Journal of Computer Science and Electrical Engineering
ISSN: 2663-1946
DOI: 10.61784/jcsee3009

RESEARCH ON FLEXIBLE AND VERIFIABLE SECURITY PROTOCOLS IN CLOUD COMPUTING ENVIRONMENTS

XiaoYin Yi[1,2*], Qian Huang[3]
[1]*Chongqing Key Laboratory of Public Big Data Security Technology, Qijiang 401420, Chongqing, China.*
[2]*Chongqing College of Mobile Communication, Qijiang 401420, Chongqing, China.*
[3]*Chongqing Polytechnic University of Electronic Technology, Shapingba401331, Chongqing,China.*
Corresponding Author:XiaoYin Yi, Email:17323415956@189.cn

Abstract: With the rapid development of cloud computing technology, an increasing number of enterprises and individuals are migrating their data and applications to cloud environments. Although cloud computing offers convenient services and flexible resource management, security issues remain a primary concern for users. This paper explores flexible and verifiable security protocols in cloud environments with the aim of enhancing data security and user trust. By analyzing and improving existing security protocols, a novel security protocol framework is proposed and experimentally validated. The results indicate that this framework demonstrates significant advantages in both flexibility and security.
Keywords: Cloud Storage; Data Integrity; Hierarchical Merkle Tree; Multi-Grained

1. INTRODUCTION

The popularization of cloud computing has brought about a revolutionary change in the way data is stored and processed. However, the accompanying security issues have also become increasingly prominent. Users face risks such as data leakage and unauthorized access when storing data in the cloud. Therefore, researching flexible and verifiable security protocols is crucial for ensuring the security of cloud environments.
Structurally, a cloud storage system [1] can be divided into four layers: storage layer, basic management layer, application interface layer, and access layer. Generally speaking, the architecture of a cloud storage system may vary depending on the application environment and research objectives. Figure 1 illustrates the general architecture of existing cloud storage systems.

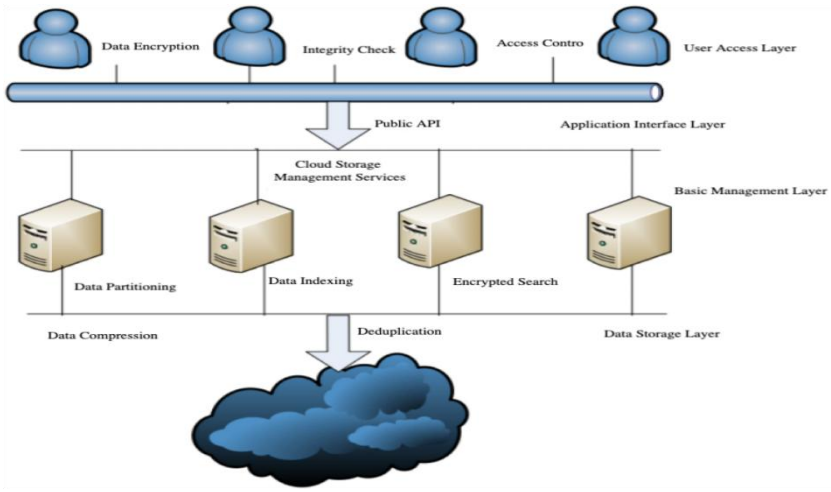

Figure 1 General Architecture of Cloud Storage Systems

The issue with cloud storage services is finding an efficient way for users to verify the authenticity and integrity of data on remote cloud servers. When the service provider is untrustworthy, whether due to dishonesty or system failures, this security threat can be mitigated by integrating data fault tolerance with data integrity and availability verification. When integrity verification schemes involve a third-party auditor, this third party, as an entity within the cloud storage architecture, may also be untrustworthy. The third party could collude with the cloud service provider to help conceal data that has been lost or altered by the provider [2]. Literature confirms users' concerns by indicating that after multiple integrity verifications, as long as a sufficient number of linear equations involving identical blocks are collected, the content of these blocks can be easily obtained by solving the equations. When the third-party auditor is untrustworthy, further integration of data encryption or privacy protection schemes is necessary to address this issue.
The greatest challenge in cloud storage performance is the ability to move data between users and remote cloud storage providers [3], primarily through TCP, which is a connection-oriented protocol based on packet acknowledgment and controls data flow from peer endpoints. Multi-tenancy is a key feature of cloud storage architecture. The characteristics

of cloud storage include: manageability, access methods, performance, multi-tenancy, scalability, availability, control, efficiency, and cost. For details, see Table 1 below.

