Hindawi
Security and Communication Networks
Volume 2022, Article ID 1870779, 15 pages
https://doi.org/10.1155/2022/1870779

Research Article

A Lightweight Data Integrity Verification with Data Dynamics for Mobile Edge Computing

Haiyan Wang, Yi Lin, and Fu Xiao

School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, China

Correspondence should be addressed to Haiyan Wang; wanghy@njupt.edu.cn

Received 4 November 2021; Accepted 7 February 2022; Published 4 March 2022

Academic Editor: Xiaolong Xu

As a special scenario of mobile cloud computing, mobile edge computing can meet the requirements of low latency of data integrity verification and support of mobility in mobile scenarios. However, most existing data integrity verification methods have relatively large computational overhead and few considerations of data dynamic update. To address the above problems, we propose a lightweight data integrity verification method that can support data dynamics in mobile edge computing scenarios. The proposed method is based on an algebraic signature and data integrity verification framework, which ensures security and reduces the computational overhead to achieve the requirement of lightweight. On this basis, analysis and proof of the feasibility, security, and privacy are given. At the same time, in order to support the dynamic update of the data, an optimized strategy based on matrix index is designed with low overhead. In comparison with other baseline methods, simulation experiments show that our method is superior in terms of computational overhead and has good performance in supporting data dynamics.

1. Introduction

With the development of cloud computing, mobile edge computing (MEC) has been proposed as a special scenario of mobile cloud computing (MCC), which plays an important role in mobile scenarios with low service latency [1, 2]. When verifying the integrity of data stored on remote servers, MEC can provide lower service latency and support mobility, which is more appropriate for data integrity verification in mobile scenarios than MCC.

Most of the existing data integrity verification methods are based on cryptography theory, signature strategy, and blockchain. Cryptography theory includes elliptic curve cryptography theory [3] and homomorphic verification [4], and signature strategy includes ZSS short signatures [5, 6] and aggregate signatures [7]. These methods can safely and reliably verify the data stored in remote servers. However, for mobile users, the computing performance and communication resources of their portable mobile devices are limited, such as when users are on a high-speed train or bus journey. Most of the existing methods for data integrity verification are based on mobile cloud computing scenarios, which consume a large computational overhead. Moreover, the methods have poor support for data dynamic update operations, which is not conducive to the dynamic update operation of user data in mobile scenarios. Therefore, data integrity verification in mobile scenarios to meet the rapidly growing needs of low latency, low computing overhead, and support for mobility has important research significance.

To solve the above problems, based on our previous work, combined with mobile edge computing and data integrity verification, we propose a lightweight data integrity verification method supporting data dynamics in a mobile edge computing scenario. The method can take into account both security and computing overhead to achieve lightweight requirements. In addition, considering the low performance of the user's mobile device and difficulty in independently verifying the integrity of the data, we introduce a third-party audit (TPA) and assume that it is not fully trusted to verify the security of our method. When the data block being queried is missing at the edge, cloud services may provide additional assistance, and our method will check all data blocks in the data set to eliminate any illegal operations that may exist. The main contributions of our work can be summarized as follows:

(i) Firstly, we design a system framework based on data integrity verification in mobile edge computing scenarios. In our framework, we use edge nodes to prestore the data blocks to be checked and use a semitrusted third-party auditor to verify the integrity of the data blocks that users need to query. Our method can ensure that the data blocks are securely protected, and users' privacy will not be disclosed.

(ii) Secondly, we propose a data integrity verification protocol based on algebraic signatures (ASDIV-MEC); this protocol can not only ensure security and reliable verification of data but also ensure low computational overhead in the case of verifying all data blocks to achieve lightweight requirements, allowing users to verify the integrity of data under acceptable computational and communication overhead. On this basis, the feasibility, security, and privacy of the algorithm and performance are analyzed and proved.

(iii) Thirdly, based on previous research, we propose an optimization strategy for data dynamic update, which uses the data dynamic operation based on matrix index to support dynamic update of user data in mobile scenarios and reduces the computing overhead of dynamic update.

(iv) Finally, we carry out a series of simulation experiments. Through simulation and comparison experiments, our method is superior to other methods in terms of computational overhead, which verifies the efficiency of the method. At the same time, we conduct comparative experiments on several data dynamic update operations to further verify that our optimization strategy has better performance than other methods.

The rest of the paper is organized as follows: Section 2 introduces the background of data integrity verification, mobile edge computing, and algebraic signatures. Section 3 expounds the system framework from the design goal and the architecture. Section 4 describes the content of the ASDIV-MEC agreement in detail. Section 5 analyzes and proves the performance of the proposed method. Section 6 gives the specific content of the dynamic update optimization strategy. In Section 7, a simulation experiment is carried out, and the experimental results are given. Finally, Section 8 summarizes this paper.

2. Background

In this section, we describe the related work. Firstly, the development of data integrity verification methods and the shortcomings of each method are introduced. Secondly, the application of data integrity verification in the mobile edge computing scenario is given. Finally, we introduce the algebraic signatures used in this paper.

2.1. Data Integrity Verification. Data integrity verification was first proposed by Deswarte et al. [8], who proposed two data integrity verification methods using hash operation and Diffie–Hellman key exchange protocol. However, the method based on hash operation requires a lot of calculation and communication overhead. Subsequently, Venkatesh et al. [9] considered using homomorphic encryption technology based on RSA signatures for integrity verification, but this method also requires a lot of computational overhead. On this basis, Ateniese et al. [10] proposed a probabilistic verification method using message authentication codes to reduce communication overhead. The work of Shacham and Waters [11] used the Boneh–Lynn–Shacham (BLS) signatures mechanism to construct a homomorphic encryption verifiable label and proved the security and reliability of the mechanism. Wang et al. [12] considered the characteristics of the Merkle hash tree and proposed using the Merkle hash tree to verify the correctness of the data block.

In recent years, the combination of data integrity verification and cloud computing has also been studied. Zhu et al. [6] proposed a cloud-IoT data integrity verification method combined with ZSS signatures. However, they use ZSS signatures-based data integrity verification, which is limited by the large computational overhead required and cannot verify all data blocks to ensure that 100% of the data blocks are complete. Shen et al. [13] proposed the use of algebraic signatures for data integrity verification, but this method did not achieve faster data processing and analysis in mobile edge scenarios to reduce latency and support mobility. Ren et al. [14] proposed a sensor data integrity verification mechanism based on bilinear mapping accumulators. Compared with other works, this method is considered to verify the integrity of the entire data set, but bilinear mapping-based methods require relatively high computational overhead, and they do not consider whether the verifier is secure and reliable, so they cannot completely ensure the correctness of data verification results. Fan et al. [7] used aggregate signatures to verify data integrity. X. Lu and Pan [15] proposed a security and lightweight integrity verification method for IoT mobile terminal devices, which ensures the privacy and efficiency of data sharing in the cloud and achieves relatively lightweight operations for data owners.

At the same time, based on the key characteristics of blockchain decentralization, many researchers have studied the data integrity verification scheme based on blockchain. Aiming at the shortcomings of existing data integrity verification schemes, Wang et al. [16] proposed a data integrity verification scheme based on blockchain, which greatly improved the efficiency and security of the verification process. In order to avoid overreliance on TPA, Yue et al. [17] proposed a data integrity verification framework for distributed edge cloud storage (ECS) based on blockchain, which adopted a Merkle tree with random challenge number to verify data integrity without relying on TPA. Similarly, Liu et al. [18] believed that the reliability of the framework

based on TPA was not satisfactory and then proposed a data integrity verification framework based on blockchain. While existing blockchain-based data integrity verification schemes can avoid the trust issues of TPA, they must face another challenge, the issue of huge computing and communication overhead.

2.2. Mobile Edge Computing. Mobile edge computing (MEC) is a special scenario of mobile cloud computing (MCC). It provides lower service latency than MCC. It is used to meet the low latency and high mobility requirements of data integrity verification in mobile scenarios to enhance the ability of IoT terminal devices to process data. With the development of 5G technology and the widespread application of mobile networks, MEC has received widespread attention. MEC has been applied to many fields, such as healthcare, education, and public services [2]. In recent years, the security issues of MEC have also attracted some attention. For example, Tong et al. [19] first studied the data integrity verification of mobile edge computing. They proposed two methods, which are suitable for users who want to verify the integrity of unilateral or multilateral data. On the basis of Zhu et al. [6], Wang et al. [5] proposed integrity verification based on ZSS signatures [20] in mobile edge scenarios. This method transfers the data integrity verification to the edge node closer to the user to provide lower delay and meet the user's strong mobility, but the consideration of computational overhead is relatively insufficient. Liu and Shen [4] proposed using homomorphic verification technology to ensure data integrity, but this method is similar to ZSS short signatures, which are based on bilinear mapping, and the problem is also that the computational cost is relatively high.

