Research Article

Public Integrity Auditing of Shared Encrypted Data within Cloud Storage Group

Chunxia Han and Linjie Wang

School of Data Science, Tongren University, Tongren, Guizhou 554300, China

Correspondence should be addressed to Linjie Wang; wanglinjie_66@hotmail.com

Received 1 November 2021; Revised 26 November 2021; Accepted 25 January 2022; Published 25 February 2022

Academic Editor: Yingjie Wang

Copyright © 2022 Chunxia Han and Linjie Wang. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The pandemic of COVID-19 has posed a severe challenge to the traditional on-site centralized development projects; people therefore have to share data in a group by the cloud storage server and develop projects at home. The cloud server is untrustworthy, although it supplies the powerful computing capability and abundant storage space; so far wide research has been proposed to verify data integrity. Therefore, how to leverage the cloud server and ensure the integrity of the data (especially the encrypted data) stored on the remote cloud devices remains an issue for the clients. To address this issue, we utilize the technique of homomorphic hash function to implement re-encryption ciphertext blocks and introduce a certificateless signature scheme for the integrity verification of encrypted data shared within a group. A detailed challenge-and-response game represents that the proposed scheme can preserve encrypted data blocks integrity against the internal/external attacker and malicious cloud service servers. We give the theoretical and experimental performance analysis of the scheme and exhibit that the scheme is efficient and practical.

1. Introduction

To date, cloud storage service has been an efficient paradigm for storing and sharing data information and a cooperation platform for staff to collaborate in many companies. Once a project manager can upload tasks to the server, each project participant can access, download, and modify the corresponding files through the network without any geographical restriction. Especially with the travel restrictions caused by the COVID-19 pandemic, people can only stay in their own homes and work through the Internet; thereby the use of cloud services for task cooperation and sharing has become particularly important. In the real world, Dropbox for Business [1], TortoiseSVN [2], and Google Drive already have become cloud service platforms for employees to share and collaborate online.

However, the prerequisite for this type of application to facilitate many company staff to work together is whether the cloud server provider (CSP) can make sure that the data is retained intact. In the field of cloud services, there is a multitude of inevitable internal and external attacks [3], as a slice of examples, the failure of software or hardware, illegal access, and deliberate deletion or corruption of the outsourced data, resulting in unreliable cloud services. Owing to the existence of these above attacks, the integrity of data is destroyed, which will inevitably reduce the availability and storage significance of data. This paper focuses on the integrity of data stored in the cloud.

For decades, to address the integrity verification of data, an army of studies on remote data integrity checking have been proposed by papers [4–16], and these schemes give efficient approaches to verify the integrity of outsourced data on cloud server without downloading them. However, all the above solutions are focused on the integrity auditing for individual data without involving the situation of sharing data in a group. How to verify the integrity of the shared data in a group is an interesting and essential task in the cloud server, which is also another item for the cloud service.

Remote data integrity verification is a technology that, for the data stored in the virtual cloud server, there is no
need to download the entire file locally to check the integrity of data. When a project with data attached to it is uploaded to a cloud server and shared among multiple engineers, some new challenges emerge and these challenges cannot be solved well with existing individual data integrity verification solutions. According to the above scenario, the project with data attached is divided into blocks and sent to the engineers of the project group, and different engineers will output different block tags in the same block. When a block is modified by the engineers of the project group, the new block tags will be regenerated. In a project group, all engineers will either online or offline compile their tasks, but no matter which kind they are required to store the results of the day’s tasks in the cloud server and generate block tags for checking. To ensure that each engineer is honest to compile the project with the given data, the project manager must act as a verifier to verify the integrity of the data from time to time. When the verifier wants to audit the integrity of raw data, it needs to aggregate all tags with the engineer’s identity information. The process of verification is more complicated and brings a significant volume of calculation [17–24]; these protocols thereby are not valid for the case of data sharing in a group.

When the data is shared among the engineers in the project group, also other challenges appear where some engineers in the group maybe withdraw from the group due to some special circumstances, such as being transferred to another project group or misbehaving. As a result of the above situations, the tags generated by the revoked user are invalid and need to be renewed by the other legitimate members. In addition, the data in the shared group also needs to be updated frequently, which also leads the tags to be changed constantly. For security reasons, if the identity of an engineer is revoked, all data as well as the corresponding tags, which belong to the revoked members previously, still have to be renewed by the existing user in the group. According to common sense, when an engineer exits the project, then its task will be transferred to other engineers, and its identity information in the project will be revoked; that is, its public/private key for participating in the project will become invalid. Considering the fact that the shared data is not stored on local devices, the traditional way is to download all data previously generated by the revoked engineer and ask an existing engineer to renew the tags and finally upload the new tags to the cloud server again. This operation can safely transfer the task to the engineer existing in the task, but it may significantly increase the existing engineer communication cost and calculation resources, especially when a considerable volume of the blocks needs to frequently change and update. To overcome the above drawbacks, the execution of the verification operations should be outsourced to CSP instead of execution by the existing engineers. Besides, integrity verification of shared data can be verified not only by the members of the shared data group but also by everyone who wants to leverage the data blocks in the cloud service. As a result, it is of tremendous significance that the scheme to be proposed can meet the public verification with the help of CSP.

At present, plenty of integrity verification schemes for shared data in this group have been put forward. Most of them [25–29] focus on the PKI technology based on the trustworthiness of certificate authority (CA), where it is difficult to find a trusted CA. Others are identity-based [28, 30] remote data integrity verification protocols, which rely on the private key generator (PKG) to generate all private keys. However, this approach suffers from a key escrow problem. Therefore, how to efficiently verify the integrity of outsourced data in a shared group by a public verifier and transfer the revoked members’ data to existing members without downloading the data from the cloud service, as well as solving the key escrow and certificate management issues, is a challenging task.

Reviewing the existing protocol solutions, we mainly focus on the integrity verification for the encrypted shared data in a group. In this paper, we assume that there is an encrypted business project, which is divided into numerous encrypted subprojects, and it needs plenty of engineers to participate in development. A project manager, who invites the engineers to a temporary project group, takes charge of the system parameters and encrypts the raw project. Then the project blocks encrypted with public keys of specified members in the group are uploaded to the cloud service so that the engineers within the group can modify and upload subprojects compiled online or offline. If this is a big project with plenty of engineers in the project group, there are some issues to be addressed efficiently, for example, the integrity verification after legitimate changes to subprojects under development, the members revocation problem, and the entry of new members.

1.1. Contributions. To overcome the disadvantages of previous schemes and address the aforementioned issues, we propose a new remote data possession checking scheme for encrypted shared data group. The contributions of the proposed scheme are presented as follows:

(i) We propose a new remote data possession checking scheme to audit encrypted shared data in each group, in which the certificateless public key system is utilized as an underlying encryption mechanism, and the homomorphism hash approach is used to regenerate the ciphertext to improve the efficiency of member revocation scheme.