Table 1 Characteristics of Cloud Storage

Characteristic	Description
Manageability	The ability to manage the system with minimal resources
Access Methods	The protocols used for accessing public cloud storage
Performance	Performance measured based on bandwidth and latency
Multi-Tenancy	Support for multiple users (or tenants)
Scalability	The ability to handle higher demands or scale appropriately
Storage Efficiency	Measurement of how efficiently raw storage is utilized
Cost	Measurement of storage costs (typically per GB)
Data Availability	Measurement of system uptime
Control	The ability to control the system—especially configuring for cost, performance, or other features

Currently, in cloud storage, file chunking is handled with fixed chunk sizes corresponding to BLS signatures, where 20 bytes are used as the basic data unit for authentication. This results in a large Merkle hash tree structure with significant storage costs and slow search operations. Moreover, this approach is only suitable for verifying small data chunks, leading to excessive communication volume and inefficiency during verification [4]. Additionally, the granularity of verification and update operations is limited to 20-byte data blocks, which does not integrate verification environments and actual dynamic operation needs, thus limiting user control. This paper addresses these shortcomings through detailed research and analysis, proposing a reliable and efficient verifiable scheme that further improves and refines the existing verification methods.

The main contributions of this paper are as follows:

(1) We have researched and compared existing cloud storage data integrity verification protocols. Based on the shortcomings of these schemes, we propose a dynamic data security solution that can be flexibly verified in an untrusted environment.

(2) The scheme is applied in an environment where all three parties (users, third-party auditors, and cloud storage servers) are untrustworthy. It includes an operation for the cloud storage server to verify root nodes, enabling the server to more promptly assess the trustworthiness of users and accept user data. Additionally, we have constructed a new hierarchical Merkle hash tree.

(3) The scheme supports dynamic data operations, public verification, and privacy protection. It takes into account and integrates the varying needs of different users, enhancing overall efficiency. Verification operations support chunks and sub-chunks, and dynamic operations can also be supported at these levels. This approach better meets actual user needs and increases the flexibility of verification by elevating the granularity from a singular to an optionally selectable level.

2. RELATED WORK

2.1 File Partitioning

First, the file F is divided into I chunks, represented as $F=(m_1,m_2,...,m_I)$. Then, each chunk m_i is further divided into J sub-chunks, represented as $F=(m_{1,1},m_{1,2},...,m_{I,J})$. Finally, each sub-chunk $m_{i,j}$ is further divided into K basic blocks, at which point the file is represented as $F=(m_{1,1,1},m_{1,1,2},m_{1,1,K},m_{1,2,1},...,m_{I,J,K})$. From these three definitions of the file F, the file can be viewed as a 3-dimensional file structure, as shown in Figure 2. In simple terms, this means decomposing the file from coarse granularity to various sizes of finer granularity [5]. In this paper, 160-bit chunks are used as basic blocks in our scheme, with data sub-chunk sizes of 8KB and data chunk sizes of 512KB. The sub-chunks serve as the J-layer leaf nodes, and data chunks serve as the I-layer leaf nodes to construct a hierarchical Merkle hash tree.