2.3. Algebraic Signatures. The algebraic signatures are a signature defined on the Galois field. It is a kind of hash function with algebraic properties; that is, the signature of taking the sum of a certain file block is the same as the signature of taking the sum of the corresponding block. Therefore, the algebraic signatures can be regarded as a kind of algebraic hash function, which can return a part of the data signature for data integrity verification, thereby saving the computational overhead of data integrity verification at the edge node. The algebraic signatures method in this paper is based on Mokadem and Litwin [21] and Schwartz and Miller [22]. Algebraic signatures are similar to cryptographically secure hash functions such as MD5 and SHA-1. But MD5 and SHA are not secure in terms of encryption because it is easy to deliberately construct two strings with the same signature [22].

Mokadem and Litwin [21] first used algebraic signatures in Scalable Distributed Data Structures (SDDSs) to check distributed files stored in distributed networks. Schwarz and Miller [22] used algebraic signatures to check data in remote servers. Later, Luo et al. [23] used algebraic signatures in the cloud to check the data possession in the cloud. In the method of Luo et al., a trusted third-party auditor is used to check data in the cloud. In addition, this method uses an index table method to support dynamic operations of data. Ping et al. [3] and Shen et al. [13] based on the work of [22, 23] further proposed the use of algebraic signatures in cloud computing to verify the integrity of data. However, these tasks are based on data integrity verification in cloud computing scenarios, which cannot meet the needs of low latency and strong user mobility.

3. System Model

3.1. Design Objective. Compared with the data integrity verification in the traditional MCC environment, the data integrity verification in the MEC environment faces additional and complex interaction problems, which are more challenging. Although previous work has conducted research on scenarios and data integrity verification, the consideration of low latency and low computing overhead is relatively insufficient, and the research on data dynamic update operations needs to be improved. Therefore, in this paper, we propose a lightweight data integrity verification method that supports data dynamics in mobile scenarios. Our goals are as follows:

(i) Design a data integrity verification framework in a mobile scenario and design an algebraic signature-based data integrity verification protocol (ASDIV-MEC) under this framework. The verification protocol will verify all required data blocks instead of randomly selecting data block verification and, on the premise of guarantee security, maintain a low computational overhead to meet the requirements of lightweight. At the same time, the protocol ensures that the semitrusted third-party auditor will not obtain the user's private information from the verification, ensuring the security and privacy of the entire verification process. On this basis, in order to ensure the complete verification of the data, the situation where a single edge node, multiple edge nodes, and the cloud collaborate together is considered.

(ii) A data dynamic update optimization strategy with a lower computational cost is designed for mobile users to dynamically update data. This strategy is based on an index matrix to support the correct update of mobile user data and ensure low computational cost.

3.2. Architecture of ASDIV-MEC. The overall architecture of our method is shown in Figure 1, which consists of four types of entities: users, edge, cloud, and semitrusted third-party auditor (semi-TPA). The following is a detailed functional description of these four main components:

User: the user is the initiator who queries the data block and requests verification of integrity. Each user has a different fixed or mobile device, and the location is also different. Through our method, the user quickly signs the uploaded data with low computational overhead, stores the data in the cloud, and sends the signatures to

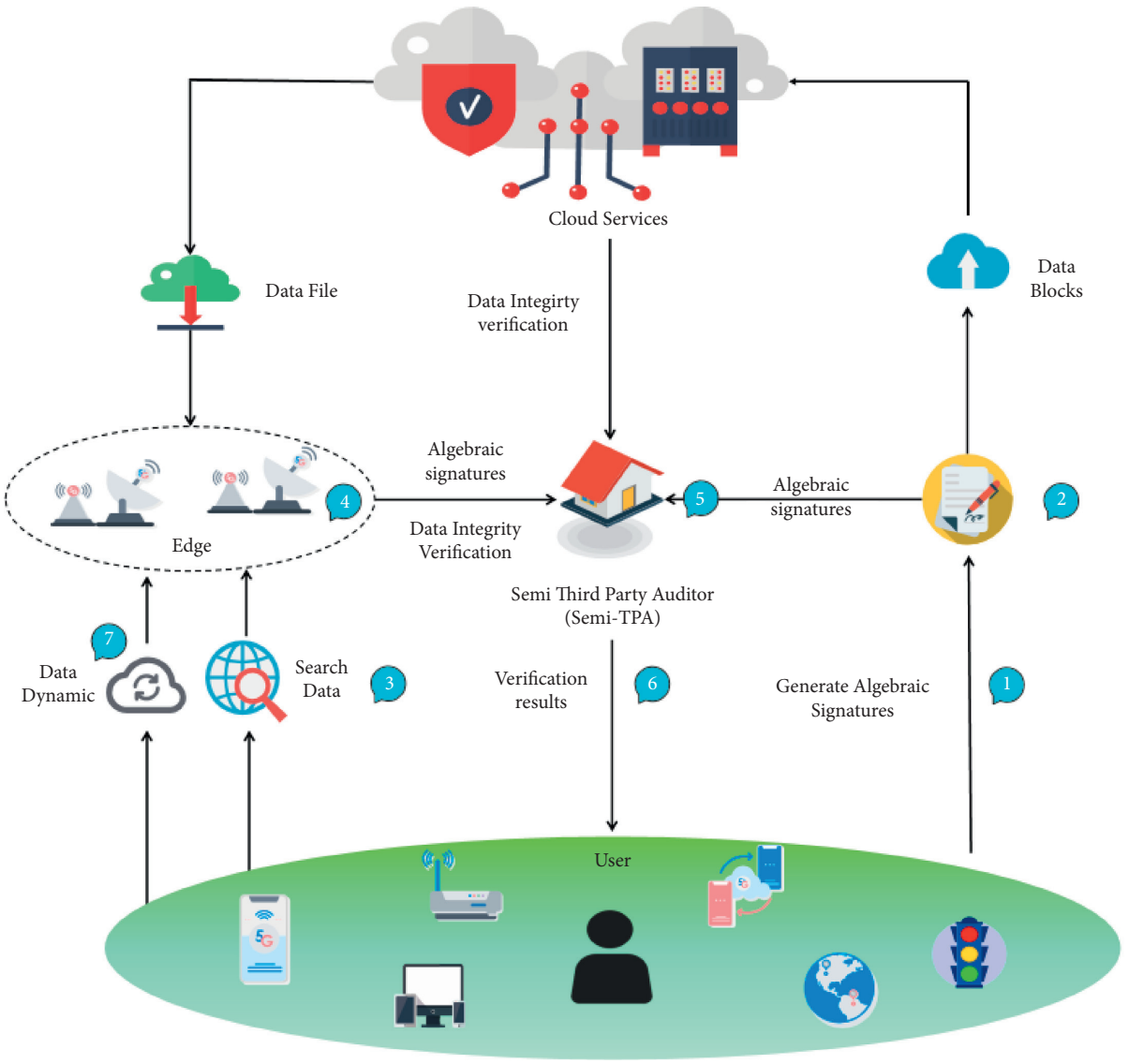

Figure 1: The architecture of ASDIV-MEC.

TPA. When the user needs to query a data block, a request is sent to the edge node through the method, and the verification result is obtained with lower latency and computational overhead so as to achieve a lightweight effect. User can also update data dynamically according to the data update optimization strategy, which consumes less computing overhead.

Edge: it is deployed at the edge node of the network close to the terminal device and the user. The edge node has the characteristics of miniaturization, distribution, and being closer to the user, which can realize the processing of data on the edge of the network, reduce the request and response time and computing overhead, and support users' strong mobility. After the user sends the integrity verification request, the edge node finds the corresponding data block from the predownloaded data, uses our method to quickly generate the corresponding proof with low overhead, and returns the result to the TPA for verification.

Cloud: it is a server that stores and processes user data. Each cloud service provider has powerful capabilities and huge storage space to provide comprehensive services with a short execution time, which greatly saves local storage space. In addition to storing the entire data file for the user, the cloud can also generate a certificate for the missing data block in the edge server and send it to the TPA.