(ii) We then construct a public auditing scheme for verifying the integrity of encrypted data in the cloud service provider based on the corresponding certificateless authentication tag aggregation.

(iii) We design a ciphertext conversion scheme which leverages a homomorphic hash function to convert the ciphertext of the revoked member into the ciphertext of the existing member. The scheme has been implemented and the results are more efficient compared to state-of-the-art protocols.

(iv) We have proven the security of the proposed scheme which is based on the stability of CDH and DL assumptions by simulating a challenge-and-
response game involving two players: a challenger and an adversary.

1.2. Organization. The rest of this paper is organized as follows. Section 2 discusses the prior work done in verifying the integrity of group shared data. Section 3 introduces the preliminaries and Section 4 defines the problem statement which includes system model, design goals, outline of the scheme, and the secure model. The detailed construction of our scheme is presented in Section 5. The proposed scheme is simulated and a challenge-and-response secure model is formalized in Section 6, and we assess the efficiency of the proposed scheme in Section 7 based on the computation cost of tag generation and verification, communication cost analysis, and vocation analysis compared with the existing schemes. The conclusion of this paper is presented in Section 8.

2. Related Work

Since Deswarte et al. [4] first proposed a scheme for checking the integrity of data stored on remote virtual cloud servers; so far, a number of auditing schemes have been proposed. Among the proposed schemes, they can be generally divided into two directions: Provable Data Possession (PDP) [5] and Proofs of Retrievability (POR) [23].

The PDP scheme is proposed by Ateniese et al. [5] based on RSA signature and sampling strategies to improve the efficiency of integrity checking without retrieving the whole file. However, there is a limitation for this scheme; that is, it is only suitable for the auditing of static data files. To overcome this limitation, Atieniese et al. [31] presented the revised version of PDP based on symmetric encryption to efficiently address the dynamic checking issues instead of handling the insertion operation. Erway et al. [18] gave a dynamic provable data possession (DPDP) scheme, which supports full data dynamic operations to solve the insertion operation and improve the verification efficiency by leveraging the authenticated skip list.

To support fully dynamic data, Wang et al. [32], Erway et al. [18], and Zhu et al. [33] successively proposed schemes to construct auditing mechanisms supporting fully dynamic data, respectively. To realize public verification and dynamic data operation, Liu et al. [34] gave a dynamic public auditing scheme based on the Merkle Hash Tree (MHT), in which the block tags are generated by the data owners, and this incurs the increase of communication and calculation cost. To overcome this drawback, a scheme [35] to solve the heavy calculation burden on the data owner side at the expense of data owner’s privacy has been proposed, in which both tag generation and integrity verification are implemented by the cloud server. The issue of privacy-preserving in public auditing has been addressed, in which the data blocks are blinded by a data owner before generating signatures by the third party [36]. In another related research, to avoid the certificate management problem of PKI, some PDP schemes based on Identity-Based Signature (IBS) [37, 38] were proposed. The major problem of IBS is the key escrow, which is solved by a certificateless-based signatures PDP scheme [39].

Similar to PDP, the POR is another approach introduced by Juels et al. [23] to audit the integrity of remote data stored on the cloud service. An improved POR scheme is given by Wang et al. [10] to authenticate block tags, in which a security proof is revised in their previous work. Based on the previous works of Erway et al. [18] and Atieniese et al. [5], a generic framework DPOR is proposed by Etemad et al. [40] to store call updated information in the logs. Apart from the aforementioned protocols, some other publicly verifiable protocols are published. Hao et al. [41] gave a public verification without including a TPA, and Shen et al. [42] solved the loss of private key for auditing issue. Wu et al. [43] introduced a time encapsulated POR protocol that could check the integrity of data and timestamp by verifier.

All schemes mentioned above mainly devote themselves to verifying the integrity of individual data. Since Wang et al. [44] proposed a scheme for auditing the integrity of data shared in a group in 2012, a succession of verification schemes for sharing data in a group have been proposed [7, 26–29, 45]. Among these schemes, [26, 27, 29, 45] represented PDP schemes for group data based on the signatures, respectively, and all of these schemes are more or less deficient in efficiency and revocation. To solve the multiuser modification problem of blocks, [28] based on PKI mechanism proposed a PDP scheme of polynomial authentication tags, which led to a heavy burden of certificate management. Recently, Li et al. [7] based on certificateless mechanism proposed a public integrity checking of group shared data on cloud storage, which changes the data tags of the revoked member into the existing member’s tags. However, the data within the group in the scheme is all plaintext, which cannot satisfy the situation that the shared data in the group is ciphertext. According to all references mentioned above, although there are numerous schemes that can solve the problems of user adding and revocation in a shared group, on the premise of integrity auditing, there is no verification research on the integrity of encrypted data in a shared group. Therefore, we devote to designing a scheme for the integrity verification of encrypted data group in cloud service, which not only satisfies the member addition and revocation but also decreases the computational burden of challenge proof on the client side with the help of CSP.

3. Preliminaries

3.1. Bilinear Maps. Let \(G_1\) and \(G_2\) be two multiplicative cyclic groups of prime order \(p\), and let \(g\) be a generator of \(G_1\). A bilinear map \(e: G_1 \times G_1 \rightarrow G_2\) has the following properties:

(1) Computability: there exists an efficient algorithm to compute map \(e\).

(2) Bilinearity: for all \(u, v \in G_1\) and \(a, b \in \mathbb{Z}_p\),

\[
e(u^a, v^b) = e(u, v)^{ab}
\]

(3) Nondegeneracy: there exists a point \(g\) such that \(e(g, g) \neq 1\).
3.2. Complexity Assumptions. In our scheme, the security is based on the following security assumptions.

**Definition 1** (Computational Diffie-Hellman (CHE) Problem). Let \(a, b \in \mathbb{Z}_p^*\); given the tuple \((g, g^a, g^b)\in \mathbb{G}_1\) as input, output \(g^{ab} \in \mathbb{G}_1\).

**Assumption 2** (Computational Diffie-Hellman (CHE)). For any probabilistic polynomial time (PPT) algorithm \(\mathcal{A}\), \(\Pr[\mathcal{A}(g, g^a, g^b) \rightarrow g^{ab}]\) is negligible, where \(g, g^a, g^b \in \mathbb{G}_1\).

**Definition 3** (Discrete Logarithm (DL) Problem). Let \(g\) be a generator of \(\mathbb{G}_1\); given the tuple \((g, g^a)\) as input, output \(a \in \mathbb{Z}_p^*\).