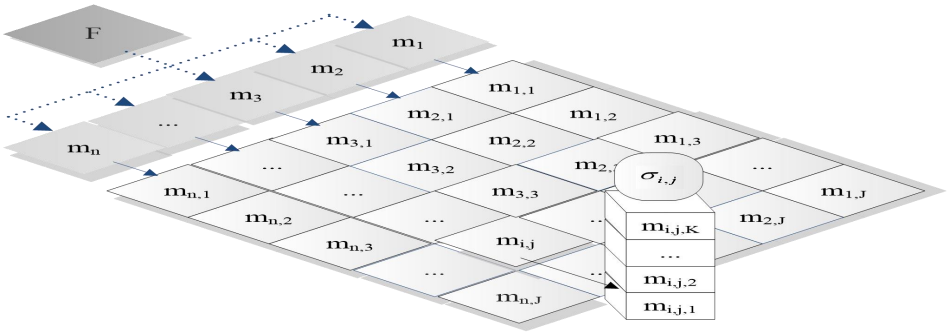

Figure 2 File Chunking Diagram

2.2 Hierarchical Merkle Hash Tree

A hierarchical Merkle hash tree is a logical structuring of the Merkle Hash Tree, aimed at providing a more intuitive representation of the integrity verification process based on two levels of granularity, as well as supporting dynamic operations on blocks. The structure of the hierarchical Merkle hash tree is shown in Figure 3, with the entire tree divided into two levels: Level *I* and Level *J*. Below the dashed line is Level *J*, where a leaf node at Level *J* represents the hash $h(H(m_{i,j}))$ of a sub-chunk label $H(m_{i,j})$.Above the dashed line is Level *I*, where a leaf node at Level *I* represents the hash $h(H^{'}(m_i))$ of a data chunk label $H^{'}(m_i)$. The hash value of the root node of the *J*-level subtree is the same as the hash value of the leaf nodes in the *I*-level tree. Additionally, the relationship between the data chunk label $H^{'}(m_i)$ and the block label $H(m_{i,j})$ is defined as $H^{'}(m_i)=\prod_{1\le i\le n,1\le j\le r} H(m_{i,j})$. Here, *H*(.)is a one-way mapping function $\{0.1\}^{*}\rightarrow G_1$, and *h*(.) is a one-way secure hash function.

In the data integrity verification process, the prover needs to provide the verifier with auxiliary authentication information (abbreviated as AAI). AAI consists of the sibling nodes along the path from the leaf node to the root node in the hierarchical MHT. For different levels of granularity verification, examples include sub-chunks $x_{8.2}$, $x_{8.8}$ and chunks x_2, x_4. The verifier possesses the actual h_r. When the cloud storage server receives verification requests for sub-chunks $x_{8.2}$, $x_{8.8}$ and chunks x_2, x_4, the cloud storage server provides the verifier with reliable AAI, which are as follows:

$\Omega_{8,2}$ =<$h(x_{8,1})$,$h(x_{8,d})$,$h(x_{8,b})$,$h(x_7)$,$h(x_e)$,$h(x_a)$>,

$\Omega_{8,8}$ =<$h(x_{8,7})$,$h(x_{8,e})$,$h(x_{8,a})$,$h(x_7)$,$h(x_e)$,$h(x_a)$>

Ω_2 =<$h(x_1)$,h_d,h_b>, Ω_4 =<$h(x_3)$,h_c, h_b>

The verifier computes:

$h(x_{8,c})=h(h(x_{8,1})||h(x_{8,2}))$,$h(x_{8,a})=h(h(x_{8,c})||h(x_{8,d}))$,$h(x_8)=h(h(x_{8,a})||h(x_{8,b}))$,

$h(x_{8,f})=h(h(x_{8,7})||h(x_{8,8}))$,$h(x_{8,b})=h(h(x_{8,e})||h(x_{8,f}))$,$h(x_8)=h(h(x_{8,a})||h(x_{8,b}))$,

$h_f=h(h_7||h_8)$,$h_b=h(h_e||h_f)$,$h_c=h(h_1||h_2)$,$h_d=h(h_3||h_4)$,$h_a=h(h_c||h_d)$,$h_r=h(h_a||h_b)$,

Then, the verifier checks whether the computed h_r matches the actual one.