Semi-TPA: it has powerful computing and storage capabilities and performs data integrity verification. Our method introduces TPA to reduce the user's computing overhead and the communication overhead between the user and the edge node and replace the user to perform integrity verification, which can meet

the lightweight requirements. TPA collects and verifies the proof sent from the edge node and returns the verification result to the user.

As shown in Figure 1, in the MEC environment, due to the low computing and storage performance of mobile devices, mobile users use our method to quickly divide data into multiple data blocks with a small amount of overhead and generate signatures for each data block and then upload the data blocks to the cloud storage device and delete the data on the local device. After that, the signature generated by the corresponding data is sent to TPA. In many studies, it is generally assumed that TPA is completely reliable. This is unrealistic because TPA may be attacked by external or internal attacks and return wrong results to users, posing a threat to data security. Our model system can handle this situation well, so we assume that TPA is semireliable. Edge nodes deployed near mobile terminals periodically predownload data blocks from remote clouds to provide data integrity services with low latency and computing overhead to achieve lightweight requirements. When a user requests to verify data blocks, the user sends the request to the edge node, which generates a proof of the data blocks to be verified with low computational overhead and sends it to the TPA. After that, TPA verifies whether the received data block proof and the data block signature uploaded by the user are correct according to the characteristics of the algebraic signature and returns the result to the user.

Users usually verify the integrity of data stored on edge nodes before querying the stored data. When the queried data is not predownloaded on the edge nodes, the cloud server will provide a certificate to help return the stored data to the TPA for verification. When a user needs to update stored data, he can send an update request through our method. The method adopts the data dynamic update optimization strategy to dynamically update the user's data with a low computational overhead to ensure lightweight operations.

4. Protocol of ASDIV-MEC

4.1. Preliminaries. This paper introduces a secure and efficient data integrity verification signature, which is a hash function with algebraic properties, that is, algebraic signatures, which can quickly sign data blocks to improve signature efficiency. Algebraic signature is a hash function with algebraic properties proposed by Schwarz and Miller [22]; that is, the sum of the signatures of a certain file block is the same as the signature of the sum of the corresponding block. Algebraic signatures have homogeneity and algebraic properties, consume less computational overhead when signing and verifying data, and can be used for lightweight integrity verification of remote data [22].

Assuming that a block of data in data file F is composed of $F_1, F_2, \ldots, F_n$, then the algebraic signatures have the following properties.

4.1.1. Compressibility.

$$AS_\alpha(F_1, F_2, \ldots, F_N) = \sum_{i=1}^{N} F_i \alpha^i. \tag{1}$$

It can be seen from equation (1) that algebraic signatures are compressible and can compress a file into a small string, where AS is the algebraic signature method and α is the algebraic signature parameter. When the original file is modified, the corresponding algebraic signatures value will also change accordingly. This attribute is similar to the hash functions MD5 and SHA in cryptography. However, when using MD5 and SHA, users need to keep all hash values and must retrieve all their own data, which causes huge communication and computational overhead. Therefore, MD5 and SHA are not suitable for remote data integrity verification [22]. The algebraic characteristics and compressibility of algebraic signatures enable users to check all data stored remotely with less overhead. Therefore, algebraic signatures are an ideal way to verify whether remote data is stored intact.

4.1.2. Algebraic Property. In addition, the technology has low communication and computing overhead. Suppose two large data files X and Y are composed of n subblocks, which are, respectively, expressed as $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$. The algebraic properties of algebraic signatures can be observed as follows:

$$\begin{aligned} AS_\alpha(X) + AS_\alpha(Y) &= AS_\alpha(x_1, x_2, \ldots, x_N) + AS_\alpha(y_1, y_2, \ldots, y_N) \\ &= \sum_{i=1}^{N} x_i \alpha^i + \sum_{i=1}^{N} y_i \alpha^i = \sum_{i=1}^{N} (x_i + y_i)\alpha^i = AS_\alpha(X + Y). \end{aligned} \tag{2}$$

Therefore, algebraic signatures enable edge nodes to return a portion of the data signature for data integrity checking, thus saving bandwidth for edge node data integrity checking applications. In the face of the diversity of data, the following expansion can also be carried out:

$$\begin{aligned} &AS_\alpha(X) + AS_\alpha(Y) + \cdots + AS_\alpha(Z) \\ &= AS_\alpha(x_1, x_2, \ldots, x_N) + AS_\alpha(y_1, y_2, \ldots, y_N) + \cdots + AS_\alpha(z_1, z_2, \ldots, z_N) \\ &= \sum_{i=1}^{N} x_i \cdot \alpha^i + \sum_{i=1}^{N} y_i \cdot \alpha^i + \cdots + \sum_{i=1}^{N} z_i \cdot \alpha^i = \sum_{i=1}^{N} (x_i + y_i + \cdots + z_i) \cdot \alpha^i = AS_\alpha(X + Y + \cdots + Z). \end{aligned} \tag{3}$$

Algebraic signatures consist mainly of the following three functions:

KeyGen: in the KeyGen phase, it generates some initialization parameters, such as the master key k, the signatures parameter, and some random parameters.

Sign: the algebraic signatures of a data block are as follows: $AS_\alpha(F_i) = F_i \cdot \alpha^i$.

Verify: given a fixed key k, data block F_i', and signature Sig, the verifier needs to verify that the signature of the stored data block is equal to the original signature. If they are equal, then the signature is generated by the user who has the signature, and verification can be confirmed as follows: $AS_\alpha(F_i') = \text{Sig}$.

For the sake of clarity, we list some notations and their descriptions in Table 1, which will be used throughout the paper.

4.2. ASDIV-MEC. ASDIV-MEC includes five stages: parameter generation stage, signature generation stage, challenge generation stage, proof generation stage, and verification proof stage. We use **Paragen**, **SigGen**, **ChallGen**, **ProofGen**, and **VerifyProof** to represent these five stages. These five stages are described in detail as follows.

Step 1. KeyGen() $\longrightarrow$, (k, a, r_1, r_2): first, some parameters need to be generated in the initial stage. The user needs to generate a master key k and signature parameter a from the secure hash function. Meanwhile, the user and the TPA generate two security parameters r_1 and r_2, according to the secure hash function. Finally, the TPA sends r_2 to the user over a secure channel between the client and the TPA.

Step 2. SigGen $(F, k, r_1, r_2) \longrightarrow$ (Sig): this stage generates an algebraic signature for each data block. The user divides the data file F into blocks: $\{m_1, m_2, \ldots, m_n\}$. Since TPA is not necessarily honest, in order to ensure the security of data blocks, data owners need to preprocess data blocks $m_i \in F$ to prevent the disclosure of user data when TPA is attacked.

$$F_i = m_i \oplus H(r_1 \| i), \tag{4}$$

where H is a one-way hash function with collision resistance, used to protect security parameter r_1 and block of the confidential nature of ID_i, and $\|$ is a concatenation operation.

From the above operation, the data block of data file $F = \{F_1, F_2, \ldots, F_n\}$. According to our method, the signature of block i can be generated with little overhead.

$$\text{Sig}_i = F_i \cdot a^{i \cdot r_2}. \tag{5}$$

On this basis, the signature of the data file F is $\text{Sig} = \{\text{Sig}_1, \text{Sig}_2, \ldots, \text{Sig}_n\}$, which reduces communication overhead and facilitates support for TPA auditing.

After that, the user sends the data file block $F = \{F_1, F_2, \ldots, F_n\}$ that is uploaded to the cloud server for storage, and the edge node periodically downloads the data block that the user needs to query in advance and signs the data block $\text{Sig} = \{\text{Sig}_1, \text{Sig}_2, \ldots, \text{Sig}_n\}$ that is uploaded to TPA.

Step 3. ChallGen (r_1), $\longrightarrow$, (chall): when a user wants to check whether some of his data is stored intact in the cloud, the user initiates a data integrity request. The user first generates a random x in the Galois field and calculates $c_k = f_k(r_1 + x)$, generates the challenge $\text{chall} = \{(c_k, x)\}$, and sends it to the edge node and TPA.

