which effectively decreases the computational overhead of data owners in traditional private auditing. To achieve the dynamic processing of data, Wang et al. also introduced the Merkle tree model into their method. However, if the data volume increases too much, the verification path of the node will also become particularly long, which will reduce the audit efficiency. Erway et al. [16] proposed a scheme using a skip table (ST). Although this scheme can narrow the scope of data authentication, it also results in significant computational overhead. Shen et al. [17] provided a scheme based on a location array (LA) and a doubly linked information table (DLIT). This scheme can support both dynamic updates and batch auditing. However, when the files become more and more complex, the retrieval of data becomes difficult because of the characteristics of the linked list. Su et al. [18] developed a system based on a binary-ordered Merkle tree with a local authoritative root node, which can effectively shorten the authentication path of the data nodes. A position-aware Merkle hash tree (HMHT) was proposed by Hariharasitaraman et al. [19], which can judge the positional relationship between nodes in the tree and their parent nodes. However, if the amount of cloud data is too large, the structure of the PMHT will also become complex, resulting in reduced audit efficiency.

According to different application scenarios, domestic and foreign scholars have also proposed many characteristics of auditing protocols in recent years. For example, Shen et al. [20] proposed an identity-based scheme that realized confidential information-hiding and data sharing by introducing a disinfectant mechanism. Li et al. [21] used multi-copy distributed verification to provide a higher guarantee for the reliability of data. Liu et al. [22] proposed a red–black tree storage structure to improve auditing efficiency. Grag et al. [23] proposed simplifying the exponentiation in bilinear pairs to decrease the computational overheads. Zhou et al. [24] proposed a location Merkle tree and a local signature aggregation technique to improve the data update efficiency. Tian et al. [8] used Bell triangles to increase the correlation between signatures, but this also increased the complexity of the verification phase. Thangavel et al. [25] used a ternary hash tree to shorten the authentication path length, to reduce the computational overhead. Zhou et al. [26] realized the traceability of data changes through the adjacency list, based on a dynamic hash table. Xu et al. [27] proposed tag replacement technology to strengthen the protection of the private keys and signatures of users. Luo et al. [28] designed an MHB * tree in the research, which can improve file retrieval efficiency. Gudeme et al. [29] introduced a user revocation mechanism to enhance the protection of data in a shared environment. Li et al. [30] and Shen et al. [31] improved the security level of auditing protocols by fuzzing user identities with biometrics. Zhang et al. [32] and Huang et al. [33] applied blockchain technology to the verification mechanism to enhance the immutability and non-repudiation of data. Yang et al. [34] utilized a compressed storage protocol to reduce communication overhead and improve data storage efficiency.

To sum up, an auditing scheme is needed in general scenarios to improve both auditing efficiency and update efficiency; this can also resolve the trust and fairness issues of all parties involved in the protocol, to enhance the security of the protocol.

3. Preliminaries

3.1. Bilinear Map

Let G_1 and G_2 be multiplicative cyclic groups of order p, where p is a sufficiently large prime number, and g is a generator of G_1; then, the bilinear map e: $G_1 \times G_1 \rightarrow G_2$ has the following properties:

- Computability: for any $x_1, y_1 \in G_1$, there is always an efficient algorithm to calculate $e(x_1, y_1)$.
- Bilinearity: for any $x_2, y_2 \in G_1$ and $\alpha, \beta \in Z_p{}^*$, and $e(x_2^{\alpha}, y_2^{\beta}) = e(x_2, y_2)^{\alpha\beta}$.
- Non-degeneracy: for any $x_3, y_3 \in G_1$, and $e(x_3, y_3) \neq 1$.

3.2. The Computational Diffie–Hellman (CDH) Problem

For any unknown x, $y \in Z_p{}^*$, and $u \in G_1$, it is computationally infeasible to output $u^{xy} \in G_1$ with the triplet (u, u^x, u^y) as an input. In other words, the probability of cracking u^{xy} in G_1 is negligible.

3.3. Notations

Table 1 describes the notations used in the proposed scheme.

Table 1. Notations in the proposed scheme.

Notation	Description
p	Large prime number
G, G_T	Multiplicative cyclic group of order p
e	Bilinear map
$Z_p{}^*$	$\{1, 2, \ldots, p-1\}$
H_1, H_2, H_3	Three hash functions
M	System public parameters
$F = \{b_1, b_2, \ldots, b_n\}$	The original file
$F^* = \{m_{ij}\}_{1 \le i \le n, 1 \le j \le m}$	The encrypted file copies
g	A generator of G
u	Random elements in G
n	The number of file blocks
m	The number of file copies
$\{\sigma_i\}$	Signature collection of the file blocks
C	Authentication tree information
$chal$	Challenge
$proof$	The evidence information
$f(v_R), f(v_{SR}), f(v_i)$	Node hash value
$OP_{modify}, OP_{insert}, OP_{delete}$	File update operators

4. Problem Statement

4.1. System Model

The data integrity verification system model proposed in this essay is shown in Figure 1, which includes four different entities: the data owner (DO), third-party monitoring center (TPM), third-party audit center (TPA), and cloud service provider (CSP).

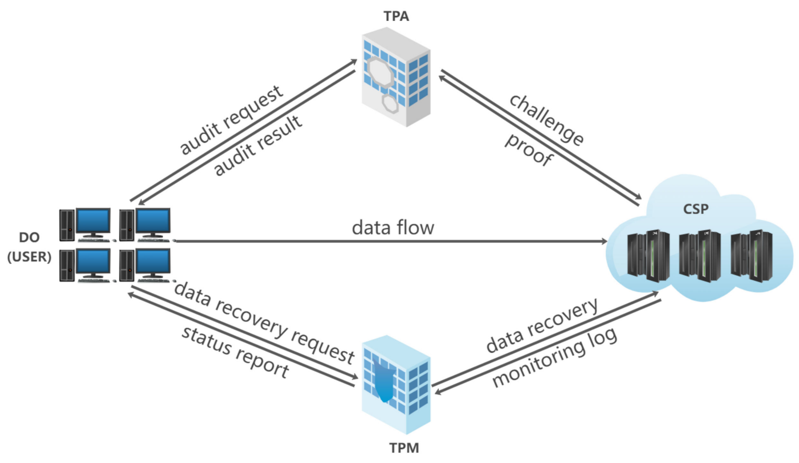

Figure 1. System model of the scheme.

The DO updates data to the cloud, after which the CSP stores, manages, and maintains the data. The DO can access and update their own cloud data in real time. However, the computing power of the DO is relatively limited. To reduce the computational burden on the user side, the DO entrusts the integrity verification task to a trusted TPA, which has professional audit capabilities. The TPA periodically issues audit challenges to the CSP. The CSP calculates the corresponding evidence parameters when it receives the audit request and then responds to the TPA. The TPA verifies the correctness of the evidence and feeds the result back to the DO honestly. After the DO transfers his or her data to the cloud server, the TPM will monitor the DO's files comprehensively and record the operation records of each file. When the data integrity verification fails, the TPM recovers the incomplete data, according to the monitoring log, traces any abnormal behavior, determines whether the data corruption is caused by accident or caused by the illegal operation of the malicious user, and implements arbitration against the malicious user. In the entire process, the algorithms involved are as follows:

1. KeyGen(1^k) $\rightarrow$ (sk, M). The key generation algorithm is executed by the DO, with an input security parameter, k, output private key, sk, and system parameter, M.
2. RepGen(F, m) $\rightarrow$ F^*. The replica generation algorithm is executed by the DO. We input the original file F and the copy number m, and output the encrypted copy block set $F^* = \{m_{ij}\}_{1 \le i \le n, 1 \le j \le m}$.
3. SigGen(F^*, sk) $\rightarrow$ $\{\sigma_i\}$. The signature algorithm is executed by the DO. We input the replica block set F^* and private key sk and output the aggregate signature $\{\sigma_i\}_{1 \le i \le n}$ of all replica blocks.
4. Store(F^*, M, $\{\sigma_i\}$, C) $\rightarrow$ $valid$. The storage algorithm is executed by the CSP. The input encrypted files are F^* and other authentication parameters, along with the output storage results.
5. ChalGen(c) $\rightarrow$ $chal$. The challenge algorithm is executed by the TPA. Input challenge parameter c, output random challenge information $chal$.
6. ProofGen($chal$) $\rightarrow$ $proof$. The evidence generation algorithm is executed by the CSP. Input challenge information is $chal$, and output evidence information is $proof$.
7. ProofVerify($proof$, pk) $\rightarrow$ $valid$. The evidence verification algorithm is executed by the TPA. Input the audit evidence $proof$ and public key, pk, to output the integrity verification result.
8. UpdateVerify($proof$) $\rightarrow$ $valid$. The update verification algorithm is executed by the TPA. We input the update evidence, $proof$, and public key, pk, and output the update result.
9. DataRecover(U_{id}, $\{F_{id}$, $\{b_{id}$, $\{W_{type}$, $offset$, $data\}\}$) $\rightarrow$ (U_{id}, $\{F_{id}\}$). The data recovery algorithm is executed by the TPM. We input the user ID and the damaged file parameters, along with the output metadata information of the recovered files.