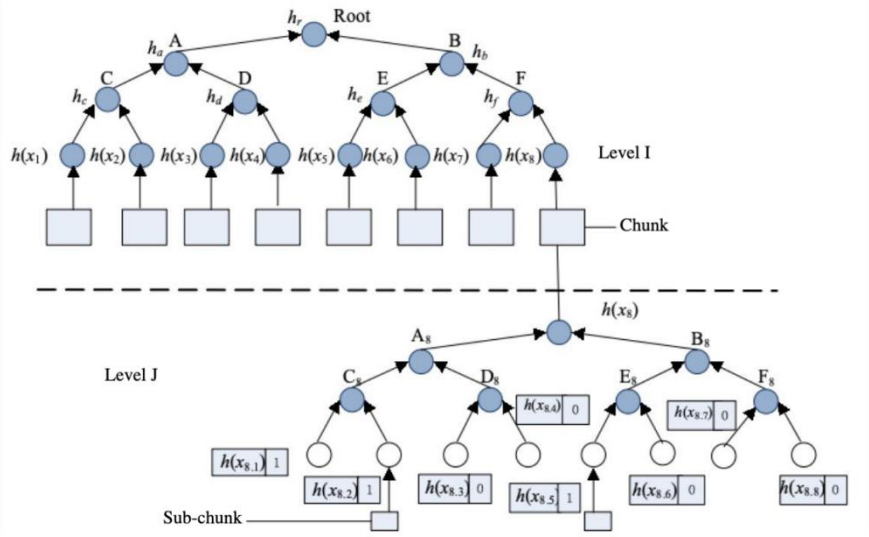

Figure 3 Hierarchical Merkle Hash Tree structure diagram

3. PAPER AND TEXT FORMAT A FLEXIBLE VERIFICATION SECURITY PROTOCOL FRAMEWORK

3.1 Support for Two Levels of Granularity in Integrity Verification Protocols

In this paper, the verification protocol supporting two levels of granularity involves three main stages and employs five algorithm functions. The five algorithm functions are listed in Table 2. The three stages are: the setup stage, the server verification stage, and the user or third-party verification stage. The five algorithm functions are: KeyGen, SigGen, VerifyUser, GenProof, and VerifyProof.

Table 2 The Five Algorithm Functions

Algorithm	Input	Output
KeyGen	A signature pair (*spk*, *ssk*) and security parameter *k*	Public key *pk* and private key *sk*
SigGen	Private key *sk* and outsourced data file *F*	Sub-blocks of the file, the set of signatures for the blocks Φ, Φ', and the metadata for the hierarchical Merkle hash tree root signature $sig_{sk}(H(R))$
VerifyUser	User file and signature	Accuracy of file data
GenProof	File *F*, set of signatures Φ, Φ', and challenge value *chal*	Aggregated results and data proof for the challenged sub-blocks or blocks
VerifyProof	Aggregated results and data proof	Verification results for the challenged sub-blocks or blocks

3.2 Support for Dynamic Data Operations at Two Levels of Granularity

In an untrusted environment, data integrity verification schemes can efficiently handle fully dynamic data operations based on a hierarchical Merkle hash tree as the storage verification structure. In a remote cloud storage environment, without retrieving the entire data file, dynamic update operations utilize four algorithm functions: VerifyUser, ExecUpdate, SearchUpdate, and VerifyUpdate, as shown in Table 3. Remote support for users includes operations such as insertion, deletion, and modification of data files.

Table 3 The Foure Algorithm Functions

Algorithm	Input	Output
VerifyUser	User file and signature	Accuracy of file data
ExecUpdate	Dynamic operation execution request	Updated file, set of signatures after F', operation Φ_{xin}, $\Phi_{\text{xin}'}$, and update proof P_{update}
SearchUpdate	Dynamic operation search request	Results of leaf nodes corresponding to the searched sub-blocks and blocks
VerifyUpdate	Update operation verification request	Verification results

3.2.1 Sub-block or block insertion

This paper will introduce block insertion; insertion of sub-blocks is similar. Suppose a client needs to insert a data block m_i^* after an existing data block (here $1 \le i \le n$). The specific protocol operation process is as follows:

(1) The user first decomposes the data block m^* into J data sub-blocks, represented as $(m_{i,1}^*, m_{i,2}^*, ..., m_{i,J}^*)$. Finally, each data sub-block $m_{i,j}^*$ is further decomposed into K basic blocks, referred to as $(m_{i,1,1}^*, m_{i,1,2}^*, m_{i,1,k}^*, m_{i,2,1}^*, ..., m_{i,J,K}^*)$. The user then signs the data block m_i^* to be inserted and the data sub-blocks that compose it as $\sigma_i^* = (H'(m_i^*).\prod_{k=1}^{k} u_k^{\sum m_{i,j,k}^*})^a$ and $\sigma_{i,j}^* = (H(m_{i,j}^*).u^{m_{i,j}^*})^a$. Next, the user constructs a Merkle Hash Tree (MHT) with the data block m_i^* as the root node and sends the data insertion request information "$update = (I, i, m_i^*, \sigma_i^*, \sigma_{i,j}^*)$" to the server.