**Assumption 4** (Discrete Logarithm (DL)). For any probabilistic polynomial time (PPT) algorithm \(\mathcal{A}\), \(\Pr[\mathcal{A}(g, g^a) \rightarrow (a)]\) is negligible, where \(a \in \mathbb{Z}_p^*\).

3.3. Homomorphic Hash Function. For a finite field \(F_n\) and a multiplicative group \(\mathbb{G}\) of order \(p\), a family of homomorphic hash functions are a collection \(\mathcal{H} = \{h_i : F_n \rightarrow \mathbb{Z}_q\}\), where \(i\) is the index yielded by an efficient algorithm. A homomorphic hash function [46] consists of the following properties:

1. One way: given \(x \in F^n\) and an index \(i\), there is no polynomial time adversary which can find a \(h_i^{-1}(x)\).
2. Collision resistance: given an index \(i\), it is hard (computationally infeasible) to find two vectors \(x, y \in F^n\) such that \(h_i(x) = h_i(y)\).
3. Homomorphism: given an index \(i\) and any \(x, y \in F^n\), \(x \neq y\), \(h_i(x \ast y) = h_i(x) \ast h_i(y)\), where \(\ast\) is either a "\(\cdot\)" or a "\(\ast\)."

4. Problem Statement

In this section, we show the system model and secure model and illustrate the design goals and the outline of our proposed scheme.

4.1. System Model. Similar to [7, 27, 29], we combine the cloud architecture with an example of sharing and developing encrypted files by the staffs of a company that are in the same group or department. The system model consists of three major entities: project group (i.e., members involved in the project), cloud service provider (CSP), and public verifier, and the relationship and the interaction situation among them are represented in Figure 1.

Project group consists of a volume of project members and a project manager that rents the cloud service platform. In the given example, a project manager is the original owner of the project file and takes charge of dividing the file into encrypted blocks, system parameters generation, member joining/revocation, and sharing the blocks in the project group through a cloud service provider. All project members can access, download, and modify the specified, encrypted data blocks.

Cloud service provider offers a wealth of storage services and powerful computing abilities by charging a certain fee. Referring to the research in [7], CSP can honestly implement the scheme but may try to gain the content of stored files and return an incorrect result to the verifier to get some extra benefits. Therefore, we assume that the CSP is semitrusted, encrypting all file blocks stored in the CSP, and generate tags corresponding to the project members.

The verifier can be any member of the project group that checks the integrity of encrypted data blocks kept in the CSP. Once a verifier sends an integrity auditing request, the CSP generates and returns the verification information. The verifier then checks the correctness of the auditing proof and reports the verification result.

4.2. Design Goals. To efficiently and securely verify shared encrypted data with a volume of members in a project group, our proposed scheme should be designed to achieve the following properties:

1. Correctness: Based on the challenged proof generation, the verifier is able to correctly detect the integrity of challenging blocks.
2. Unforgeability: Only the specified member in the project group can yield valid verification information on the encrypted data blocks.
3. Identity privacy: During the integrity of auditing, the CSP cannot distinguish the identity of tag generator on each randomly picked block in the shared project group.
4. Tag-updating: When the identity of some members in the project group is revoked or new members are added, the corresponding ciphertext tags should be updated efficiently and securely.
5. Verifiability: Random verifier is able to verify the integrity of ciphertext attached tags by the challenged proof calculated by the CSP.

4.3. Outline of the Scheme. The scheme consists of eight steps:

1. **Setup** (\(\mathcal{S}:\)): Taking a security parameter \(\kappa\) as input, the project manager implements this step and outputs the master key \(msk\) and all system parameters \(params\).
2. **PartialKeyGen** (\(ID_i, params, msk\) --- \(D_i\)): Taking the member’s identity \(ID_i\), the master key \(msk\), and the parameters \(params\) as input, the project manager executes this step and outputs member \(u_i\)’s partial key \(D_i\).
3. **KeyGen** (\(D_i, params\) --- \(ssk_i, spk_i\)): Taking the member’s partial key \(D_i\) and the parameters \(params\) as input, the project member runs this step and returns pairing private/public key \((ssk_i, spk_i)\).
4. **Encrypt** (\(m_i, spk_i\) --- \(\sigma_i\)): Taking the file blocks \(m_i\) and member’s public key \(spk_i\) as input, the project
manager executes this step and generates a ciphertext \( \sigma_i \).

(5) \( TagGen(\sigma_i, ssk_i) \rightarrow (T_i) \): Taking the ciphertext \( \sigma_i \) and a private \( ssk_i \) as input, the project member leverages this step to generate a tag \( T_i \) attached to the ciphertext block \( \sigma_i \) when uploading to CSP.

(6) \( Challenge(c) \rightarrow (chal, F_{i,d}) \): Taking the count of challenged block \( c \) as input, the verifier runs this step to output a challenge information \( chal \) appending the block name \( F_{i,d} \) for the integrity querying of the data file.

(7) \( ProofGen(chal, F^*, [T_i | i \in \mathbb{N}]) \rightarrow P \): Taking the challenge information \( chal \), the challenged encrypted block set \( F^* \), and tag set \( [T_i | i \in \mathbb{N}] \) as input, CSP runs this step and responses with the integrity proof \( P \).

(8) \( VerifyProof(P, chal, params) \rightarrow 0, 1 \): Taking the integrity proof \( P \), the challenge information \( chal \), and the public parameters \( params \) as input, the verifier implements this step and returns 1 if result \( P \) passes the verification; otherwise, it returns 0.

Note that, in addition to the above steps, there are two other steps: \( JoinGen \) and \( RevGen \). Step \( JoinGen \) is executed by the project members, which invites some other members who are not in the project group, and step \( RevGen \) is also implemented by the project members, and the procedure is divided into two scenarios depending on whether the revoked member has invited members to participate in the project group.

4.4 Secure Model. Since the certificateless cryptography \([47]\) is the underlay of our new scheme, and referring to the security model of data integrity auditing protocols represented in papers \([30, 39, 48]\), we consider the security requirement and adversary model of the encrypted shared scheme against a fully-adaptive chosen ciphertext attacker (IND-CCA) \([47, 49]\) which involves a challenger \( C \) and four types of adversaries, namely \( \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \) and \( \mathcal{A}_4 \). Among the four adversary types, although adversaries \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) both execute the tag-forge attacks, they have different attack capabilities. Type \( \mathcal{A}_3 \) implements the ciphertext integrity proof attack to cheat the verifier, and type \( \mathcal{A}_4 \) tries to generate a forgery of regenerated ciphertext and passes the verification of ciphertext receiver. We give the four following games to illustrate the security model in detail.