4.2. Security Goals

In order to provide more reliable and effective data integrity auditing, the designed method should achieve the necessary security objectives:

1. Public auditing: The integrity of cloud data can be verified by the TPA on behalf of the DO.
2. Storage integrity: Only when the CSP stores the data correctly can it pass the verification of the TPA.
3. Batch auditing: The TPA has the ability to verify the data of different numbers of users or files at the same time.
4. Privacy protection: The data are stored in the cloud in the form of ciphertext, wherein only the DO has the ability to decrypt the data.
5. Data recovery: The TPM can recover damaged or tampered data via the cloud.

4.3. Threat Model

In order to verify the security of the scheme, we defined a threat model through several games between an adversary, A, and a challenger, C, where the adversary represents a dishonest cloud service provider and the challenger represents a user. In the game, the adversary tries to pass the challenger's verification by using expired data or forging data signatures. The specific process of the game is as follows:

- Setup phase: Challenger C runs the secret key generation algorithm to generate *sk* and system parameter *M*, and then sends *M* to Adversary A.
- Query phase: Adversary A can perform the following three queries through Challenger C:
    (1) Private key query: Adversary A sends the identity to Challenger C, and C obtains the corresponding private key and returns it to A.
    (2) Public key query: Adversary A sends the identity to Challenger C, and C obtains the corresponding public key and returns it to A.
    (3) Hash query: Adversary A sends the parameters to be encrypted to Challenger C, and C calculates the corresponding hash value and returns it to A.
    (4) SigGen query: Adversary A sends the tuple (ID, m) to Challenger C, and C executes the signature algorithm to generate the signature σ corresponding to file block m and returns it to A.
- Challenge phase: In this phase, Challenger C will act as an auditor. C generates a random challenge and sends it to Adversary A, then A calculates the corresponding evidence and returns it to C.
- Verification phase: Challenger C verifies the received evidence. If the evidence generated by A can pass the verification of C with a non-negligible probability, this means that A wins the game.

5. Detailed Implementation of the Scheme

5.1. The Hierarchical Authentication Data Structure

The hierarchical Merkle hash tree consists of a Merkle hash tree with file information as leaf nodes and multiple hash subtrees with data block information as leaf nodes. Each subtree can represent a file of the DO; its structure is shown in Figure 2. In the construction process of the HMHT, the aggregate hash is first calculated for all copies of every block in all files. We construct the leaf nodes of all subtrees with these aggregated hashes, then the hash concatenation operation is performed upward to obtain the subtrees corresponding to all files. The root nodes of the subtrees (such as SR_1, SR_2, SR_3, and SR_4) are the authoritative nodes of the HMHT. Finally, the root nodes of all the subtrees are used as the leaf nodes of the HMHT to perform the hash concatenation operation in turn, to obtain the root node (R) of the entire HMHT. Suppose that we use two files $F_1 = \{b_1, b_2, \ldots, b_8\}$ and $F_2 = \{b_1, b_2, \ldots, b_{32}\}$ to build an HMHT. According to the nature of the full binary tree, the heights of the two subtrees are 4 and 6, respectively, and the balance factor of any branch node in them is 0. When the root nodes of these two subtrees are built up again, an HMHT with a height of 7 can be obtained. At this time, the balance factor of the root node of the HMHT is 6 − 4 = 2. The balance factor is the height difference between the left and right subtrees of the branch node in the tree. Therefore, due to the different number of blocks in each file, the whole HMHT may not be a balanced binary tree, but, hierarchically speaking, each subtree is internally balanced.

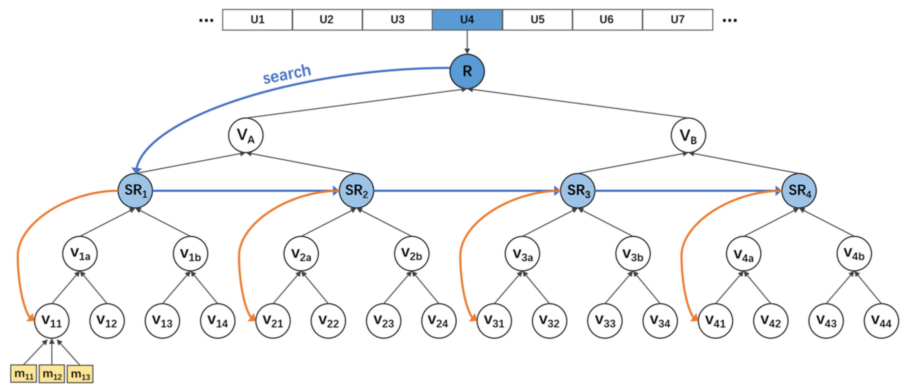

Figure 2. The structure of the hierarchical Merkle hash tree.

We set different attributes for the nodes of different levels in the HMHT to reduce redundant information storage. An explanation of each field is shown in Table 2. It is noteworthy that we maintain a group of leaf pointers in both the root node of the HMHT and the root node of the hash subtree, to achieve a horizontal search of the data and speed up data retrieval efficiency. Compared to many other schemes, such as scheme [19], adding the relative position of its parent node in each node attribute, scheme [24] adds its location coordinate in the tree in each node. There is no doubt that the HMHT can generate a smaller storage overhead and reduce the complexity of the authentication calculations.

Table 2. The node attributes at different levels.

Root Node		Authoritative Node		Leaf Node	
Root hash	$f(v_R)$	Node hash	$f(v_{SRi})$	Node hash	$f(v_{ij})$
		File ID	F_{id}	Block ID	b_{id}
Leaf pointer	LP	Parent pointer	PP	Version	*version*
		Leaf pointer	LP	Parent pointer	PP

5.2. The Specific Realization of the Scheme

5.2.1. Setup Phase

KeyGen(1^k) → (sk, M). We assume that G and G_T are two multiplicative cyclic groups of order p, where p is a large prime number. e is a computable bilinear map and satisfies e: $G \times G \to G_T$. There are three hash encryption functions; H_1: $\{0, 1\}^* \to G$, H_2: $\{0, 1\}^* \to Z_p{}^*$, H_3: $\{0, 1\}^* \to Z_p{}^*$. The DO chooses two random elements, $u \in G$ and $\alpha \in Z_p{}^*$, then calculates g^α, where g represents the generator of the cyclic group G. Finally, we obtain the private key, $sk = \alpha$, and the public key, $pk = g^\alpha$. Then, the DO sets the public verification parameters, $M = \{G, G_T, e, p, g, u, H_1, H_2, H_3, pk\}$.