(2) When the cloud server receives the insertion request, it first verifies the data block to be inserted using the VerifyUser algorithm and then performs the insertion operation. The process is as follows: First, store the data block m_i^* and save all its sub-block signatures $\sigma_{i,j}^*$. Using the SearchUpdate algorithm, locate node $h(H'(m_i))$ in the hierarchical Merkle Hash Tree (MHT) and retain its auxiliary authentication information Ω_i. Next, insert the MHT with $h(H'(m_i^*))$ as the root node, and add a new node as the parent of both node $h(H'(m_i))$ and node $h(H'(m_i^*))$. The value of this parent node is the hash of the concatenation of its child nodes $h(H(m_i))$ and $h(H'(m_i^*))$. Then, update the information of all sibling nodes along the path from this parent node to the root node of the tree, i.e, update the hash values of all nodes along this path, and compute the new root R'. Add the signature σ_i^* to Φ'. Finally, the cloud server sends the result of the data insertion operation $P_{update} = (\Omega_i, H'(m_i), Sig_{sk}(H(R), R'))$ back to the user.

(3) When the user receives the operation result, they first verify the validity of m_i using the auxiliary authentication information $\{\Omega_i, H'(m_i)\}$ and the verification algorithm. If the verification is invalid, the result is FALSE. If the verification is valid, calculate the old root node R based on the relevant information and use equation to verify the authenticity of the tree root R. If the equation does not hold, output FALSE. If the equation holds, compute the information of the new root R_{new} using $\{\Omega_i, H'(m_i), H'(m^*)\}$ and compare it with R'. If the values are different, output FALSE. Otherwise, output TRUE. The user then signs the new root node R' on the hierarchical MHT and sends the signature of the new root to the cloud server. Upon receiving the new root's signature, the cloud server saves the new root signature and deletes the old root signature file. Finally, the user performs a complete integrity verification of the blocks or sub-blocks. If the result is TRUE, it indicates that the server has completed the data insertion operation. At this point, the user deletes the locally stored information $\{Sig_{sk}(H(R')), P_{update}, m_i^*, \sigma_i^*, \sigma_{i,j}^*\}$.

3.2.2 Deletion of Sub-blocks or Blocks

Next, we will describe the deletion of a block, which is similar for the deletion of sub-blocks. Suppose the user needs to delete the *i*-th data block m_i (here $1 \le i \le n$). The specific deletion protocol process is as follows:

(1) The user first sends a data block deletion request information "$update = (D,i)$" to the cloud server.

(2) When the cloud server receives the data deletion request information from the user, it uses a search algorithm to find the node $h(H'(m_i))$ associated with the data block m_i and all corresponding sub-block nodes $h(H(m_{i,j}))$. After retaining the auxiliary authentication information Ω_i of $h(H'(m_i))$, the server deletes the node m_i and its parent node, as well as all corresponding sub-block nodes of m_i. It then takes the sibling nodes of m_i as the new parent node and updates the hash values of all nodes along the path from this modified parent node to the root node. The new root R' is calculated, and σ_i is deleted from Φ' and $\sigma_{i,j}$ is deleted from Φ. Finally, the result $\{\Omega_i, H'(m_i), Sig_{sk}(H(R), R')\}$ is sent to the user.

(3) When the user receives the deletion operation proof, they first use the verification algorithm to check whether the data block is the one intended for deletion. Then, they generate the root value R using $\{\Omega_i, H'(m_i)\}$. By verifying the equation, they confirm the authenticity of the auxiliary authentication information (AAI) and the root value R. If the equation verification fails, the result is FALSE. Otherwise, the user signs the new root with $sig_{sk}(H(R'))$ and sends the new signature back to the cloud server. Upon receiving the new root signature information, the cloud server saves it and deletes the old root signature file $sig_{sk}(H(R'))$. The user then performs an integrity verification of the data block or sub-block. If the result is TRUE, it indicates that the cloud server has completed the user's data deletion operation; otherwise, it indicates that the cloud server has not completed the user's data deletion operation.