4.4.1. Setup. There are two parties, adversary \( \mathcal{A}_j \) for \( \mathcal{A}_j \) and challenger \( C \) that keeps the private keys and the master key security and sends the public system parameters to \( \mathcal{A}_j \). Challenger \( C \) interacts with adversary \( \mathcal{A}_j \) for \( j \in \{1, 2\} \) in this game. In order to generate a forgery of tag in a security game, \( \mathcal{A}_j \) needs to execute the following different queries: hash query, key query, public key query, public key replacement, encryption query, and tag query.

4.4.2. Common Queries. \( \mathcal{A}_j \) gives the polynomial times different queries to \( C \) which responds to the following queries:

(1) Hash query is adaptively made by \( \mathcal{A}_j \) and the hash values are responded by \( C \).

(2) Key query is adaptively run by \( \mathcal{A}_j \) to submit different target identity \( ID \) (first running the step PartialKeyGen by \( C \) for \( ID \) if necessary) to \( C \) for
4.4.3. Game1. In this game, adversary $\mathcal{A}_1$ not only executes the Common Queries above but also runs the following specialized queries:

1. Partial key query: is adaptively implemented by $\mathcal{A}_1$ to submit different target identity ID to $\mathcal{C}$, and then the partial key for the ID is responded to $\mathcal{A}_1$ by $\mathcal{C}$ running the step PartialKeyGen.

2. Public key replacement: According to the assumed capability of $\mathcal{A}_1$, it can replace the public key $ssk_{ID}$ of any ID with random value $ssk_{ID}'$ multiple times if necessary.

4.4.4. Forgery. Eventually, there are two scenarios on the forgery tag $Tag'$ output by adversary $\mathcal{A}_1$. One is that $\mathcal{A}_1$ outputs a forgery tag $Tag'$ for the ciphertext block $\sigma$ encrypted by the public key $ssk_{ID}'$ and the tag is generated with the public key $ssk_{ID}'$ and the identity $ID'$. The other one is that $\mathcal{A}_1$ outputs a forgery tag $Tag'$ for ciphertext block $\sigma'$ by public key $ssk_{ID'}$, and the tag is generated with the public key $ssk_{ID'}$ and the identity $ID'$. In both scenarios, such adversary $\mathcal{A}_1$ does not have access to the master key of system, but it can request the public key and has the capability to replace the member's public key and make the tag queries for all identities of its random choice. However, if $\mathcal{A}_1$ wants to win the game, there are several natural restrictions on adversary $\mathcal{A}_1$ as discussed below:

1. $\mathcal{A}_1$ cannot request a query on the private key for identity $ID'$ at any time.

2. $\mathcal{A}_1$ cannot both query the partial key for $ID'$ and substitute the public key of identity $ID'$ at the same time.

3. $\mathcal{A}_1$ cannot make a tag query for the encrypted target data block $\sigma'$ with the identity $ID'$ and the public key $ssk_{ID'}$.

4. In addition to the above limitations, $\mathcal{A}_1$ can forge a valid tag for the encrypted data block $\sigma'$ with the identity $ID'$ and the public key $ssk_{ID'}$, and it also can forge a valid tag for the ciphertext block $\sigma$ encrypted with the legitimate public key $ssk_{ID}$, where the tag is generated by the replaced public key $ssk_{ID'}$ and the identity $ID'$.

4.4.5. Game2. In this game, adversary $\mathcal{A}_2$ only executes the Common Queries above and then forges the tag $Tag'$ for the ciphertext $\sigma'$ with the identity $ID'$.

4.4.6. Forgery. In this process of forgery, adversary $\mathcal{A}_2$ is unable to replace the member's public key, but it has the capability to access the master key of system. However, there are also two scenarios on the forgery tag $Tag'$ output by adversary $\mathcal{A}_2$. One is that $\mathcal{A}_2$ outputs a forgery tag $Tag'$ for ciphertext block $\sigma$ encrypted by the public key $ssk_{ID'}$, and the tag is generated with the public key $ssk_{ID'}$ and the identity $ID'$. The other one is that $\mathcal{A}_2$ outputs a forgery tag $Tag'$ for ciphertext block $\sigma'$ by public key $ssk_{ID'}$, and the tag is generated with the public key $ssk_{ID'}$ and the identity $ID'$. In addition, if $\mathcal{A}_2$ wants to win the game, it is subject to the following restrictions:

1. $\mathcal{A}_2$ can neither query the private key nor replace the public key for $ID'$ at any point.

2. $\mathcal{A}_2$ cannot make a tag query for the encrypted target data block $\sigma'$ with the identity $ID'$.

3. In addition to the above limitations, $\mathcal{A}_2$ can forge a valid tag $Tag'$ for the encrypted data block $\sigma'$ with the identity $ID'$, as well as the legitimate public key $ssk_{ID'}$.

Definition 5. The scheme is semantically secure against the single tag forged attack of the ciphertext block if adversary $\mathcal{A}_1$, or $\mathcal{A}_2$ in polynomial probability time has a negligible advantage to win Game 1 and Game 2.

4.4.7. Game3. In terms of Definition 5, in an adversary cannot forge a legitimate label for a single ciphertext block without accessing the right private key. In this game, we consider that adversary $\mathcal{A}_3$ that acts as the untrusted CSP in the system attempts to persuade the verifier to pass the integrity verification of corrupted data. Inspired by [7], challenger $\mathcal{C}$ plays two roles, that is, the honest CSP and an integrity checker, and the operation of Game 3 is executed as follows:

Tag query: The target tuple $(ID, m)$ is adaptively selected by $\mathcal{A}_3$, and sent to $\mathcal{C}$, which responds with the querying tag which is generated with the ciphertext $\sigma$ and the identity $ID$ by the step TagGen.

Challenge: Challenger $\mathcal{C}$, which acts as the verifier, generates and sends a random challenge information $chal$ to $\mathcal{A}_3$, which is requested to respond with the corresponding data possession proof $P$ for $chal$.

Forgery: Once receiving the challenge information $chal$, $\mathcal{A}_3$ acts as the CSP, generates a proof $P$, and sends it to $\mathcal{C}$.

The premise for $\mathcal{A}_3$ to win the game is that the miscalculated block information in proof $P$ can pass the integrity verification successfully.

Definition 6. The scheme is semantically secure against forging the integrity proof on incorrect data if adversary $\mathcal{A}_3$ in polynomial probability time has a negligible advantage to win Game 3.