RepGen(F, m) → F^*. If we suppose that there is an original file, F, the DO first splits the file, F, into n pieces of equal size $F = \{b_1, b_2, \ldots, b_n\}$; if the last data block is not large enough, it needs to be appended with 0 at the end. Then, the DO uses random masking to calculate $m_{ij} = b_i + H_3(b_{id} \| j \| version)$ for the data of each block, $b_{i,}$ and obtains the encrypted data block set, $F^* = \{m_{ij}\}_{1 \le i \le n, 1 \le j \le m}$. m is the total number of replicas of the block. If we want to restore the ciphertext data, $m_{ij,}$ to plaintext data, b_i, we only need to satisfy the equation $b_i = m_{ij} - H_3(b_{id} \| j \| version)$, where b_{id} represents the identification of the data block, and j represents the copy number. In addition, we use the *version* to record the freshness of the blocks.

SigGen(F^*, sk) $\rightarrow$ $\{\sigma_i\}$. The DO uses the random elements u and α to calculate the signature for each replica block, $m_{ij,}$ in the file $F^* = \{m_{ij}\}_{1 \le i \le n, 1 \le j \le m}$ and obtain $\sigma_{ij} = (H_1(\theta_{ij}) \cdot u^{mij})^{\alpha}$, where θ_{ij} = (*filename* | | b_{id} | | j | | *version*), and then calculate the aggregated signature, $\sigma_i = \prod_{j=1}^{m} \sigma_{ij}$, of all the replica blocks, $m_{ij,}$ of b_i. After the signature calculation is completed, the DO initializes the authentication tree structure. Firstly, the DO generates a subtree according to the block information in each file, then identifies all subtree root nodes as authoritative nodes, and then builds the complete authentication tree information, C, upward in turn. Finally, the DO uploads {F^*, M, $\{\sigma_i\}$, C} to the CSP.

Store(F^*, M, $\{\sigma_i\}$, C) $\rightarrow$ *flag*. When the initialization request is received, the CSP first stores the public parameter set M to local storage. Afterward, it verifies the correctness of the signatures $\{\sigma_i\}$ using Formula (1):

$$e(\sigma_i, g) = e(\prod_{j=1}^{m} H_1(\theta_{ij}) \cdot u^{m_{ij}}, pk). \tag{1}$$

The CSP rejects keeping the file and signatures if the Formula (1) validation is unsuccessful, at which point the CSP returns "false"; otherwise, the CSP stores the user data F^*, data signature $\{\sigma_i\}$, and authentication tree C, and returns "true" to the DO. Afterward, the DO uploads the verification parameters {M, C} to the TPA and deletes all local copies of the data to save storage space. At the same time, the TPM is notified to enable the monitoring of cloud data. Figure 3 depicts the complete initialization phase parameter passing process.

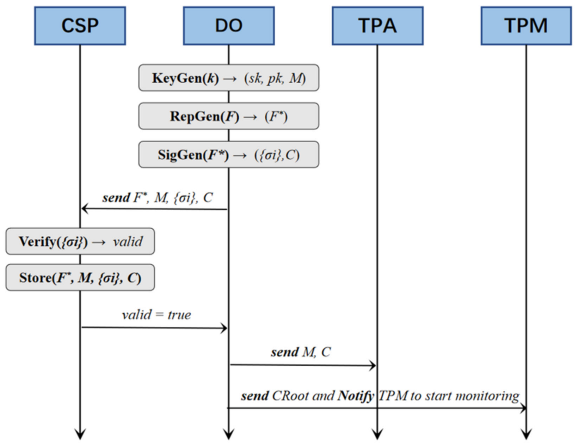

Figure 3. The parameter passing process during the initialization phase.

5.2.2. Verification Phase

The TPA creates a random challenge request, *chal*, during the verification process and transmits it to the CSP. Afterward, the CSP calculates several corresponding evidence parameters, according to the *chal*. Finally, the TPA determines if the data are stored intact in the cloud server by evaluating the validity of the proof. The detailed parameter transfer flow of the verification process is shown in Figure 4.

ChalGen(*params*, c) $\rightarrow$ *chal*. After receiving the audit request from the user, the TPA randomly selects c file block indexes from the file $F = \{b_1, b_2, \ldots, b_n\}$ according to the challenge parameters, to form the challenge set $Q = \{Q_1, Q_2, \ldots, Q_c\}_{c \in [1,n]}$. For each challenge block index $Q_i \in Q$, the TPA generates a random check parameter, $\lambda_i \in Z_p{}^*$, for it, and the challenge information, $chal = \{(Q_i, \lambda_i)\}_{1 \le i \le c,}$ is obtained. Finally, the TPA sends $\{U_{id}, F_{id}, chal\}$ to CSP.

ProofGen(U_{id}, F_{id}, *chal*) $\rightarrow$ *proof*. The CSP obtains $f(v_R)$ of the root node and $f(v_{SR})$ of the authoritative node, according to U_{id} and $F_{id,}$ and searches all the challenge data in the leaf nodes of the corresponding subtree of the HMHT, then calculates the signature aggregation evidence, σ, and data aggregation evidence, μ, of the challenge blocks. Finally, the CSP returns $proof = \{\sigma, \mu, f_{CSP}(v_R), f_{CSP}(v_{SR})\}$ to the TPA.

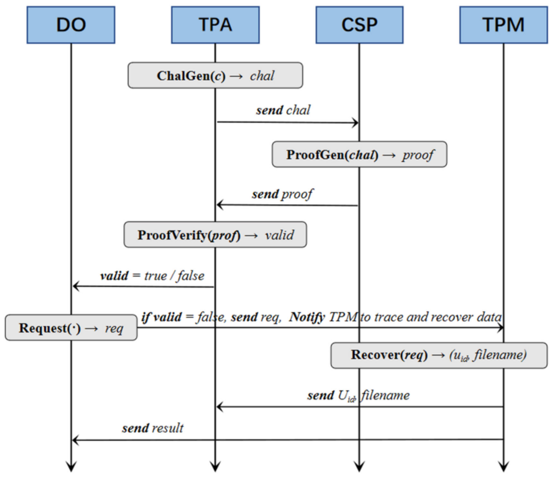

Figure 4. Parameter passing in the verification phase.

$$\sigma = \prod_{i=Q_1}^{Q_c} \sigma_i^{\lambda_i} \ldots \mu = \sum_{i \in Q_1}^{Q_c} \sum_{j=1}^{m} \lambda_i m_{ij}$$

ProofVerify(*proof*, *pk*) $\rightarrow$ *valid*. After the TPA receives the evidence, it first judges whether the root node and authoritative node of the cloud are correct by checking $f_{TPA}(v_R) = f_{CSP}(v_R)$ and $f_{TPA}(v_{SR}) = f_{CSP}(v_{SR})$. If they are correct, the TPA proceeds to verify the aggregate evidence parameters, σ and μ, using the following formula:

$$e(\sigma, g) = e\left(\prod_{i=Q_1}^{Q_c} \prod_{j=1}^{m} H_1(\theta_{ij}) \cdot u^{\mu}, pk\right). \tag{2}$$

If Formula (2) is valid, it returns “true” to the DO. If any part fails to verify, the data that were outsourced are considered to be corrupted or incomplete; the TPA immediately stops the verification and returns “false” to the DO. The DO then transmits a violation review and data recovery request to the TPM at the same time. In Algorithm 1, we give the implementation flow of the verification procedure in detail.