3.2.3 Modification of Sub-blocks or Blocks

we will describe the modification of a block, which is similar for sub-block modifications. Suppose the user needs to modify the *i*-th data block m_i to m^*_i. The specific data block modification protocol process is as follows:

(1)The user first decomposes the data block m^*_i into *J* sub-blocks, represented as $(m^*_{i,1}, m^*_{i,2}, ..., m^*_{i,J})$. Each sub-block $m^*_{i,j}$ is further decomposed into *K* basic blocks, resulting in $(m^*_{i,1,1}, m^*_{i,1,2}, m^*_{i,1,k}, m^*_{i,2,1}, ..., m^*_{i,J,K})$. The user then signs the data block m^*_i and its constituent data sub-blocks $\sigma_i^* = (H'(m_i^*) \cdot \prod_{k=1}^{k} u_k^{\sum m^*_{i,j,k}})^a$ and $\sigma^*_{i,j} = (H(m^*_{i,j}) u^{m^*_{i,j}})^a$. Next, the user constructs a Merkle Hash Tree (MHT) with m^*_i as the root node and sends the data modification request information "$update = (M, i, m_i^*, \sigma_i^*, \sigma_{i,j}^*)$" to the server. Here, *M* represents the modification operation request, which updates the request to the server.

(2) Upon receiving the update request, the cloud server runs the VerifyUser and ExecUpdate algorithms. After verifying the data block m^*_i, it performs the modification operation: first, the data blocks m^*_i, $\sigma^*_{i,j}$, and the corresponding layer I leaf nodes $h(H(m^*_i))$ are stored. Then, using a search algorithm, the server finds node $h(H'(m_i))$in the hierarchical MHT and retains its auxiliary authentication information Ω_i. The server then updates the hash values of all nodes along the path from $h(H'(m^*_i))$ (the modified node) to the root node at layer I, and modifies node $h(H'(m_i))$ to be an MHT with $h(H'(m^*_i))$ as the root node. This results in a new root value R', and signatures σ_i^* and $\sigma_{i,j}^*$ are added to Φ' and Φ respecti vely. Finally, the cloud server responds to the user's operation by sending $P_{update} = \{\Omega_i, H'(m_i), Sig_{sk}(H(R), R')\}$ back to the user.

(3) After the user receives the proof of the modification operation from the cloud server, they first use the verification algorithm to check whether the data sub-block is the one intended for modification. They then generate the root value R using $\{\Omega_i, H'(m_i)\}$. The authenticity of the auxiliary authentication information (AAI) and the root value R is verified through the verification equation $e(sig_{sk}(H(R)), g) = e(H(R), v)$. If the equation verification fails, FALSE is output; otherwise, the user verifies whether the server has faithfully executed the data modification operation by further using $\{\Omega_i, H'(m^*_i)\}$ to compute a new root value. The newly calculated root value is compared with R'. If the values are not equal, FALSE is output; otherwise, TRUE is output. The user then signs the new root node R' with $sig_{sk}(H(R'))$, and sends the signature of the new root $sig_{sk}(H(R'))$ to the cloud server. Upon receiving $sig_{sk}(H(R'))$, the server saves it and deletes the old root signature file. At the same time, the server deletes the original signature of the data block σ_i and all signatures of the corresponding sub-blocks $\sigma_{i,j}$. Finally, the user performs an integrity verification on the data, treating it

as blocks or sub-blocks. If the output result is TRUE, the stored information $\{Sig_{sk}(H(R')),P_{update},m^*_i,\sigma^*_i,\sigma^*_{i,j}\}$ is deleted by the user's client.

4. SECURITY ANALYSIS

High security is always a primary concern. For this solution, while improving overall efficiency, the main concerns for users are whether data integrity and privacy can be adequately assured. For cloud storage servers, the key concerns are the correctness and completeness of the data submitted by users. The following security analysis addresses these concerns based on the protocol. The security analysis of the protocol is based on the Diffie-Hellman problem and the Discrete Logarithm Problem (DLP).