Step 4. ProofGen (chall, r_2), $\longrightarrow$ $(proof)$: the edge node stores all the data that users need to query. After receiving the chall, our method requires the edge node to generate a certificate based on the algebraic signatures, generate a certificate for the data block to be queried with a lower computational cost, and reply to the TPA with a storage certificate. Considering the possible lack of data transmission between the cloud and edge nodes, we consider three cases in Figure 2 in this step, which are described in detail as follows:

Case 1 (Single Edge). The edge node first calculates the position of the selected block, as shown as follows:

$$l_i = \sigma_k(r_2 + i), \quad 0 \le i \le c. \tag{6}$$

And let $L = \{l_i\}_{0 \le i \le c}$, c is the number of blocks per chall.

Then the edge node computes the algebraic signatures of the sum of the selected blocks:

Table 1: Summary of main notations.

Notations	Descriptions
F	Data file
m_i	i^{th} data block of data file F
n	Number of data blocks
k	Master key
a	Signature parameter
r_i	Security parameter
H	Hash function
c_k	Pseudorandom number
x	Random number generated in Galois field
Chall	Challenge request chall = $\{(c_k, x)\}$
Proof	Proof of data blocks
l_i	The position of i^{th} data block
c	The number of blocks per chall
t	Number of edge nodes working together in Case 2
I	Queried data block index set

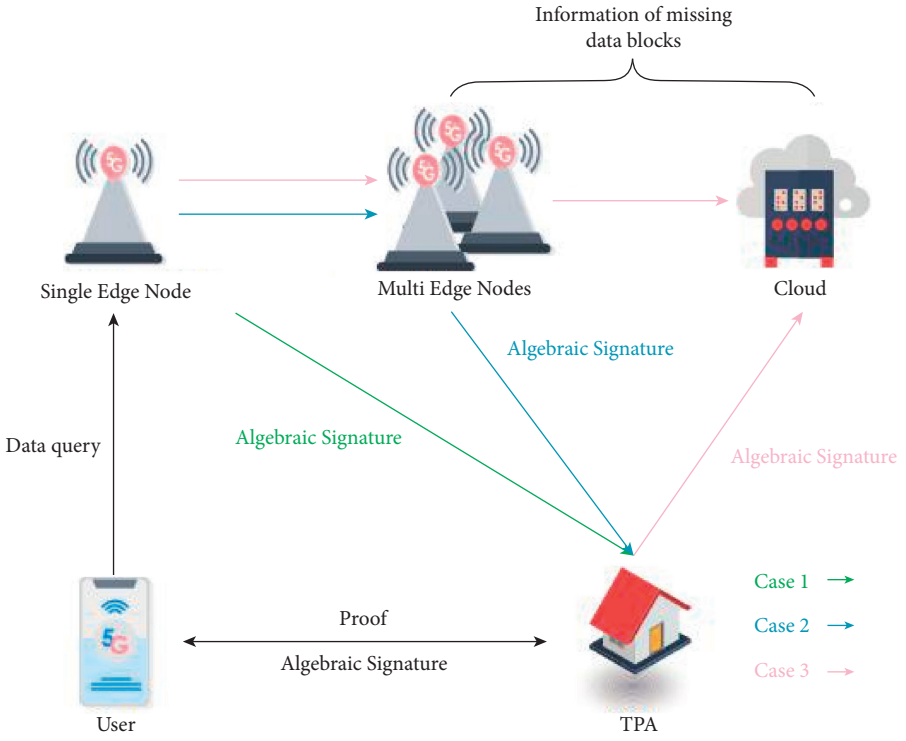

Figure 2: Three cases in our verification protocol.

$$\beta = AS_{\alpha}\left(\sum_{l \in L} F_l\right). \tag{7}$$

The signature of the sum of the computed data blocks is sent to TPA as proof.

Case 2 (Multiple Edges). In this case, a single edge node cannot provide enough data blocks, and it queries nearby edge nodes for help. Let us assume that there are T edge nodes working together to solve this tricky problem, and each edge node has its own index set of data blocks O_j, $j \in \{1, 2, 3, \ldots, t\}$; on this basis, for

every $j \in \{1, 2, 3, \ldots, t\}$, edge node E_j can extract the valid data block index set for the missing data block, as shown as follows:

$$\begin{aligned} I_j &= \left(I - (I \cap O_1) \cup (I \cap O_2) \cup \ldots \cup \left(I \cap O_{j-1}\right)\right) \cap O_j \\ &= \left(I - I \cap \left(O_1 \cup O_2 \cup O_3 \cup \ldots \cup O_{j-1}\right)\right) \cap O_j \\ &= I \cap O_j - I \cap O_j \cap \left(O_1 \cup O_2 \cup O_3 \cup \ldots \cup O_{j-1}\right) j \in \{1, 2, 3, \ldots, t\}. \end{aligned} \tag{8}$$

We define the edge node E_t as the last one if $O_1 \cup O_2 \cup O_3 \cup \ldots \cup O_{t-1} \cup O_t = I$. Assume that $I_j = \left\{s_1, s_2, \ldots, s_{e_j}\right\}$ and $e_j \in \{1, 2, 3, \ldots, n\}$; according to formula (8), we can get

$$\beta = AS_\alpha \left(\sum_t \sum_{l \in L} F_l \right). \tag{9}$$

The last edge node sends the signature of the sum of the computed data blocks to TPA as proof.

Case 3 (A Joint of Multiple Edges and Cloud). In this case, a single edge node and its nearby edge nodes cannot provide enough data blocks and will eventually seek help from the central cloud. The last edge node communicates with the central cloud for the missing data block. The calculation is similar to Case 2. Finally, the central cloud returns the integrated evidence to the TPA.

Step 5. Verify(*chall*, *proof*) $\longrightarrow$ \{TRUE, FALSE\}: when TPA receives the storage proof *proof* from the edge node, it verifies the correctness of *proof*

$$\text{proof} = \sum_n \text{Sig}_i. \tag{10}$$

If the above equation is true, TPA will return true to the user to confirm that the outsourced data is complete; otherwise, it will return false to notify the user that the outsourced data is corrupted.

5. Performance of Analysis

In this section, we analyze the performance of the proposed ASDIV-MEC model system from three aspects, namely, feasibility, security, and privacy. Both edge nodes and TPA are semitrusted in our model. For the sake of description, we examine the case of Case 1 in detail.

5.1. Feasibility

Lemma 1. *If both TPA and edge nodes can carry out data transmission and communication normally, then the data integrity verification scheme ASDIV-MEC proposed in this paper is feasible.*

Proof. According to the protocol described earlier, if the data block on the edge node is stored intact, then TPA can verify the correctness through the proof generated by the edge node and the user-generated signature. We can accurately infer and verify the feasibility of the scheme through the following calculations:

$$\begin{aligned} AS_\alpha(X) + AS_\alpha(Y) &= AS_\alpha(x_1, x_2, \ldots, x_N) + AS_\alpha(y_1, y_2, \ldots, y_N) \\ &= \sum_{i=1}^{N} x_i \cdot \alpha^i + \sum_{i=1}^{N} y_i \cdot \alpha^i = \sum_{i=1}^{N} (x_i + y_i) \cdot \alpha^i = AS_\alpha(X + Y). \end{aligned} \tag{11}$$

From the above derivation that the sum of the user's signatures for each data block is equal to the sum of the signatures of the data block stored in the edge node, it can be concluded that our verification algorithm is feasible. □

5.2. Security and Privacy

Lemma 2. *If our model system is maliciously attacked, TPA can detect data file corruption.*

Proof. Assume that the edge node is attacked and tamper with the data block F_i stored in the edge node. If an attacker wants to pass TPA authentication, then they need to build an alternative signature:

$$\text{Sig}' = AS_\alpha(X^*) = \text{Sig}. \tag{12}$$

Make

$$AS_\alpha(X^*) + AS_\alpha(Y) + \cdots + AS_\alpha(Z) = AS_\alpha(X + Y + \cdots + Z). \tag{13}$$

In order to ensure the security of the original data, this paper uses the hash function and random parameters to blind the original data block and uses the hash function to encrypt the security parameters r_1 and i, which can not only protect the security parameters but also resist forgery attack. The data file is hidden in $F_i \cdot a^{i \cdot r_2}$, and a is the security parameter randomly generated by the data owner in the *KeyGen* stage. In addition, a is masked by the data block i and the user identifier r_1, enhancing the privacy of the data.