Algorithm 1: Data Auditing

```
Require: $U_{id}$, $filename$, $c$
1: function ChalGen($c$)
2: for $i = 0$; $i < c$, $i$++ do
3: $Q_i$ = random element from $[1, n]$
4: $\lambda_i$ = random element from $Z_p{}^*$
5: $chal_i = \{Q_i, \lambda_i\}$
6: end for
7: return $chal$
8: end function
9: funtion ProofGen($chal$)
10: get block signatures $\{\sigma_i\}$ according to $chal.Q$
11: search blocks $\{m_i\}$, $f(v_R)$, $f(v_{SR})$ according to $chal.Q$ and $HMHT$
12: for $i = 0$; $i < chal.length$; $i$++ do
13: $\sigma = \sigma \cdot \sigma_i^{\lambda_i}$ $\mu = \mu \cdot \lambda_i m_{ij}$
14: end for
15: return $proof = \{\sigma, \mu, f(v_R), f(v_{SRi})\}$
16: end function
```

Algorithm 1: Cont.

```
17: function ProofVerify($proof$, $pk$)
18: if $f(v_R), f(v_{SRi})$ is not correct then
19: return "false"
20: end if
21: compute $left = e(\sigma, g)$
22: compute $right = e(\prod_{i=Q_1}^{Q_c} \prod_{j=1}^{m} H_1(\theta_{ij}) \cdot u^{\mu}, pk)$
23: if $left == right$ then
24: return "true"
25: else
26: return "false"
27: end if
28: end function
```

5.2.3. Dynamic Update Phase

When the DO updates the data, they send the updated parameters to the CSP and TPA. The CSP provides the updated evidence for the TPA to verify, after updating the relevant information, in accordance with the corresponding type. There are three types of data updates: OP_{modify} represents data modification, OP_{insert} represents data insertion, and OP_{delete} represents data deletion.

Data modification: Assuming that the data block, b_i, is modified to b'_i, the DO first performs the encryption calculation $m'_{ij} = b'_i + H_3(b_{id} || j || version)$ on b'_i, then generates a new signature $\sigma'_{ij} = (H_1(\theta'_{ij}) \cdot u^{mij'})^{\alpha}$ for each m'_{ij} and aggregates the signatures of all replica blocks in b'_i to obtain $\sigma'_i = \prod_{j=1}^{m} \sigma'_{ij}$. The DO sends $\{F_{id}, OP_{modify}, b_{id}, \{m'_{ij}\}, \sigma'_i\}$ to the CSP and $\{F_{id}, OP_{modify}, b_{id}, \{f(m'_{ij})\}\}$ to the TPA. Afterward, the CSP replaces all the previous m_{ij} with m'_{ij} according to F_{id} and b_{id}. To guarantee the freshness of the node, the *version* of each modified leaf node is increased by 1, and the CSP recalculates the node value on the authentication path, as shown in Figure 5. Finally, the CSP sends the updated $f'_{CSP}(v_{SR})$ to the TPA, and the TPA checks that $f'_{CSP}(v_{SR}) = f'_{TPA}(v_{SR})$ is correct, based on the locally recomputed HMHT. If it passes the verification test, that means that the modification is successful; otherwise, it is considered that the modification has failed.

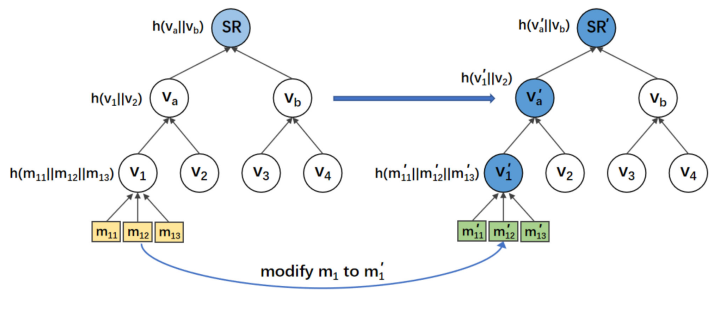

Figure 5. An example of data block m_1 being modified to m'_1.

Data insertion: Data insertion will not affect the entire structure of the HMHT; only the subtree where the data block is located needs to be adjusted. Assuming that b'_i is added following the data block b_i, the DO first performs a masking calculation, $m'_{ij} = b'_i + H_3(b_{id} || j || version)$, on b_i, then generates a new signature, $\sigma'_{ij} = (H_1(\theta'_{ij}) \cdot u^{mij'})^{\alpha}$, for each m'_{ij} and aggregates the signatures of all replica blocks in b'_i to obtain σ'_i. The DO sends $\{F_{id}, OP_{insert}, b_{id}, \{m'_{ij}\}, \sigma'_i\}$ to the CSP and $\{F_{id}, OP_{insert}, b_{id}, \{f(m'_{ij})\}\}$ to the TPA. Subsequently, the CSP inserts all m'_{ij} into their corresponding positions according to F_{id} and b_{id}, and initializes the *version* of the newly inserted leaf node to 1. At this time, if the subtree balance factor is greater than 1, the CSP updates the node value on the authentication path and balances the subtree, as shown in Figure 6. Finally, the CSP sends $f'_{CSP}(v_{SR})$ to the TPA, and the TPA checks that $f'_{CSP}(v_{SR})= f'_{TPA}(v_{SR})$ is correct, according to the adjusted HMHT. If it passes the verification test, this means that the insertion is successful; otherwise, it is considered that the insertion has failed.

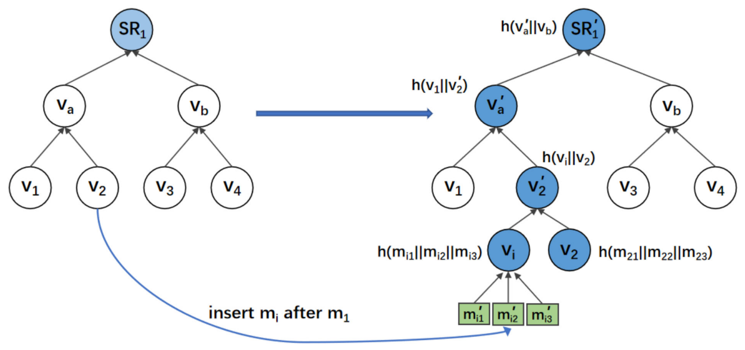

Figure 6. The example of inserting m_i after data block m_1.

Data deletion: When deleting block b_i, the DO directly sends $\{F_{id}, OP_{delete}, b_{id}, nil, nil\}$ to the CSP and $\{F_{id}, OP_{delete}, b_{id}, nil\}$ to the TPA. The CSP deletes the leaf node, replica blocks $\{m_{ij}\}_{1\leq j\leq m,}$ and signature σ_i according to F_{id} and b_{id}, then recalculates the node value on the verification path and balances of the subtree, as shown in Figure 7. Finally, the CSP sends the updated $f'_{CSP}(v_{SR})$ to the TPA; the TPA recalculates the value of the relevant nodes after deleting the leaf node corresponding to b_i in the local storage and checks whether $f'_{CSP}(v_{SR})$ = $f'_{TPA}(v_{SR})$ is correct. If it passes the verification, that means the deletion is successful; otherwise, it is considered that the deletion has failed.

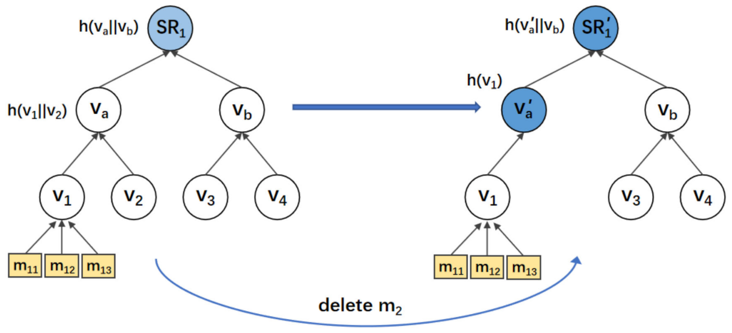

Figure 7. The process of deleting the data block m_2.