4.4.8. Game4. In this game, the specified member acts as adversary $\mathcal{A}_4$ that interacts with challenger $\mathcal{C}$. Here, the revoked member and CSP are regarded as the trusted parties.
If the reencrypted data has been corrupted, \( \mathcal{A}_4 \) tries to cheat the verifier that the tag generated by reencrypted data can pass the integrity verification. In terms of Definitions 5 and 6, we know that any adversary cannot pass the tag verification on a single block without accessing the private key and correct data. Therefore, the focus of this game is on whether adversary \( \mathcal{A}_4 \) can forge the integrity proof of reencrypted data to pass the verification. Inspired by [7, 29], challenger \( \mathcal{C} \) plays two roles, that is, the honest CSP and a revoked member, and the operation of Game 4 is executed as follows:

Reencrypt key query: \( \mathcal{A}_4 \) adaptively picks an identity \( ID \) and submits it to challenger \( \mathcal{C} \) for querying the reencrypting key of \( ID \). \( \mathcal{C} \) runs the reencrypting key subroutine in step \( \text{RevGen} \) and returns the reencrypting key \( r_{ID} \). Tag query: The target tuple \( (\sigma, ID) \) is adaptively selected by \( \mathcal{A}_4 \) and sent to \( \mathcal{C} \) for querying the tag for the reencrypting ciphertext \( \sigma' \). According to the step \( \text{RevGen} \), \( \mathcal{C} \) responds the tag generated by ciphertext \( \sigma' \) and \( ID \) to adversary \( \mathcal{A}_4 \).

4.4.9. Forgery. Eventually, \( \mathcal{A}_4 \) outputs a forgery tag \( \operatorname{Tag}' \) for the target ciphertext \( \sigma \) with the identity \( ID \).

Definition 7. The scheme is semantically secure against forging the integrity proof without both correct identity and reencryption key if adversary \( \mathcal{A}_4 \) in polynomial probability time has a negligible advantage to win the aforementioned Game 4.

5. Our Scheme

Without loss of generality, there is a project manager named \( u_0 \) in the group which is in charge of the generation of system parameters and other engineers’ partial secret keys. Suppose that the project F is divided into n project blocks as the following \( M = (m_1, m_2, \ldots, m_n) \). \( u_0 \) invites \( z \) engineers in one group to execute this project, and each engineer \( u_i \) has a unique identity represented as \( ID_i \) for \( 1 \leq i \leq z \). In order to keep the security of each block, all project blocks stored on the CSP should be encrypted by the public key of the corresponding developing engineer; namely, \( C^* = (\sigma_1, \sigma_2, \ldots, \sigma_z) \), in which \( \sigma_i \) represents the ciphertext of \( i \)th subproject \( m_i \). The scheme consists of the following steps:

\[
e\left(\prod_{k=1}^{l} T_{i_k}, g\right) = e\left((x_{i_1}^l \cdot \prod_{k=1}^{l} (H_2(\omega_{i_k})), x_{i_1}) \cdot e\left(H_1(ID_{i_k}) \sum_{k=1}^{l} a_{i_k}, P_0\right)\right),
\]

where \( a_{i_k}^l = g^c \).

**Challenge:** Anyone as a verifier can check the integrity of group data stored in CSP. The verifier randomly picks the challenged block count \( c(1 \leq c \leq n) \) and two values \( k_1, k_2 \in \mathbb{Z}_r^* \). Then the challenged information \( \operatorname{chal} = (c, k_1, k_2) \) appending the file name \( F_{i,d} \) is sent to CSP.

**Proof Gen:** Once the CSP receives \( \operatorname{chal} = (c, k_1, k_2) \), the challenge information set \( I = \{ (v_i, a_i) \} \) is calculated, in which

\[
a_i = \pi(k_i, i) \text{ is the regenerated parameter by the pseudorandom generator} \pi, \text{ and the subset } \{v_i | i \in (1, c)\} \text{ for } v_i = \phi(k_i, i) \text{ of } \{1, 2, \ldots, n\} \text{ is a new index permutation of challenge block generated by the pseudorandom generator} \phi. \text{ Without loss of generality, suppose that the challenge block set consists of encrypted blocks } \sigma_1, \sigma_2, \ldots, \sigma_c \text{ and let } C_n \text{ denote } \{\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_l}\}. \text{ Let the challenge block subsets } C_{i_1} = \{\sigma_{i_1}, \ldots, \sigma_{i_1}\}, \ldots, C_{i_l} = \{\sigma_{i_l}, \ldots, \sigma_{i_l}\}.
\{\sigma_{v_{e_1}}, \ldots, \sigma_{v_e}\} \text{ belong to engineers } u_{i_1}, \ldots, u_{i_j}, \text{ respectively, where the permutation } \{v_{\ell_1}, \ldots, v_{\ell_k}\} \text{ is the rearrangement of permutation } \{v_1, \ldots, v_c\}. \text{ We can obtain } C = C_{i_1} \cup C_{i_2} \cup \cdots \cup C_{i_j} \text{ and } C_{i_k} \cap C_{i_{k'}} = \emptyset \text{ for } k \neq k'. \text{ The set } \{I_1, I_2, \ldots, I'_c\} \text{ is the subset of permutation } \{1, 2, \ldots, z\} \text{ and } |C_{i_1}| + |C_{i_2}| + \cdots + |C_{i_j}| = c. \text{ The CSP calculates two sets } T = \{T_1, \ldots, T_x\} \text{ and } F = \{F_1, \ldots, F'_z\}, \text{ where } T_k = \prod_{v_i \in C_k} T_{v_i} a_i \text{ and } F_k = \sum_{v_i \in C_{i_k}} a_j. \text{ Finally, the proof } P = (T, F) \text{ is sent to the verifer.}

**VerifyProof:** Upon receiving proof } P, \text{ the verifer utilizes the precalculated values set } \{\{v_{\ell}, a_{j}\}\} \text{ to generate a set of challenge blocks. According to the tag generation rules, the verifer obtains } \omega_{v_{\ell}} \text{ to generate all the tags } T_{\omega_{v_{\ell}}} \text{ of participating challenge blocks. Then it takes all above proof information as input to check whether } (2) \text{ holds, where } I'_k = |C_{i_k}|:

\[
e^\left(\prod_{k=1}^{z'} T_k, g\right) = e^\left(\prod_{k=1}^{z'} T_k, g\right) = \prod_{k=1}^{z'} e^\left(\prod_{v_i \in C_k} T_{v_i} a_i, g\right) = e^\left(\prod_{v_i \in C_k} H_2(\omega_{v_i}) \cdot (\sigma_{v_i})^{a_i}, X_{a_i} \right) \quad \text{and} \quad e^\left(\prod_{k=1}^{z'} H_1(ID_k)T_k, P_0\right).
\]

If this equation holds, it outputs either 1 ("accept") or 0 ("reject"). The correctness of this scheme can be checked by the following equality:

\[
e^\left(\prod_{k=1}^{z'} T_k, g\right) = \prod_{k=1}^{z'} e^\left(\prod_{v_i \in C_k} H_2(\omega_{v_i}) \cdot (\sigma_{v_i})^{a_i}, X_{a_i} \right) \quad \text{and} \quad e^\left(\prod_{k=1}^{z'} H_1(ID_k)T_k, P_0\right).
\]

5.1. *Invite to Join.* If engineer \( u_j \) invites another engineer \( u_i \) to participate in its subproject, \( u_i \) first sends an identity concatenation \( ID_i \| ID_j \) to \( u_j \), and then \( u_j \) responds a partial private key \( D_{ji} = H_1(ID_i) \) to \( u_i \) by secure channel. \( u_j \) randomly chooses \( x_{ji} \in Z_q^* \) to generate its secure key \( ssK_{ji} = \langle D_{ji}, x_{ji} \rangle \) and public key \( spk_{ji} = \langle X_{ji}, Y_{ji} \rangle \), where \( X_{ji} = g^{x_{ji}} \) and \( Y_{ji} = P_{ji}^{x_{ji}} \).