Therefore, if the attacker modifies the block m_i, but cannot obtain the user's security parameter r_1, then the following cannot be constructed:

$$F_j^* = m_j^* \oplus H(\cdot) = F_i = m_i \oplus H(\cdot). \tag{14}$$

Thus, (13) cannot be verified through TPA, so the attacker cannot pass TPA verification by constructing a new signature. □

Theorem 1. *When the attacker attacks the TPA or the edge node, the attacker cannot obtain the user's private information by intercepting the signature information.*

Proof. In our approach, TPA has a signature set $Sig_a = \{Sig_i\}$, and the signature set is the hidden user data information generated by $F_i \cdot a^{i \cdot r_2}$. Even in batch authentication, the information of multiple users will be hidden in the data block. Therefore, the security parameters block ID and signature parameters to hide the user's private information. □

Theorem 2. *If the TPA and edge nodes are honest, the integrity verification process is feasible and secure.*

Proof. On the basis of Lemmas 1 and 2, we can prove the theorem directly. □

6. Data Dynamics Optimization Strategy

In mobile scenarios, users need to update the data stored on the server, which consumes a large amount of overhead. However, the limited computing resources of the devices carried by users are not conducive to the data dynamic update in mobile scenarios. Therefore, it is necessary to develop corresponding optimization strategies for the data dynamic update to reduce the computing cost of data update and support the data dynamic update operation of users in mobile scenarios, which has important research significance.

According to the validation method proposed in Section 4, if all entities can honestly communicate and transmit data, we design a data dynamic update optimization strategy that allows users to dynamically insert, delete, and modify data blocks with low computational overhead. We propose two functions: UpdateReq, an update request algorithm, and UpdateExec, an update execution algorithm, to implement dynamic update operations on data blocks. Update operations include adding, deleting, and modifying data blocks. Some existing methods to realize data dynamics mainly focus on using linked lists [3], index tables, and trees [5]. Reference [13] proposed using the matrix to realize the dynamic change of data, and through simulation experiments, it is concluded that the calculation cost of using the matrix method is lower than that of using the index table and tree method. Therefore, this paper uses a matrix-based approach to achieve dynamic data change.

First, we represent the file block in the form of a matrix index. Each row index of the matrix corresponds to a data block, as shown as follows:

$$F = \begin{bmatrix} F_1 & f_{11} & \cdots & f_{1k} \\ F_2 & f_{21} & \cdots & f_{2k} \\ F_3 & f_{31} & \cdots & f_{3k} \\ \vdots & \vdots & \ddots & \vdots \\ F_n & f_{n1} & \cdots & f_{nk} \end{bmatrix}. \tag{15}$$

$F_1, F_2, \ldots, F_n$, respectively, represent the index value of the data block; f_{ij} represents the data subblock j under data block i. If the data subblock exists in the edge node, it is represented by 1; if it does not exist, it is represented by 0. The following matrix index can be obtained:

$$M_I = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 2 & 1 & \cdots & 1 \\ 3 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ n & 1 & \cdots & 1 \end{bmatrix}. \tag{16}$$

When a data block is modified, the index matrix needs to be changed accordingly. Because of the hierarchical structure of data, we can modify both a data block and a data subblock, which is very suitable for the operation of a large amount of data on edge nodes. The index matrix is managed by the user, and the edge nodes and cloud services process the data according to the index matrix provided by the user.

UpdateReq: a function that handles user requests. The input is $\langle \text{BlockOperation}, \text{Index}, M \rangle$, where BlockOperation is based on the specific data block requested by the user, *In de x* is the index of the updated data block, and M is the index matrix. The output is $\langle \text{BlockOperation}, \text{Index}, \text{block}', t', M' \rangle$, where $block'$ is the updated data block, t' is the updated signature, and M' is the updated matrix index.

UpdateExec: it handles functions executed on the edge server. The input is the output of UpdateReq, which is a new copy of the file. After each update, the user can perform challenge validation to ensure that the update operation is correct.

6.1. Insertion Operations. Data insertion includes data block insertion and data subblock insertion. When a user wants to insert a data block after a data index, he finds the row in the index matrix that needs to be inserted and then inserts the new row in the next row. The change process of the index matrix of data block insertion is shown in Figure 3.

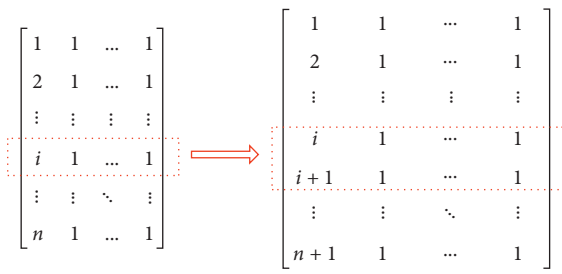

Figure 3: Data block insertion.

The insert operation also supports the insertion of data subblocks. When the user wants to insert a new data block $f_{i,i+1}$ after the data subblock $f_{i,i}$, the system needs to determine the index position of the inserted block. After that, the system will add a column to the index matrix, move the 1 after the index i backward, and finally insert the data subblock at the position of the index $i + 1$. The changing process of the index matrix is shown in Figure 4. No subblocks are inserted except for the index i row, and the other data blocks remain unchanged.

6.2. Deletion Operations. Similarly, data deletion operations include two parts: block deletion and subblock deletion. During block deletion, when you need to delete data block F_i, set the corresponding row matrix to -1, and the row index changes accordingly. The M_{BID} block index deletion matrix is used to assist in data deletion as follows:

$$M_{\text{BID}} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -i & -1 & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{17}$$

During the subblock deletion process, if the subblock $f_{i,i}$ needs to be deleted, the value of the relative position of index matrix S in the deletion matrix can be changed to -1. Therefore, we can write the subblock index deletion matrix M_{SID} as the following matrix:

$$M_{\text{SID}} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -1 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{18}$$

As mentioned above, the edge node has an index matrix and a modification matrix for the data. If we need to delete the data block F_i, we can use the index matrix plus the corresponding block index delete matrix to get a new index matrix NM_I. The block F_i deletion process can be written as follows:

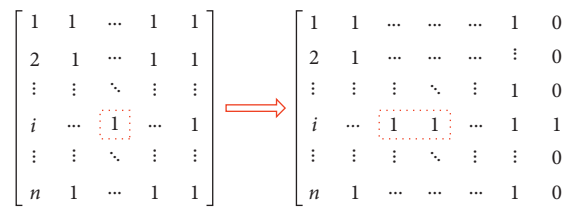

Figure 4: Data subblock insertion.

$$NM_I = M_I + M_{\text{BID}}. \tag{19}$$

If the subblock $f_{i,i}$ needs to be deleted, the index can be modified using the index matrix plus the corresponding subblock index deletion matrix. The new index matrix NM_I can be obtained by

$$NM_I = M_I + M_{\text{SID}}. \tag{20}$$

Using matrix addition operation, the proposed data dynamics method can reduce the deletion operation overhead [13].

6.3. Update Operations. Another common operation is data update. When a block or subblock is updated, the index is first found and deleted, and then the new block or subblock is inserted into the original block location.

Assume the data block F_i is updated to F_i'. First, find the i-th row of the index matrix. Second, delete the values of the matrix rows. Again, after the new data is inserted into the original data location, the corresponding index value is updated. The block update process in an index operation is shown in Figure 5.

Similarly, our scheme supports subblock updates, as well as the other two dynamic operations. To update the data subblock f_{ii} to f_{ii}', the corresponding index value needs to be found and deleted. After the new subblock is inserted, the value of the index matrix is updated. The operation method is similar to data block update. The detailed process of index matrix operation is shown in Figure 6.

Through the above analysis, we can prove the dynamic nature of our integrity verification algorithm. None of these changes breaks the signature policy technology, so the privacy of the data is still protected.

For Cases 2 and 3, we can get similar results because the main difference in performance compared to Case 1 is the additional communication costs from the edge nodes and the cloud.

7. Performance Evaluation

7.1. Experimental Settings. In the experiment, we carried out experiments on the proposed prototype system. For the four entities involved in the system, namely, users, edge nodes, CSP, and TPA, we deployed these entities using machines with different configurations according to the roles and functional requirements in the system model. CSP and TPA are deployed on two Dell Precision T9720 tower servers. The user and edge nodes are deployed on two different laptops.

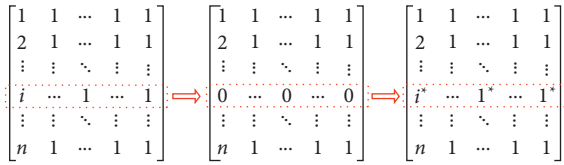

Figure 5: Data block update.