5.2.4. Batch Auditing Phase

In practical applications, the TPA may serve multiple Dos; each DO may have a large number of files, and numerous audit tasks delegated by different DOs may be waiting to be processed at the same time. Therefore, we introduce batch auditing to solve the above phenomenon. There are two categories for batch auditing:

When the TPA audits multiple files $f_i \in [f_1, f_s]$ for a single user, the following equation, Formula (3), can be used to confirm that it is correct:

$$\prod_{f_i\in[f_1,f_s]} e(\sigma, g) = \prod_{f_i\in[f_1,f_s]} e(\prod_{i=Q_1}^{Q_c}\prod_{j=1}^{m} H_1(\theta_{ij})\cdot u^{\mu}, pk). \tag{3}$$

When the TPA audits multiple files $f_i \in [f_1, f_s]$ for multiple users, $u_i \in [u_1, u_s]$, the following equation, Formula (4), can be used to confirm that it is correct:

$$\prod_{u_i\in[u_1,u_s]}\prod_{f_i\in[f_1,f_s]} e(\sigma,g) = \prod_{u_i\in[u_1,u_s]}\prod_{f_i\in[f_1,f_s]} e(\prod_{i=Q_1}^{Q_c}\prod_{j=1}^{m} H_1(\theta_{ij})\cdot u^{\mu}, pk). \tag{4}$$

5.2.5. Data Recovery Phase

When the following situations occur, the DO can apply to the TPM for violation review and data recovery:

- After the TPA issued a challenge request to the CSP, the received evidence could not be successfully verified by the TPA.
- The DO finds suspicious or illegal operations via a client of the TPM.
- The TPM detects that the root node value of the HMHT in the cloud is inconsistent with the local value.

After the TPM receives the request, it traces the source of the illegal operation by analyzing the monitoring log, determines whether it is a malicious operation by an unauthorized user or a CSP, and pursues the responsibility of the relevant personnel in accordance with the law. If the data have been tampered with, the TPM will confirm the version of the damaged data with the DO and TPA, compare it with the local operation log, and integrate the data that need to be recovered. Then, the TPM executes the algorithm *DataRecover*(U_{id}, {F_{id}, {b_{id}, {W_{type}, *offset*, *data*}}}), where b_{id} represents the data block identifier, and *offset* refers to the position of the written data within a data block. W_{type} is the type of write operation needed, W_{insert} represents inserting the data, and W_{delete} represents deleting the data. After data recovery is completed, the CSP readjusts the HMHT structure of the user. Finally, the TPM sends {U_{id}, {F_{id}}} to the TPA, then the TPA uses the recovery parameters to create a random challenge and re-initiate the verification request to the CSP. This mechanism is similar to the validation procedure described above and will not be repeated here. Finally, the result is returned to the DO after verification has been passed.

6. Security Analysis

6.1. Correctness

After receiving the storage request from the user, the CSP needs to check the correctness of the data $F^* = \{m_{ij}\}_{1\le i\le n, 1\le j\le m.}$ and the signature $\{\sigma_i\}_{1\le i\le n}$. According to the characteristics of the bilinear map, Formula (1) can be used to verify whether each data block matches the signature. The derivation process of Formula (1) is as follows:

$$\begin{aligned} & e(\sigma_i, g) \\ & = e\left(\prod_{j=1}^{m}\sigma_{ij}, g\right) \\ & = e\left(\prod_{j=1}^{m}\left(H_1(\theta_{ij})\cdot u^{m_{ij}}\right)^{\alpha}, g\right) \\ & = e\left(\prod_{j=1}^{m} H_1(\theta_{ij})\cdot u^{m_{ij}}, pk\right) \end{aligned}$$

After receiving the TPA challenge request, *chal*, the CSP will retrieve the relevant data and signatures of all challenge blocks in the *chal* and generate evidence. If the CSP stores the user's data correctly, the evidence can pass the verification of the TPA. According to the characteristics of bilinear mapping, Formula (2) can be used to verify this evidence. The proof process of Formula (2) is as follows:

$$
\begin{aligned}
& e(\sigma, g) \\
& = e\left(\prod_{i=Q_1}^{Q_c} {\sigma_i}^{\lambda_i}, g\right) \\
& = e\left(\prod_{i=Q_1}^{Q_c} \prod_{j=1}^{m} \left(H_1\left(\theta_{ij}\right) \cdot u^{m_{ij}}\right)^{\alpha\lambda_i}, g\right) \\
& = e\left(\prod_{i=Q_1}^{Q_c} \prod_{j=1}^{m} \left(H_1\left(\theta_{ij}\right) \cdot u^{m_{ij}}\right)^{\lambda_i}, pk\right) \\
& = e\left(\prod_{i=Q_1}^{Q_c} \prod_{j=1}^{m} H_1\left(\theta_{ij}\right)^{\lambda_i} \cdot u^{\sum\limits_{i=1}^{c} \sum\limits_{j=1}^{m} \lambda_i m_{ij}}, pk\right) \\
& = e\left(\prod_{i=Q_1}^{Q_c} \prod_{j=1}^{m} H_1\left(\theta_{ij}\right)^{\lambda_i} \cdot u^{\mu}, pk\right)
\end{aligned}
$$

6.2. Resisting Forgery Attacks

Assuming that Adversary A can forge a valid signature with a non-negligible probability, ε, within a certain period of time, then we can assert that there is an algorithm that can solve the CDH problem with non-negligible probability.

Proof: In the signature forgery game, Challenger C first generates the private key, *sk*, and system parameters, $M = \{G, G_T, e, p, g, u, H_1, H_2, H_3, pk\}$, and sends them to Adversary A. Afterward, A selects a user ID to pose multiple queries of corresponding types to C. The interaction process between A and C is as follows:

Private key query: A obtains the private key, *sk**, by sending *I** to C; if C retrieves ID^* in the internal table $L_1 = \{(ID, sk)\}$, C returns *sk** to A, otherwise C stops the game.

Public key query: A obtains the public key, *pk**, by sending *ID** to C. If C retrieves ID^*, C calculates $pk = g^{sk^*}$ and sends it to A.

Hash query: When C receives the θ^* sent by A, C calculates the corresponding hash value and returns it to A, where $\theta^* = (filename || b_{id} || j || version)$.

SigGen query: A sends a triple (ID^*, m^*, θ^*) to C, and C executes the signature algorithm to generate the data signature σ^* corresponding to the file block.

Forge: Finally, A forges the signature corresponding to the data block, *m**. If the following conditions are met, this proves that A wins the game.

(1) E_1: Challenger C did not stop the game during the private key retrieval phase.
(2) E_2: Adversary A successfully outputs the signature σ^* corresponding to *m**.
(3) E_3: Under the condition that E_2 has occurred, the forged signature, σ^*, can pass the verification stage.

Assuming that the query times of the private key, public key, hash, and signature are Q_{sk}, Q_{pk}, Q_{hv}, and Q_{sig}, respectively, we obtain:

$$
\begin{aligned}
& \Pr[E_1] \geq (1 - \tfrac{Q_{sk}}{p})^{Q_{sk}} \\
& \Pr[E_2 | E_1] \geq \varepsilon \\
& \Pr[E_3 \Big| E_1 \wedge E_2] \geq \tfrac{Q_{sig}}{p}
\end{aligned}
$$

From the above formula, the probability of C solving the CDH problem can be obtained as:

$$
\begin{aligned}
& \Pr[E_1 \wedge E_2 \wedge E_3] \\
& = \Pr[E_1] \cdot \Pr[E_2 | E_1] \cdot \Pr[E_3 | E_1 \wedge E_2]. \\
& \geq \tfrac{Q_{sig}}{p} (1 - \tfrac{Q_{sk}}{p})^{Q_{sk}} \varepsilon
\end{aligned}
$$

Since p is large enough, we can obtain $\Pr[E_1 \wedge E_2 \wedge E_3] \ll \varepsilon$. Therefore, Challenger C cannot solve the CDH problem with non-negligible probability within a certain period of time. In other words, the adversary cannot forge a valid signature with non-negligible probability within a set timeframe.