4.1 Data Correctness and Integrity for Users

When users send outsourced data files to a remote cloud storage server, they must ensure the correctness, consistency, and integrity of the data $\{F,t,\Phi,\Phi',sig_{sk}(H(R))\}$ and its associated parameters. The signature scheme ensures that the data is unforgeable, meaning $H'(m_i)$ and $H(m_{i,j})$ cannot be forged. Additionally, since $spk,g,v,\{w_k\}_{1\le k}$, $\{u_k\}_{1\le k}$ is public, if $\sigma_i/\sigma_{i,j}$ or $m_i/m_{i,j}$ provided by the user are incorrect, they will not satisfy the bilinear mapping equation. Due to the difficulty of the discrete logarithm problem, it is infeasible for users to forge $\sigma_i/\sigma_{i,j}$ and $m_i/m_{i,j}$ simultaneously, which guarantees their consistency. For the user's data to pass verification by the cloud storage server, the user must submit authentic metadata and signatures to the cloud storage server.

4.2 Privacy Protection

During the verification process, if μ'/μ'' is directly exposed to the Third Party Auditor (TPA), the TPA can collect enough combinations of identical blocks or sub-block equations, which would allow them to easily solve a set of linear equations to obtain the data file block $\{m_i\}_{i\in I}$ or sub-block $\{m_{i,j}\}_{1\le i\le I,1\le j\le J}$. Therefore, it is crucial to ensure that μ'/μ'' information is not exposed to the TPA during the verification process. In the verification proof $\{\bar{\sigma},\mu_j,Q_k,(H'(m_i),\Omega_i)_{i\in I}/(H(m_{i,j}),\Omega_{i,j})_{i\in I,j\in I_2},sig_{sk}(H(R))\}$ sent to the TPA, since μ_j is the value obtained after masking the information μ'/μ'' with the element O_j randomly selected by the server, the TPA knows Q_k but, due to the difficulty of the discrete logarithm problem, O_j remains unknown. Thus, the privacy of μ'/μ'' can be assured through μ_j. Based on the assumptions of the Diffie-Hellman problem, the TPA cannot obtain the value of μ'/μ''. Therefore, privacy protection is guaranteed [6].

5. CONCLUSION

This paper first proposes a flexible data integrity verification scheme that supports multiple granularities based on the different requirements of verification operations. It introduces various data verification methods for users, further enhancing the overall verification efficiency. Additionally, under a new security model where the cloud server, TPA, and users are in an untrusted environment, it adds verification processes for the cloud server and root verification. This enables timely and efficient detection of threats to users. Furthermore, for data updates, it introduces the option for dynamic operations, allowing both sub-blocks and blocks to be dynamically updated. This scheme increases users' control over data verification, enabling cloud storage to provide better and more tailored services to users. At the same time, it integrates the features of public auditability and privacy protection. Finally, through analysis and comparison, the proposed scheme is demonstrated to be feasible.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

FUNDING

This research was funded by the Scientific and Technological Research Program of Chongqing Municipal Education Commission (Grant No. KJQN202302403, KJQN202303111).

REFERENCES

[1] Yue Ru. Research on Data Storage and Management Technology in Cloud Computing Environment. Science and Technology Information, 2023, 21(21): 29-32.
[2] Dhakad N, Kar J. EPPDP: An Efficient Privacy-Preserving Data Possession With Provable Security in Cloud Storage. IEEE Systems Journal, 2022, 16(4): 6658-6668.
[3] Li Hao, Zhang Qian, Wang Lei. A Hybrid Approach for Cloud Data Integrity Verification Based on Blockchain Technology. Future Generation Computer Systems, 2022, 128: 542-556.

[4] Sun Xize, Zhou Fucai, Li Yuxi, et al. A Database Encryption Scheme Based on Searchable Encryption Mechanism. Journal of Computer Research and Development, 2021, 44(4): 806-819.
[5] Ding Y, Li Y, Yang W, et al. Edge Data Integrity Verification Scheme Supporting Data Dynamics and Batch Auditing. Journal of Systems Architecture, 2022, 128: 128.
[6] Hou Huiying, Ning Jianting, Huang Xinyi, et al. Verifiable Attribute-Based Timed Signature Scheme and Its Application. Journal of Software, 2023, 34(5): 2465-2481.