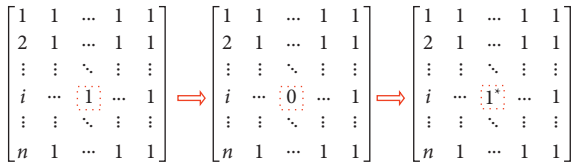

Figure 6: Data subblock update.

Table 2 shows the detailed parameters of our prototype system.

The experiment was carried out under the PBC library-0.5.14, GMP library-6.1.2, and VC ++ 6.0 software environment. In the experiment, the key size is 160 bits, and the pseudorandom number size is 80 bits. These data were the mean values of 40 replicates.

7.2. Baseline Methods. In order to prove the efficiency and effectiveness of our ASDIV-MEC method, two traditional signature methods (RSA signatures and BLS signatures) and two recently worked signature methods (aggregate signatures and ZSS short signatures) are compared with our method.

(1) RSA signatures: the RSA signatures bit ranges from 1024 to 4096. Under the same security condition, the BLS signatures are shorter than the RSA signatures.

(2) BLS: because the RSA algorithm mainly relies on the difficulty of factorization of a large integer, the calculation cost of the RSA-based method is high.

(3) Aggregate signatures scheme: Fan et al. [7] used aggregate signatures to verify data integrity. Aggregation signature is a variant signature method used to aggregate any multiple signatures into a signature. It can combine the public key and signature of each participant in a multisign transaction into a single public key and signature. The entire merge process is invisible, the premerge information cannot be derived from the merged public key and signature, and the verification only needs to be done once.

(4) ZSDIV-MEC: ZSS signature is a bilinear pin-based short signature proposed by Zhang et al. [20], which is based on encrypted hash functions such as SHA-2 or SHA-3. While the overhead of a ZSs-based signatures system is smaller than that of BLS and RSA, the user must store the hash of SHA and then retrieve the entire data file from the edge node or cloud to verify its integrity, resulting in significant communication and computing costs.

Table 2: Experiment environments.

Entity	Experiment environments		
	Device	CPU	RAM (GB)
TPA	Server	Intel XEON silver 4210 @2.20 GHz × 20	64
CSP	Server	Intel XEON silver 4210 @2.20 GHz × 20	128
Edge	Laptop	Intel Core i7 @2.70 GHz × 4	16
User	Laptop	Intel Core i7 @2.70 GHz × 4	16

7.3. Performance Comparisons

7.3.1. Response Time for Different Cases. Firstly, we evaluate the calculation cost of each signature policy in the edge node to compare the performance of each signature policy. We choose the time cost as the index; that is, the time cost is related to the number of data blocks queried. In this experiment, we set a total of 200 data blocks, and the abscissa represents the number of data blocks queried and validated by the user. The size of each data block is 64 kb. The ordinate indicates the query time. At the same time, since there are three cases of our strategy, experiments are conducted on the calculation overhead of queries in different cases.

Figure 7(a) shows the experimental results of Case 1, where the edge node has predownloaded all the data blocks requested by the user for verification. As shown in Figure 7(a), with the increase of the number of queried data blocks, the time cost of the five schemes is getting higher and higher. The RSA-based scheme has the highest time cost, and the time cost increases exponentially. The performance of the scheme based on BLS is slightly worse than that based on aggregate signatures scheme and ZSDIV-MEC, but better than that based on RSA. At the same time, the response time of our method is better than that of the baseline, so the ASDIV-MEC policy is better than the three baseline policies, which illustrates the low computational cost and effectiveness of algebraic signatures in data integrity verification, and meets the requirements of lightweight.

Figure 7(b) shows the experimental results of Case 2. In this experiment, we set a total of 5 edge nodes. It can be seen from the experimental results that our scheme also has the lowest cost, and the RSA-based scheme has the highest cost. In addition, in the case of multiple edge nodes, the response time of each scheme increases significantly compared with the case of a single edge node, which is due to the additional communication costs between the edge nodes. Figure 7(c) shows the experimental results of Case 3. Obviously, the increasing trend is very similar to Case 2. However, for each scenario, the overall response time of Case 3 is slightly greater than that of Case 2 because of the residual costs due to increased communication between the edge nodes and the cloud.

As can be seen from the experimental results, compared with the other three baselines, the proposed ASDIV-MEC solution has lower latency and low computational overhead and can better meet the requirements of lightweight.

7.3.2. Computation Overhead for KeyGen and SigGen. As we all know, *KeyGen* and *SigGen* are two important steps in data integrity verification. And depending on the protocol,

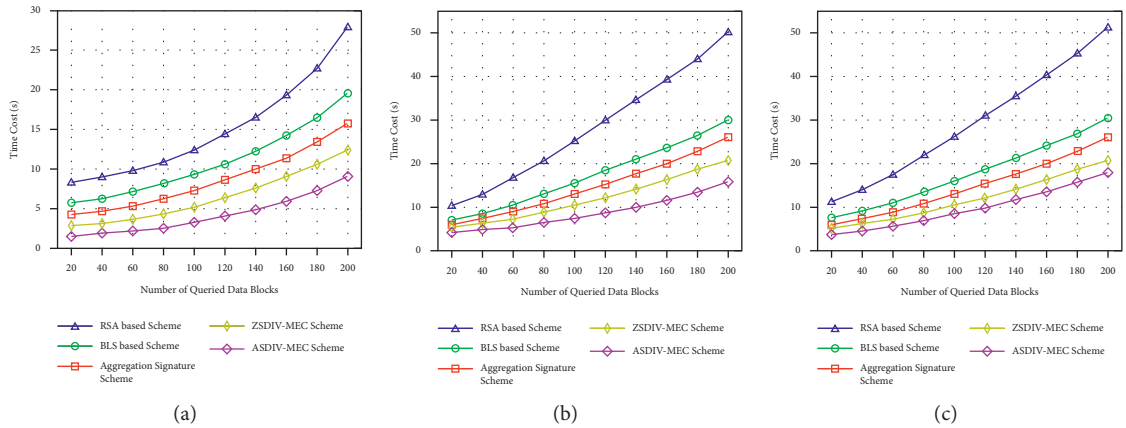

Figure 7: Comparison of response time. (a) Single edge. (b) Multiple edges. (c) A joint of multiple edges and the cloud.

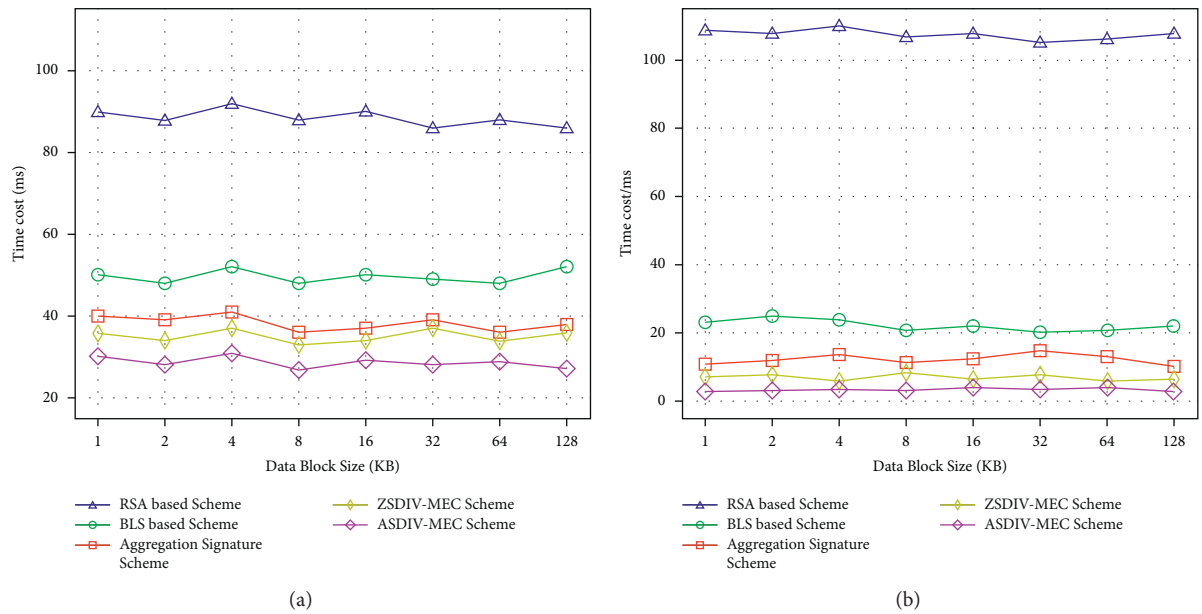

Figure 8: Computation overhead for (a) *KeyGen* and (b) *SigGen*.

the process is very different. Therefore, after comparing the overhead on the edge nodes, we further conducted experiments on the other two important steps in the strategy, *KeyGen* and *SigGen*, for different schemes and calculated the computational overhead of each scheme.