6.3. Resisting Replay Attacks

The CSP may employ a replay attack to deceive the TPA, in order to win the challenge. Now, we assume that the block m corresponding to element k is corrupted; when the CSP receives the challenge request $chal = \{(i,\ \lambda_i)\}_{Q1 \le i \le Qc}$, it uses the expired information to regenerate the evidence. At this time, the evidence of σ^- and μ^- is calculated as follows:

$$\sigma^- = \prod_{i \in Q \cap i \neq k} \sigma_i^{\lambda_i} \cdot \sigma_{k^-}^{\lambda_k{}^-}$$

$$\mu^- = \sum_{i \in Q \cap i \neq k} \sum_{j=1}^{m} \lambda_i m_{ij} + \sum_{j=1}^{m} \lambda_k^- m_{kj}^-.$$

The TPA checks the correctness of the evidence after receiving it. If the evidence successfully passes the challenge, it means that the attack is successful; if it fails to pass, the attack has failed. To appear more persuasive, we perform calculations on both sides of the verification formula, to check the invalidity of the replayed evidence.

$$\begin{aligned}
& e(\sigma^-, g) \\
&= e\left(\prod_{i \in Q \bigcap i \neq k} \sigma_i^{\lambda_i} \cdot \sigma_{k^-}^{\lambda_k{}^-}, g\right) \\
&= e\left(\prod_{i \in Q \bigcap i \neq k} \prod_{j=1}^{m} \left(H_1(\theta_{ij}) \cdot u^{m_{ij}}\right)^{\alpha\lambda_i} \cdot \prod_{j=1}^{m} \left(H_1\left(\theta_{kj}^-\right) \cdot u^{m_{kj}^-}\right)^{\alpha\lambda_k^-}, g\right) \\
&= e\left(\prod_{i \in Q \bigcap i \neq k} \prod_{j=1}^{m} \left(H_1(\theta_{ij}) \cdot u^{m_{ij}}\right)^{\lambda_i} \cdot \prod_{j=1}^{m} \left(H_1\left(\theta_{kj}^-\right) \cdot u^{m_{kj}^-}\right)^{\lambda_k^-}, pk\right) \\
&= e\left(\prod_{i \in Q \bigcap i \neq k} \prod_{j=1}^{m} H_1(\theta_{ij})^{\lambda_i} \cdot \prod_{j=1}^{m} H_1\left(\theta_{kj}^-\right)^{\lambda_k^-}, pk\right) \cdot e\left(u^{\sum_{i \in Q \bigcap i \neq k} \sum_{j=1}^{m} \lambda_i m_{ij} + \sum_{j=1}^{m} \lambda_k^- m_{kj}^-}, pk\right)
\end{aligned}$$

The proof procedure of the right end of the formula is as follows:

$$\begin{aligned}
& e\left(\prod_{i=Q_1}^{Q_c} \prod_{j=1}^{m} H_1(\theta_{ij})^{\lambda_i} \cdot u^{\mu^-}, pk\right) \\
&= e\left(\prod_{i \in Q \cap i \neq k} \prod_{j=1}^{m} H_1(\theta_{ij})^{\lambda_i} \cdot \prod_{j=1}^{m} H_1\left(\theta_{kj}\right)^{\lambda_k}, pk\right) \cdot e\left(u^{\sum_{i \in Q \cap i \neq k} \sum_{j=1}^{m} \lambda_i m_{ij} + \sum_{j=1}^{m} \lambda_k^- m_{kj}^-}, pk\right)
\end{aligned}$$

Comparing the above two derivation results, we find that the equation is valid only when $\prod_{j=1}^{m} H_1\left(\theta_{kj}^-\right)^{\lambda_k^-}$ and $\prod_{j=1}^{m} H_1\left(\theta_{kj}\right)^{\lambda_k}$ are equal. However, the signatures and content of the different data blocks are different, and the challenge parameter, $\lambda_{i,}$ is random each time. The probability of their collision is much less than $1/p^2$, but, since p is large enough, the CSP cannot use expired evidence to successfully complete the challenge process of the TPA.

6.4. Monitoring Security

In practical applications, dishonest behavior by CSPs often occurs. For example, CSPs delete data to reduce storage pressure, or CSPs gain profits by illegally analyzing or selling user data, etc. Since TPAs and TPMs will only rarely interact with other protocol parties, and the types of services they provide are relatively single, it is difficult for them to perform illegal actions. The introduction of TPM can effectively provide security protection for user data, thereby improving the level of the security protection capabilities of enterprises. First, TPM can promote mutual restraint between the parties to an agreement and reduce the probability of illegal behavior. Second, TPM can record the behaviors of users and cloud service providers in real time, so as to deal with violations in a timely manner and trace the source of accountability. Finally, TPM can use logs to restore user data, reduce the number of copies in the cloud, and reduce the storage pressure and computing burden of cloud service providers.

6.5. Data Privacy

In cloud storage, data privacy is a crucial security guarantee. Data privacy protection requires that the user's data cannot be compromised throughout its lifecycle and cannot be accessed by any other untrusted entity. To mask the characteristics of the original data, we encrypt the file contents using a hash function and use version numbers to improve the freshness of the data. Using a hash function to map security parameters can protect them; it also prevents attackers from deciphering the data and helps to resist data-block forgery attacks.

7. Efficiency Analysis and Experiment Comparison

By comparing the suggested solution with three other similar schemes, we are able to confirm its performance. Scheme [21] is an identity-based multi-copy auditing scheme; Scheme [24] is a certificate-less multi-copy integrity auditing protocol; Scheme [35] is an auditing scheme with user revocation. We will compare and verify the effectiveness of our scheme from several angles.

7.1. Theoretical Analysis

We use the following symbols to represent the time overhead of some operations in the scheme, where *Pair* represents performing a bilinear pairing operation, and *Exp* and *Mul* represent an exponential operation and a multiplicative operation in group G, respectively. The time consumed by other operations, such as hash functions and additions, is small and can be ignored here. We assume that the file number of the user is t; each file is divided into n blocks, m indicates the number of replicas, s indicates the number of sectors, and c represents the length of the challenge set.

7.1.1. Computational Overhead

Creating signatures, generating evidence, and verifying evidence are the three most resource-consuming stages in the auditing process, so we only compare the computational costs of these three stages. As shown in Table 3, our scheme only needs to perform a multiplication operation for each signature generated, while the other three comparison schemes almost need to execute it twice. Moreover, as the number of sectors increases, the number of exponential operations in the scheme [21] will also increase, therefore, the computational overhead of our solution is minimal. Before uploading the data to the cloud, our scheme already computes local aggregated signatures for each block during the verification phase, which makes our scheme independent of the number of copies when calculating the evidence. In addition, only two bilinear pairing operations are required during the verification time. Therefore, compared to the other three methods, our solution is able to consume fewer computational resources.

Table 3. Computational overhead comparison.

	Scheme [21]	**Scheme [24]**	**Scheme [35]**	**Our Scheme**
SigGen	$(2mn+s+1)Exp+2mnMul$	$2mnExp+(2mn+m-1)Mul$	$2mnExp+2mnMul$	$2mnExp+mnMul$
ProofGen	$mcExp+mcMul$	$cExp+m(c-1)Mul$	$cExp+mcMul$	$cExp+(c-1)Mul$
ProofVerify	$3Pair+(mc+s+3)Exp+(mc+s)Mul$	$3Pair+(mc+s+1)Exp+(mc+s)Mul$	$3Pair+2cExp+2mcMul$	$2Pair+(mc+1)Exp+mcMul$

7.1.2. Communication Cost

In the PDP-based audit mechanism, we mainly discuss the communication overhead complexity of the verification stage and the dynamic update stage. As shown in Table 4, in our scheme, the TPA sends the CSP a challenge request, *chal*, which causes $O(c)$ of communication overhead. CSP calculates the evidence, $proof = \{\sigma, \mu, f(v_R), f(v_{SRi})\}$, according to the *chal*. These parameters are constants that are independent of the challenge set length, and the transmission of evidence will cause $O(1)$ communication cost. In summary, the total overhead incurred in the communication process during the validation stage is $O(c)$. However, the communication overhead of schemes [21,24] is not only related to the challenge length but is also influenced by the count of sectors or storage structure. Assuming that the number of update blocks is w, our scheme only generates $O(w)$ of communication overheads in the update process, while scheme [35] is also affected by the number of replicas when generating dynamic update requests because it does not calculate the aggregate signature. All in all, our scheme can generate fewer communication overheads during the auditing process.

Table 4. Communication overhead complexity comparison.