While \( u_j \) successfully joins the project and wants to edit its block (named \( m_{ji} \)), the specified file ciphertext block \( m_{ji}^{r_{ji}} \) needs to be converted to a block encrypted by \( u_i \’s \) public key. The reencryption key \( r_{ji} \) is generated and the ciphertext \( m_{ji}^{r_{ji}} \) turns into \( m_{ji}^{r_{ji}} \) which is encrypted by \( u_i \’s \) private key and parameter \( r_{ji} \) randomly picked by \( u_i \). Note that the reencryption key \( r_{ji} \) is bidirectional; that is, it can be utilized to transfer the ciphertext from \( u_j \) to \( u_i \) and vice versa.

**JoinGen(ID_i, ID_j) \rightarrow (r_{ji}):** When receiving the identity \( ID_j \), \( u_i \) calculates the reencryption key \( r_{ji} = H_2(e(H_1(ID_i), Y_{ji})^r \oplus e(H_1(ID_i)\| ID_j), Y_{ji})^r \) and sends it to \( u_i \). Then \( u_j \) calculates ciphertext \( m_{ji}^{r_{ji}} \) for block \( m_{ji} \) as \( m_{ji}^{r_{ji}} = m_{ji}^{r_{ji}} \oplus H_2(e(H_1(ID_i), Y_{ji})^r \oplus e(H_1(ID_i)\| ID_j), Y_{ji})^r \) and \( m_{ji}^{r_{ji}} \oplus H_2(e(H_1(ID_i), Y_{ji})^r \oplus e(H_1(ID_i)\| ID_j), Y_{ji})^r \) and \( m_{ji}^{r_{ji}} \oplus H_2(e(H_1(ID_i)\| ID_j), Y_{ji})^r \).

5.2. *Revoke a Participant.* Once an engineer \( u_i \) leaves the project, the project manager \( u_0 \) should claim the private/public key pairings of \( u_i \) to be invalid. At the same time, the contents consisting of the ciphertext, tags, and so forth of the file block associated with the revoked engineer \( u_i \) are also changed. Otherwise, there are some secure risks on the ciphertext and tags which are executed by \( u_i \); thereby the integrity of the ciphertext cannot be checked either. In this process, there are two situations to be considered: one is that the revoked user \( u_i \) has invited engineers in the project, and,
in this case, any inviter (named u,.) can be required to replace the tag and ciphertext of the revoked user. In the other case, u, does not invite any users to participate in the project, so the project manager needs to convert the ciphertext and tags for u,. We represent the detailed implementation as follows:

RevGen: Assume that u, is the revoked project member, and u, is the member who continues the project in place of u,. In this section, CSP is used to check the correctness of regenerated tags by u,. In addition, suppose that u, u,, and CSP are all online simultaneously during this procedure.

Without loss of generality, let member u, as a specified recipient take charge of all blocks of the revoked engineeru,, utilizes the step JoinGen to regenerate the ciphertext of u,, and the block tag is yielded by u,.

1. u, calculates r, and sends it to u,, where r, = H2(e(Hi(ID), Y) · e(H1(ID), Y)),

2. Leveraging the key r,, u, calculates σ, = r,, and publishes the regenerated ciphertext σ, = (σ,1, σ,2) of block m,i for 1 ≤ k ≤ 1.

3. Then u, calculates the tag T, = H2((ω,1 · σ,2 · Y)), D, to CSP for ω, to ensure the validity of the tags.

6. Security Proof

In this section, we give the secure proof of the proposed scheme via the following properties.

6.1. Security Analysis

Theorem 1. If a polynomial probability time adversary A,, has an advantage to win Game 1 described in Section 4.4 within time t after executing the most q1i, Hash-1 queries, q2i, key queries, q3i, Public Key Replace, q4i encryption queries, and q5i, Hash-2 queries and requesting at most q6i times tag queries, then there exists an (ε, t)-simulator B that can address the CDH problem with (ε ≥ ε′(1 + q6i · q2i) · e) t ≤ 1 + (q1i + q2i + q4i + q5i + q6i + q2i · e) + 3m1 · q2i + q6i + q5i, where one exponentiation costs time t, on G1, one scalar multiplication operation costs time t, on G1, and e is the base of natural logarithm.

Proof. On input (g, g′, g′) in G1, if adversary A,, is able to forge a tag with the identity ID and the replaced public key in Game 1, then algorithm B has capability to address CDH problem; that is, it can calculate g′. Given g, g′, and g′, simulator B simulates each step of interaction with A, as follows:

Setup. A,, launches a query-respond game. B sets P0 = g′ with the master key a which is security picked and then outputs and returns the system parameters params = (q, g, G1, G1, e, P0, H1, H2, ϕ, π) to A,.

Hash-1 Query. A, adaptively requests the Hash-1 query results for any identity ID in terms of its capability. In order to facilitate the management of all the query results, B establishes a tuple list L1 = {(ID, r, Q, τ)} to record all query data. If a certain ID* has been recorded in the list, B directly returns its corresponding tuple (ID*, r, Q,τ) to A,. Otherwise, B selects a random value r ∈ Zq* and tosses a coin τ ∈ {0, 1}. Assume that the coin represents 1 with a probability of γ, and vice versa, 1 − γ. If τ shows 0, B sets Q* = H1(ID*) = g′ ∈ G1; if τ shows 1, B sets Q* = H1(ID*) = (g′)τ ∈ G1. Then the result Q* is returned to A, and the tuple (ID*, Q*, r, τ) is inserted to list L1, where the symbol * indicates that the position is empty and has no value, which may be generated in a subsequent query.