In this experiment, we evaluated the time overhead of key generation and signature generation for data block sizes ranging from 1 kB to 128 kB. As shown in Figure 8(a), we can see that, with the increase of data block size, the proposed time cost plan of ASDIV-MEC is about 20–30 ms, while the time cost plan based on aggregate signatures scheme, ZSDIV-MEC, and BLS is about 30–40 ms. The time cost of the RSA-based solution was much higher than the other three solutions at about 90 milliseconds, nearly three times the cost of the plan. As shown in Figure 8(b), it is clear that our proposed scheme always outperforms the baseline during the signature generation phase. This is because the computational security of the RSA algorithm depends on the difficulty of factorizing large integers, while the BLS signatures require a specific hash function, which is especially efficient for large-scale data. The strategy based on aggregation signatures is based on bilinear mapping, which leads to high overhead. Although ZSS signatures use general hash functions (such as SHA-2 and SHA-3), the calculation of its short signatures is complicated and increases the time cost, while the proposed ASDIV-MEC is relatively simple. Therefore, in terms of *KeyGen* and *SigGen* for data integrity verification, our ASDIV-MEC has better performance in terms of computational overhead, which can improve the efficiency of signature and meet the requirement of lightweight.

7.3.3. Computation Overhead Comparison. In order to better explain the reasons for the low latency and low computational overhead of ASDIV-MEC, we compared the

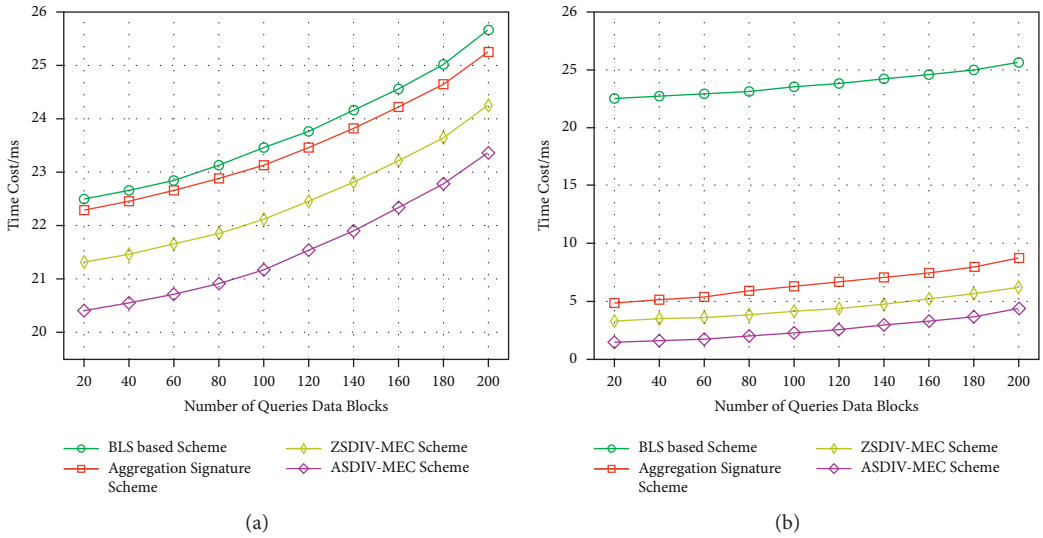

Figure 9: Computation overhead on (a) *User* and (b) *TPA*.

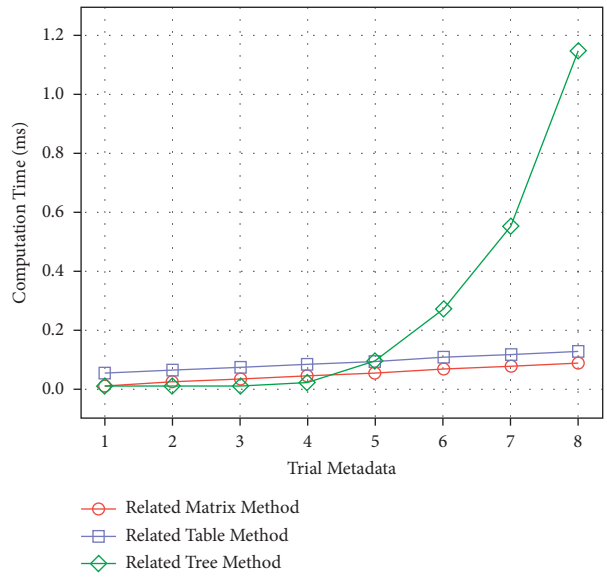

Figure 10: Comparison of different methods.

computational complexity of the BLS-based method, the ZSDIV-MEC method, and our method. Figure 9 shows the calculation times for user and TPA.

As can be seen from Figure 9(a), with the same number of data blocks, the ASDIV-MEC scheme spends less time on users than the scheme based on BLS, aggregate signatures scheme, and ZSDIV-MEC. This is because an algebraic signature is a signature similar to a hash function, but with relatively low computational complexity. As shown in Figure 9(b), due to the algebraic signatures, our protocol also takes less time than the schemes based on BLS, aggregate signatures scheme, and ZSDIV-MEC. By comparing the computational overhead of the BLS-based method, the aggregate signatures scheme-based method, the ZSDIV-MEC method, and our ASDIV-MEC method, the lightweight performance of the method is further proved.

7.3.4. Comparison of Computational Overhead for Data Dynamics. To verify the low computational overhead of our proposed data dynamics operation, we compare three main data dynamics strategies. Figure 10 shows three data dynamic update schemes. It can be seen from Figure 10 that the computing cost of the tree-based scheme increases exponentially, while that of the other two schemes increases linearly. This is because the tree-based scheme needs to calculate split subtrees, making its cost much higher than the other two.