	Verification Stage	**Update Stage**
Scheme [21]	$O(c + s)$	-
Scheme [24]	$O(clogn)$	$O(wlogn)$
Scheme [35]	$O(c)$	$O(mw)$
Our Scheme	$O(c)$	$O(w)$

7.2. Experimental Comparison

All experiments in this paper were implemented using the VMware Workstation. The configuration of the virtual machine comprises 4 G of memory and a 20 G hard disk. The virtual machine's OS type is Ubuntu and the version is 18.04.06. The virtual environment relies on a local computer configured with an Intel Core i7-10870H@2.20GHz CPU and 16G RAM. We programmed the experimental code using the Go language and implemented the BLS signature algorithm using the PBC Go Wrapper, a wrapper library of PBC functions [36]. The experiment adopts the elliptic curve group of type A, then sets the basic field size to 512 bits, and the order size of the group to 160 bits. Each series of tests was carried out 20 times to obtain an average value that would accurately reflect the correctness of the experimental data.

We first performed test experiments on a single file to evaluate the efficiency of signature generation. In this test, we fixed the number of file copies to 5 and the number of sectors to 100, then generated signatures for the different blocks of files from 0 to 1000. The test results are shown in Figure 8. In all four scenarios, the time consumed for signature generation increased linearly with the number of file blocks. For 1000 file blocks, our solution consumed only 10.3 s, while the other three solutions had a computation time of 12 s or more. Since our solution is not influenced by the sectors, the computation cost here is, thus, lower than in the other three schemes.

Then, we investigated the influence of the challenge length, c, on the validation stage. Similarly, we fixed the number of copies to 5 and the file blocks to 5000. As the challenge length increased from 0 to 1000, we evaluated the time consumed for evidence generation and evidence validation. The results of this test can be found in Figure 9. We can see that the audit time of the other three solutions is more than 10 s, while our solution only takes almost 6.5 s. Since our solution has already calculated the local aggregation signature per block before the file is uploaded, it only needs to execute two linear pairing operations when verifying the evidence. As a result, our solution has higher audit efficiency.

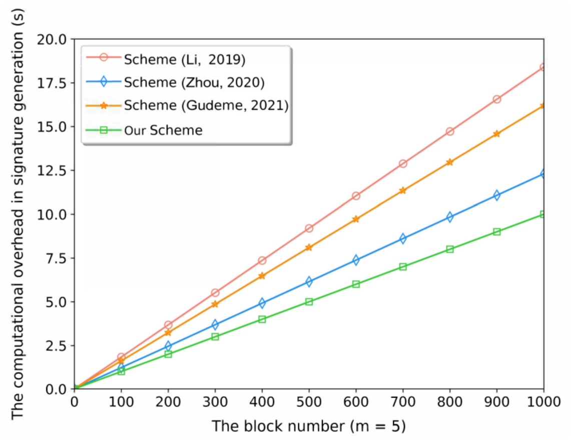

Figure 8. The computational overhead for signature generation [21,24,35].

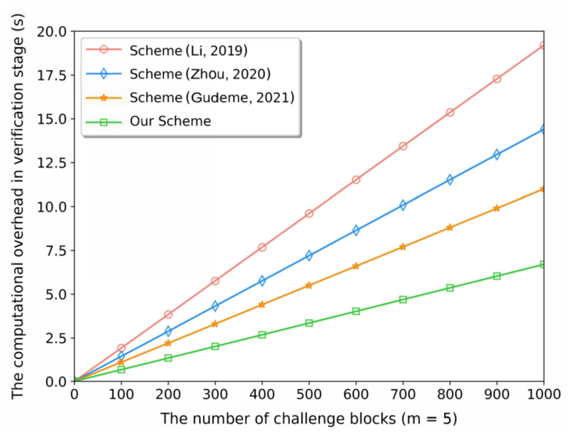

Figure 9. The computational overhead at the verification stage [21,24,35].

We compared the communication overheads incurred throughout the data update procedure after assessing the computational overheads. Since scheme [21] does not support dynamic manipulation, we can only compare it with scheme [24] and scheme [35]. We set the block number to 5000, then adjusted the number of updated blocks from 0 to 100. The outcomes of the experiment are displayed in Figure 10. During the update process, our solution only transferred 44 KB of parameter information. Due to the use of hierarchical technology to make the authentication path shorter, we can see that our approach has incurred a smaller communication cost. This advantage is more remarkable in the case of multiple users.

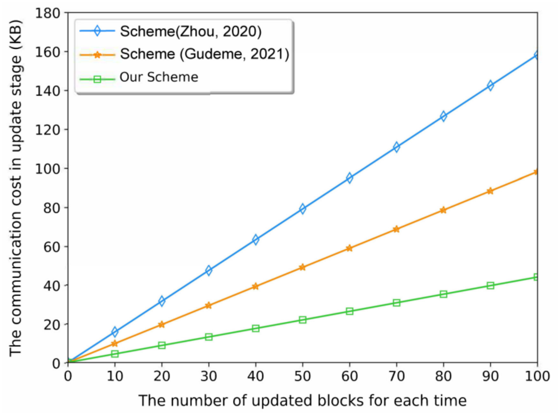

Figure 10. The communication cost at the update stage [24,35].

Finally, we evaluated the efficiency of batch auditing. For each user, we set the number of files to 5000, and each file was divided into 500 pieces. In addition, we fixed the length of the challenge at 200. Then we adjusted the number of challenge files from 0 to 100 and adjusted the number of challenge users from 0 to 10. The outcomes of the experiment are displayed in Figure 11. As the number of files and users increased, the retrieval frequency also increased. For example, when the TPA sent a challenge to the CSP to validate 100 files among 10 users, the total number of random blocks was $10 \times 100 \times 200$. Due to the hierarchical principle of the HMHT, our scheme only needs to perform 3.54×105 retrievals on average, which can greatly reduce the number of invalid retrievals.

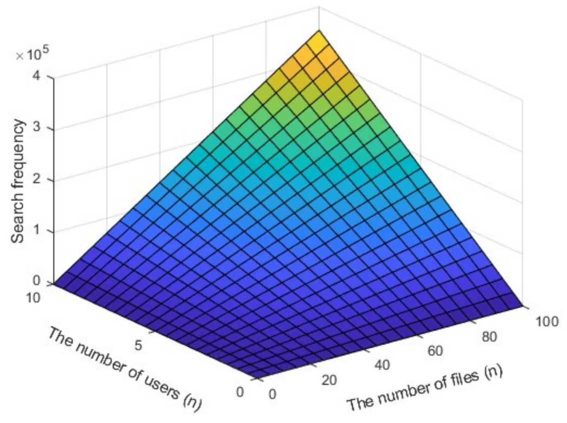

Figure 11. The search frequency used in batch auditing.

8. Conclusions

This paper provides a data integrity auditing scheme, based on a hierarchical Merkle hash tree, which can effectively reduce the scale of the authentication tree through local signature aggregation technology and hierarchical principles, thus avoiding invalid retrieval to a large extent. To improve auditing efficiency, we have also used authoritative nodes to make the authentication path shorter. In addition, we have improved the transparency and reliability of the cloud service provider by introducing a monitoring mechanism, which also realizes the recovery of incomplete data and the traceability of illegal actions by analyzing operation logs. This effectively reduces the resource consumption of the cloud service providers; it also improves the fairness of the agreement. The theoretical evaluation and practical experimental comparison have proven the high levels of safety and efficiency of our solution. In the next step, we will use lightweight encryption technology to research data security in the shared environment on this basis. In addition, we will design a new type of updateable signature algorithm, to avoid problems such as large source data transmission and revoked user attacks, after members quit the sharing group to achieve more effective data security protection.