Partial key query. In order to obtain the partial key of any identity ID*, A,, adaptively implements partial key query. B firstly checks whether ID* corresponding tuple (ID*, r, Q, τ) exists in L1. If not, B executes the Hash-1 query and inserts the result in L1. Notably, another new tuple list L2 is established by B to manage the newly queried data during this process, where L2 = {(ID, D, D, spkID, sskID1, sskID2)} if τ shows 1 in L1, B returns for ID* and then records the tuple value (ID*, r, Q, r, τ) in L2. Otherwise, B responds the partial key query as follows.

1. If ID* is stored in list L2, B checks whether the location of D,ID* is a symbol1 or not. If it is not 1, B returns it directly to A,. Otherwise, B reexecutes the coin tossing step in Hash-1 query. When the coin tosses τ = 0, B returns the value D,ID* = (Q*)τ = g′τ to A, and then updates the values Q*, τ in L1, andD,ID* in L2 on the corresponding identity ID*, respectively; otherwise, τ = 1, and B aborts.

2. If ID* is not stored in list L2, B determines the value of D,ID* according to τ in list L1. If τ = 0, B returns D,ID* = g′τ to A,; otherwise, τ = 1, and B aborts.

Note that the tuple in L1 and L2 has such a characteristic: the value of τ in the tuple (I D, D, Q, r, τ) of L1 corresponding to the tuple (I D, D, Q, r, τ) of L2 is 1; the value of τ in the tuple (I D, D, Q, r, τ) of L1 corresponding to the tuple (I D, D,ID*, spkID, sskID1, sskID2) in L2 is 0.

Key query. A,, adaptively requests the key query for any identity ID*. B searches list L2 for the tuple (I D, D,ID*, spkID, sskID1, sskID2, sskID2). Then the tuple (I D, D,ID*, spkID, sskID1, sskID2, sskID2) is stored in L2, B first checks whether the position of D,ID* is 1 or not. IfID,ID* is 1, B turns to execute Hash-1 query and partial key query in turn. Otherwise, B checks whether the position of spkID in this tuple is the symbol *. If spkID = *, B selects xID ∈ Zq* at random and sets sskID = ⟨D,ID*, xID⟩ and spkID = ⟨XID, YID⟩ = (g′τ, P0τ). B updates the tuple (I D, D,ID*, r, Q, τ) into L2 and sends the key pairing ⟨spkID, sskID1, sskID2⟩ to A,.

(2) If the tuple (I D, D,ID*, spkID, sskID1, sskID2) is not stored in L2, B turns to execute Hash-1 query and partial key query in turn. Once the value ofD,ID* has been obtained after the Hash-1 query and partial key query, B randomly selects xID ∈ Zq* and sets
**Public Key Replace.** According to the assumption, adversary $\mathcal{A}_1$ adaptively implements the Public Key Replace for the target member $ID^r$ with the substitution public key $sp_{kid}$.}

(1) If the tuple $(ID^*, D_{ID^*}, sp_{kid}^r, ssk_{ID^*}, \ast, \ast)$ is stored in $L_2$, $\mathcal{B}$ modifies the tuple as $(ID^*, D_{ID^*}, sp_{kid}^r, \ast, \ast)$ in terms of $\mathcal{A}_1$'s request, where $sp_{kid}^r = (x_{ID^*}, y_{ID^*})$.

(2) If the tuple $(ID^*, D_{ID^*}, sp_{kid}^r, ssk_{ID^*}, \ast, \ast)$ is not contained in $L_2$, $\mathcal{B}$ adds the tuple $(ID^*, \ast, \ast, sp_{kid}^r, \ast, \ast)$ to $L_2$ in terms of $\mathcal{A}_1$'s request, where $sp_{kid}^r = (x_{ID^*}, y_{ID^*})$.

**Hash-2 Query.** In order to facilitate the management of Hash-2 query, a list $L_3 = \{(\lambda, \sigma_{ID^*}, \omega_{ID^*}, \ast, \ast)\}$ is still established by $\mathcal{B}$ to record the participating tuple. As required, $\mathcal{A}_1$ runs the Hash-2 query on the identity $ID^r$, partial public key $Y_{ID^*}$, and Hash-1 query $Q^*$. $\mathcal{B}$ randomly picks a tuple value $(\lambda^*, h^*)$ and calculates $V_{ID^*} = H_2(e(Q^*, Y_{ID^*})^\lambda^*)$, and sends it to $\mathcal{A}_1$. The new tuple $\lambda^*_1$ is stored in $L_2$.

**Encrpyt Query.** For any plaintext $m$, $\mathcal{A}_1$ adaptively requests the encrypt query with identity $ID^r$. $\mathcal{A}_1$ searches list $L_2$ for the tuple $(ID^*, D_{ID^*}, sp_{kid}^r, ssk_{ID^*}, \ast, \ast)$. If the tuple $(ID^*, D_{ID^*}, sp_{kid}^r, ssk_{ID^*}, \ast, \ast)$ is stored in $L_2$, $\mathcal{B}$ returns $sp_{kid}^r$ directly to $\mathcal{A}_1$. Otherwise, the tuple is $(ID^*, \ast, \ast, \ast, \ast, \ast)$ or $(ID^*, \ast, sp_{kid}^r, \ast, \ast, \ast)$, $\mathcal{B}$ calculates the ciphertext $\mu^* = e(V_{ID^*})^\mu_1$ with the replaced key $sp_{kid}^r$, in which $\mu_1 = r_t^e \cdot (\lambda^* \cdot h^*)$. Then, $\mathcal{B}$ returns the tuple $(ID^*, \ast, \ast, \ast, \ast, \ast)$ and $(ID^*, \ast, sp_{kid}^r, \ast, \ast, \ast)$ in $L_2$, respectively.

**Tag Query.** $\mathcal{A}_1$ adaptively requests the tag on any identity $ID^r$ and plaintext block $m_{ID^r}$ by submitting the result of $\sigma_{ID^r}$. Based on the result of tossing the coin in $L_2$, $\mathcal{B}$ aborts. Otherwise, based on the values of $H_2(\omega_{ID^*})$ in $L_2$ and both $D_{ID^r}$ and $sp_{kid}^r$, $\mathcal{B}$ generates the tag for the tuple $(D_{ID^*}, \sigma_{ID^*}, H_2(\omega_{ID^*}))$ by TagGen and returns it to $\mathcal{A}_1$.

**Forgery.** Eventually, a forgery tag $T^*$, which is relevant to plaintext $m^*$ on the identity $ID^r$ with the public key $sp_{kid}^r$, is forged by $\mathcal{A}_1$. If $\tau = 0$, $\mathcal{A}_1$ aborts. Otherwise, based on the aforementioned operations, $\mathcal{A}_1$ holds the following values: $H_1(ID^r) = (g_{ID^r}^r, \sigma_{ID^r} = g_{ID^r}^\lambda, P_0 = g_{ID^r}^\sqrt{\tau}, H_2(\omega_{ID^*}) = g_{ID^*}^\lambda, \sigma_{ID^*}, \text{ and } X_{ID^*})$, and then it can output $g_{ID^r}^\ast = (T^*/X_{ID^r}^{1/\sqrt{\tau}})^{(1/\sqrt{\tau})^{\omega_{ID^*}^2}}$ solving the proposed CDH problem.