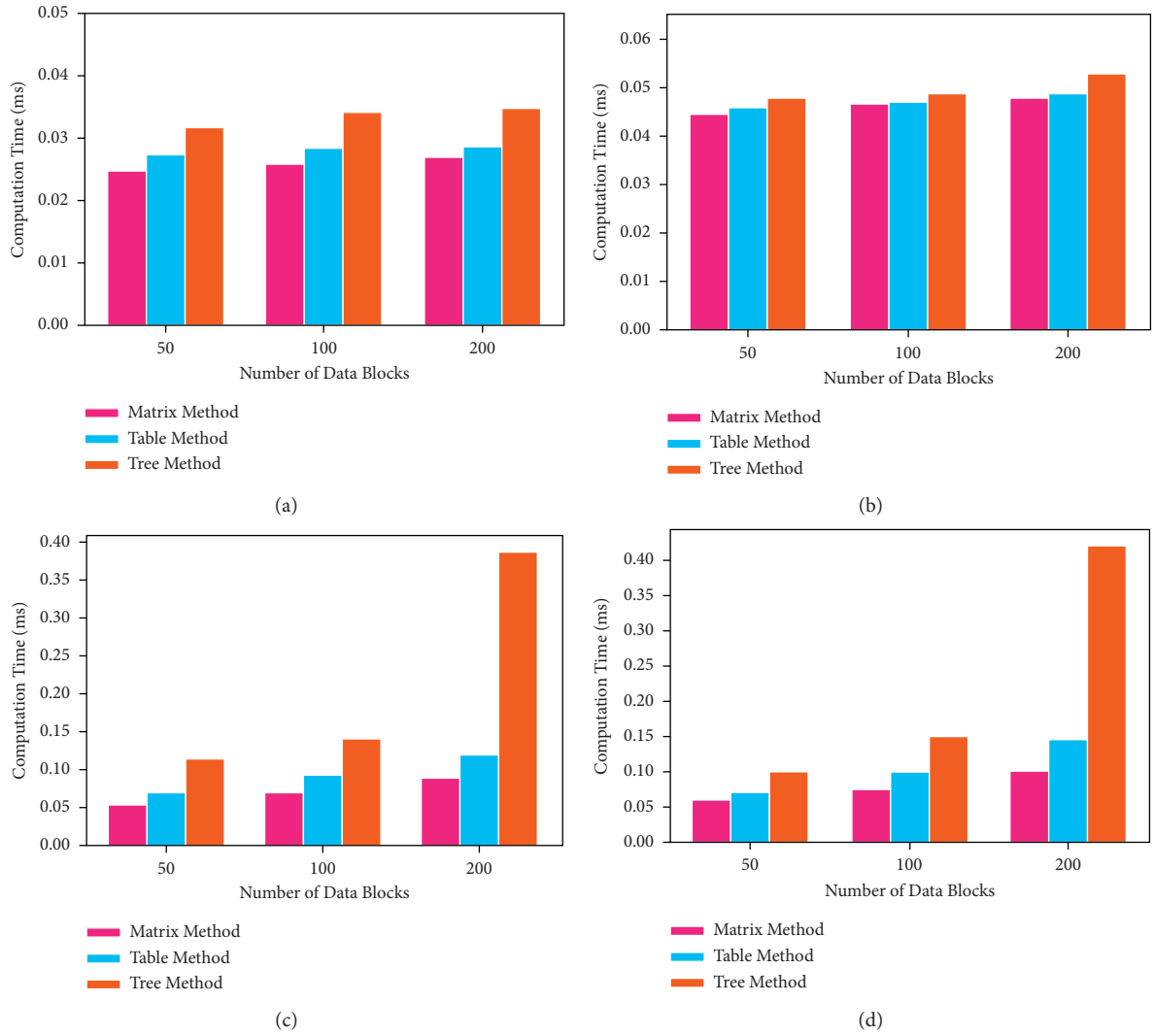

Figure 11: Comparison of computation time in insertion and deletion. (a) Block insertion. (b) Subblock insertion. (c) Block deletion. (d) Subblock deletion.

Further, we simulate the dynamic operation on the MATLAB platform and construct the index matrix M_I in MATLAB, where all the matrix values are 1, and the first column of the matrix is numbered 1 to n in MATLAB. Each operation is implemented three times according to different data blocks. The number of blocks is 50, 100, and 200. In order to reflect the rule of time overhead, we simulate three data block operations with different subblock numbers, namely, 50, 100, and 200.

Figure 11 shows the computing cost of data insertion and deletion in three dynamic operation methods. Figures 11(a) and 11(b) are the insertion operations of blocks and subblocks. The computing cost of the matrix-based method used in our dynamic update strategy is slightly lower than that of the table-based method, while that of the tree-based method is much higher than that of the matrix and table-based method. Figures 11(c) and 11(d) are block and subblock deletion operations. Therefore, the computational cost of dynamic operations based on matrix indexes is the lowest.

8. Conclusion

In this paper, data integrity verification in a mobile edge computing environment is studied. We propose a lightweight and dynamic data integrity verification method in an MEC environment. In order to achieve low latency and acceptable computational overhead, we design a data integrity verification protocol based on algebraic signatures. Through detailed performance analysis, the feasibility, security, and privacy of the proposed method are proved. At the same time, a data dynamic update optimization strategy is proposed to further reduce the computing cost. We have conducted a series of experiments to compare the computational overhead of the proposed method with other

methods at various stages. Simulation results show that the performance of our method is better than the baseline methods.

In future work, we will study how to ensure that the algebraic signatures-based integrity verification method is more secure and efficient, and we will also consider the problem of data integrity verification for multiple mobile users.

Data Availability

The data supporting the results of this study can be obtained from the corresponding author.

Disclosure

This work is extended from publication [5].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61772285), Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, and Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks.

References

[1] Y. Liu, Y. Li, Y. Niu, and D. Jin, "Joint optimization of path planning and resource allocation in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 19, no. 9, pp. 2129–2144, 2020.

[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: a survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[3] Y. Ping, Y. Zhan, K. Lu, and B. Wang, "Public data integrity verification scheme for secure cloud storage," *Information*, vol. 11, Article ID 409, 2020.

[4] D. Liu and J. Shen, "Dang with corrupted data recovery for edge computing in enterprise multimedia security," *Multimedia Tools and Applications*, vol. 79, pp. 10851–10870, 2020.

[5] H. Wang, J. Zhang, Y. Lin, and H. Huang, "ZSS signature based data integrity verification for mobile edge computing," in *Proceedings of the 2021 IEEE/ACM 21st international symposium on cluster, Cloud and Internet Computing (CCGrid)*, pp. 356–365, Melbourne, Australia, May 2021.

[6] H. Zhu, Y. Yuan, Y. Chen et al., "A secure and efficient data integrity verification scheme for cloud-IoT based on short signature," *IEEE Access*, vol. 7, pp. 90036–90044, 2019.

[7] Z. Fan, X. Lin, G. Tan, Y. Zhang, and W. Dong, "One secure data integrity verification scheme for cloud storage," *Future Generation Computer Systems*, vol. 96, pp. 376–385, 2019.

[8] Y. Deswarte, J. J. Quisquater, and A. Saidane, "Remote integrity checking," in *Proceedings of the 6th Working Conf. Integr. Internal Control Inf. Syst. (IICIS)*, pp. 1–11, Fairfax, Virginia; USA, November 2004.

[9] M. Venkatesh, M. R. Sumalatha, and C. Selva Kumar, "Improving public auditability, data possession in data storage security for cloud computing," in *Proceedings of the Int. Conf. Recent Trends Inf. Technol.*, pp. 463–467, Udiapur, India, April 2012.

[10] G. Ateniese, R. C. Burns, R. Curtmola et al., "Provable data possession at untrusted stores," in *Proceedings of the CCS*, pp. 598–610, Alexandria, VA, USA, October 2007.

[11] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the ASIACRYPT*, pp. 90–107, Melbourne, Australia, December 2008.

[12] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 847–859, 2011.

[13] I. Shen, D. Liu, D. He, X. Huang, and Y. Xiang, "Algebraic signatures-based data integrity auditing for efficient data dynamics in cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 5, no. 2, pp. 161–173, 2020.

[14] Y. Ren, J. Qi, Y. Liu, J. Wang, and G.-J. Kim, "Integrity verification mechanism of sensor data based on bilinear map accumulator," *ACM Transactions on Internet Technology*, vol. 21, Article ID 19, 2021.

[15] J. Z. Lu and H. X. Pan, "An integrity verification scheme of cloud storage for internet-of-things mobile terminal devices"," *Computers & Security*, vol. 92, Article ID 101686, 2020.

[16] H. Wang, D. He, J. Yu, N. N. Xiong, and B. Wu, "RDIC: a blockchain-based remote data integrity checking scheme for IoT in 5G networks," *Journal of Parallel and Distributed Computing*, vol. 152, pp. 1–10, 2021.

[17] D. Yue, Y. Li, Y. Zhang, W. Tian, and Y. Huang, "Blockchain-based verification framework for data integrity in edge-cloud storage," *Journal of Parallel and Distributed Computing*, vol. 146, pp. 1–14, 2020.

[18] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain based data integrity service framework for IoT data," in *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, pp. 468–475, Honolulu, HI, USA, June 2017.

[19] W. Tong, B. Jiang, F. Xu, Q. Li, and S. Zhong, "Privacy-preserving data integrity verification in mobile edge computing," in *Proceedings of the ICDCS*, pp. 1007–1018, Dallas, TX, USA, July 2019.

[20] Y. Zhang, R. Safavi-Naini, and W. Susilo, "An efficient signature scheme from bilinear pairings and its applications," in *Proceedings of the Int. Workshop Public Key Cryptogr*, pp. 277–290, Berlin, Germany, March 2004.

[21] R. Mokadem and W. Litwin, "String-matching and update through algebraic signatures in scalable distributed data structures," in *Proceedings of the International Workshop on Database and Expert Systems Applications*, pp. 708–711, Krakow, Poland, September 2006.

[22] S. S. J. Schwarz and E. L. Miller, "Store, forget, and check: using algebraic signatures to check remotely administered storage," in *Proceedings of the International Conference on Distributed Computing Systems*, pp. 12–21, Lisboa, Portugal, July 2006.

[23] Y. Luo, S. Fu, M. Xu, and D. Wang, "The Enable data dynamics for alge signatures based on remote data possession checking in cloud storage," *China Communications*, vol. 11, 2014.