Author Contributions: Conceptualization, methodology, and writing—original draft preparation, Z.L.; software and validation, Z.L. and S.W.; resources, Z.L. and J.W.; writing—review and editing, S.W.; visualization and supervision, S.D. and L.R.; project administration, J.W.; funding acquisition, Z.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Natural Science Foundation of Hebei Province, China under Grant No. F2019201427 and Fund for Integration of Cloud Computing and Big Data, Innovation of Science and Education of China under Grant No. 2017A20004.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Srivastava, S.; Saxena, S.; Buyya, R.; Kumar, M.; Shankar, A.; Bhushan, B. CGP: Cluster-based gossip protocol for dynamic resource environment in cloud. *Simul. Model. Pract. Theory* **2021**, *108*, 102275. [CrossRef]
2. Wang, H.; He, D.; Fu, A.; Li, Q.; Wang, Q. Provable data possession with outsourced data transfer. *IEEE Trans. Serv. Comput.* **2019**, *14*, 1929–1939. [CrossRef]
3. Chuka-Maduji, N.; Anu, V. Cloud Computing Security Challenges and Related Defensive Measures: A Survey and Taxonomy. *SN Comput. Sci.* **2021**, *2*, 331. [CrossRef]
4. Xu, Y.; Sun, S.; Cui, J.; Zhong, H. Intrusion-resilient public cloud auditing scheme with authenticator update. *Inf. Sci.* **2020**, *512*, 616–628. [CrossRef]
5. Hu, C.; Xu, Y.; Liu, P.; Yu, J.; Guo, S.; Zhao, M. Enabling cloud storage auditing with key-exposure resilience under continual key-leakage. *Inf. Sci.* **2020**, *520*, 15–30. [CrossRef]
6. Zhou, L.; Fu, A.; Yu, S.; Su, M.; Kuang, B. Data integrity verification of the outsourced big data in the cloud environment: A survey. *J. Netw. Comput. Appl.* **2018**, *122*, 1–15. [CrossRef]
7. Etemad, M.; Küpçü, A. Generic dynamic data outsourcing framework for integrity verification. *ACM Comput. Surv. (CSUR)* **2020**, *53*, 8. [CrossRef]
8. Tian, J.; Wang, H. A provably secure and public auditing protocol based on the bell triangle for cloud data. *Comput. Netw.* **2021**, *195*, 108223. [CrossRef]
9. Parast, F.K.; Sindhav, C.; Nikam, S.; Yekta, H.I.; Kent, K.B.; Hakak, S. Cloud computing security: A survey of service-based models. *Comput. Secur.* **2022**, *114*, 102580. [CrossRef]
10. Jin, H.; Jiang, H.; Zhou, K. Dynamic and public auditing with fair arbitration for cloud data. *IEEE Trans. Cloud Comput.* **2016**, *6*, 680–693. [CrossRef]
11. Juels, A.; Kaliski Jr, B.S. PORs: Proofs of retrievability for large files. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 31 October–2 November 2007; pp. 584–597.
12. Piao, C.; Hao, Y.; Yan, J.; Jiang, X. Privacy protection in government data sharing: An improved LDP-based approach. *Serv. Oriented Comput. Appl.* **2021**, *15*, 309–322. [CrossRef]
13. Shacham, H.; Waters, B. Compact proofs of retrievability. *J. Cryptol.* **2013**, *26*, 442–483. [CrossRef]
14. Ateniese, G.; Burns, R.; Curtmola, R.; Herring, J.; Kissner, L.; Peterson, Z.; Song, D. Provable data possession at untrusted stores. In Proceedings of the 14th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 31 October–2 November 2007; pp. 598–609.
15. Wang, Q.; Wang, C.; Ren, K.; Lou, W.; Li, J. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Trans. Parallel Distrib. Syst.* **2010**, *22*, 847–859. [CrossRef]
16. Erway, C.C.; Küpçü, A.; Papamanthou, C.; Tamassia, R. Dynamic provable data possession. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **2015**, *17*, 15. [CrossRef]
17. Shen, J.; Shen, J.; Chen, X.; Huang, X.; Susilo, W. An efficient public auditing protocol with novel dynamic structure for cloud data. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 2402–2415. [CrossRef]
18. Su, D.; Liu, Z. New type of Merkle hash tree for integrity audit scheme in cloud storage. *Comput. Eng. Appl.* **2018**, *54*, 70–76.
19. Hariharasitaraman, S.; Balakannan, S. A dynamic data security mechanism based on position aware Merkle tree for health rehabilitation services over cloud. *J. Ambient. Intell. Humaniz. Comput.* **2019**, *4*, 1–15. [CrossRef]
20. Shen, W.; Qin, J.; Yu, J.; Hao, R.; Hu, J. Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 331–346. [CrossRef]
21. Li, J.; Yan, H.; Zhang, Y. Efficient identity-based provable multi-copy data possession in multi-cloud storage. *IEEE Trans. Cloud Computing* **2019**, *10*, 356–365. [CrossRef]
22. Liu, Z.; Liu, Y.; Yang, X.; Li, X. Integrity Auditing for Multi-Copy in Cloud Storage Based on Red-Black Tree. *IEEE Access* **2021**, *9*, 75117–75131. [CrossRef]
23. Garg, N.; Bawa, S.; Kumar, N. An efficient data integrity auditing protocol for cloud computing. *Future Gener. Comput. Syst.* **2020**, *109*, 306–316. [CrossRef]
24. Zhou, L.; Fu, A.; Yang, G.; Wang, H.; Zhang, Y. Efficient certificateless multi-copy integrity auditing scheme supporting data dynamics. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 1118–1132. [CrossRef]
25. Thangavel, M.; Varalakshmi, P. Enabling ternary hash tree based integrity verification for secure cloud data storage. *IEEE Trans. Knowl. Data Eng.* **2019**, *32*, 2351–2362. [CrossRef]
26. Zhou, L.; Fu, A.; Feng, J.; Zhou, C. An efficient and secure data integrity auditing scheme with traceability for cloud-based EMR. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–6.
27. Xu, G.; Han, S.; Bai, Y.; Feng, X.; Gan, Y. Data tag replacement algorithm for data integrity verification in cloud storage. *Comput. Secur.* **2021**, *103*, 102205. [CrossRef]
28. Luo, W.; Ma, W.; Gao, J. MHB* T based dynamic data integrity auditing in cloud storage. *Clust. Comput.* **2021**, *24*, 2115–2132. [CrossRef]
29. Gudeme, J.R.; Pasupuleti, S.K.; Kandukuri, R. Attribute-based public integrity auditing for shared data with efficient user revocation in cloud storage. *J. Ambient Intell. Humaniz. Comput.* **2021**, *12*, 2019–2032. [CrossRef]

30. Li, Y.; Yu, Y.; Min, G.; Susilo, W.; Ni, J.; Choo, K.-K.R. Fuzzy identity-based data integrity auditing for reliable cloud storage systems. *IEEE Trans. Dependable Secur. Comput.* **2017**, *16*, 72–83. [CrossRef]
31. Shen, W.; Qin, J.; Yu, J.; Hao, R.; Hu, J.; Ma, J. Data integrity auditing without private key storage for secure cloud storage. *IEEE Trans. Cloud Comput.* **2019**, *9*, 1408–1421. [CrossRef]
32. Zhang, Y.; Xu, C.; Lin, X.; Shen, X.S. Blockchain-based public integrity verification for cloud storage against procrastinating auditors. *IEEE Trans. Cloud Computing* **2019**, *9*, 923–937. [CrossRef]
33. Huang, P.; Fan, K.; Yang, H.; Zhang, K.; Li, H.; Yang, Y. A collaborative auditing blockchain for trustworthy data integrity in cloud storage system. *IEEE Access* **2020**, *8*, 94780–94794. [CrossRef]
34. Yang, Y.; Chen, Y.; Chen, F. A compressive integrity auditing protocol for secure cloud storage. *IEEE/ACM Trans. Netw.* **2021**, *29*, 1197–1209. [CrossRef]
35. Gudeme, J.R.; Pasupuleti, S.K.; Kandukuri, R. Certificateless multi-replica public integrity auditing scheme for dynamic shared data in cloud storage. *Comput. Secur.* **2021**, *103*, 102176. [CrossRef]
36. Lynn, B. The Pairing-Based Cryptographic Library. Available online: https://crypto.stanford.edu/pbc (accessed on 1 December 2016).