**Analysis.** Now, we analyze the probability that $\mathcal{B}$ can guess the correct query of the target data block by simulating operation. Similar to the analysis and proof of [48, 50], $\mathcal{B}$ only halts two queries on partial key query and tag query; therefore, the probability of $\mathcal{B}$ implementing the queries is higher than $(1 - \gamma)^{e_{\mathcal{B}}}$. Assume that the probability of occurrence of output of the right value of $g_{ab}$ for $\mathcal{B}$ is $\varepsilon \cdot \gamma \cdot (1 - \gamma)^{e_{\mathcal{B}}}$. Let

$$e^* = \varepsilon \cdot \gamma \cdot (1 - \gamma).$$

In order to find the minimum value of $e^*$, let us take the derivatives of both sides of (4) with respect to $\gamma$:

$$\frac{de^*}{d\gamma} = \frac{d\varepsilon \cdot \gamma \cdot (1 - \gamma)^{e_{\mathcal{B}}}}{d\gamma} = \varepsilon \cdot (1 - \gamma)^{e_{\mathcal{B}} - 1} - \varepsilon \cdot (1 - \gamma)^{e_{\mathcal{B}} - 1} \cdot (1 - \gamma) \cdot \left(1 + \frac{q_p + q_t}{q_p + q_t + 1}\right).$$

Replace $\frac{de^*}{d\gamma} = 0$; that is,

$$e^* = \varepsilon \cdot (1 - \gamma)^{e_{\mathcal{B}} - 1} \cdot \left(1 + \frac{q_p + q_t}{q_p + q_t + 1}\right).$$

We can obtain $\gamma_{opt} = 1/(1 + q_p + q_t)$. Therefore (4) becomes

$$e^* = \varepsilon \cdot \gamma \cdot (1 - \gamma)^{e_{\mathcal{B}}} \geq \varepsilon \cdot \frac{1}{1 + q_p + q_t} \left(1 - \frac{1}{1 + q_p + q_t}\right)^{q_p + q_t}.$$

According to the formula $\lim_{n \to \infty} (1 + (1/n)) = e$ for $n \in N$, equation (7) becomes

$$e^* \geq e(\left(1 + q_p + q_t\right) \cdot e).$$

Further, simulator $\mathcal{B}$ can solve the CDH problem in polynomial time $t'$ which satisfies $t' \leq t + (q_{H_1} + q_p + 3q_K + 3q_{H_1})t_e + (3t_m + 2t_e) \cdot q_p + q_r + q_e$.

**Theorem 2.** If a PPT adversary $\mathcal{A}_2$ has an advantage to win Game 2 described in Section 4.4 within time $t$ after implementing the most $q_{H_1}$ Hash-1 queries, $q_p$ key queries, $q_e$ encryption queries, and $q_{H_1}$ Hash-2 queries and requesting at most $q_{T_r}$ times tag queries, then there exists a $(\varepsilon', t)$-simulator $\mathcal{B}$ that can address the CDH problem with $\varepsilon' \geq (\varepsilon/(1 + q_K + q_T + q_e)) \cdot e) \cdot t' \leq t + (q_{H_1} + q_p + 3q_K + 3q_{H_1})t_e + (3t_m + 2t_e) \cdot q_p + q_r + q_e$, where one exponentiation costs time $t_c$, one scalar multiplication operation costs time $t_m$ in $G_1$, and $e$ is base of natural logarithm.

**Proof.** On input $(g, g^r, g^s)$ in $G_1$, the CDH algorithm $\mathcal{B}$ has the capability to simulate a data-integrity-verifying security game and output $g_{ab}$ by interacting with adversary $\mathcal{A}_2$ as follows:

Setup. $\mathcal{B}$ chooses the master keys at random and outputs the system parameters $\text{params}$. Then, both $s$ and $\text{params}$ are returned to $\mathcal{A}_2$ by $\mathcal{B}$.

Hash-1 query. $\mathcal{A}_2$ requests the Hash-1 query results for any identity $ID^r$ in terms of its capability. A tuple list $L_1 = \{(I D, Q, r)\}$ is established to record all query data by $\mathcal{B}$. If a certain $ID^r$ has been stored in $L_1$, $\mathcal{B}$ returns $(g^s)^r$ to $\mathcal{A}_2$. [50, 50].
Otherwise, $\mathcal{B}$ selects a random value $r^* \in \mathbb{Z}_q^*$ and responds to $\mathcal{A}_2$ with $Q^* = (g^r)^*$ and then stores $(ID^*, Q^*, r^*)$ in $L_1$. Key query. According to the assumption that adversary $\mathcal{A}_2$ has an ability to access the master keys, it directly initiates the key query of the public/private key pairing $(spk_{1D}, skk_{1D})$. A list $L_2 = \{(ID, spk_{1D}, skk_{1D}, \sigma_{1D}), r^*)$ is established by $\mathcal{B}$ for recording the results of key query.

(1) If $ID^*$ is not stored in list $L_2$, $\mathcal{B}$ picks a value of $x^*$ at random and tosses a coin $\in \{0, 1\}$. Let $\gamma$ denote the probability of $r^* = 0$. Thus $\gamma$ represents the probability of $r^* = 1$.

(2) If $ID^*$ is stored in list $L_2$, $\mathcal{B}$ checks whether the value of $r^*$ is $1$ or $0$. If $r^* = 1$, $\mathcal{B}$ halts and returns $\perp$. Otherwise, assuming that $skk_{1D}$ is already in list $L_2$, $\mathcal{B}$ directly returns it to $\mathcal{A}_2$.

Notably, since $\mathcal{A}_2$ can access the master key to get the private key, there is no partial key query. Hash-2 query. As required, $\mathcal{A}_2$ runs the Hash-2 query for the target value $\omega_{1D}$. In order to record the participating tuple, $\mathcal{B}$ establishes a list $L_3 = \{(V_{1D}, \sigma_{1D}, \omega_{1D}, \lambda, h)\}$ for Hash-2 query. If $\omega_{1D}$ is stored in $L_3$, $\mathcal{B}$ returns the value $H_2(\omega_{1D}) = g^{\omega_{1D}}$ to $\mathcal{A}_2$. Otherwise, $\mathcal{B}$ randomly picks a tuple value $(\lambda', h')$ and calculates $V_{1D} = H_2(\epsilon(Q^*, \lambda', h'))$. Then $\mathcal{B}$ returns $H_2(\omega_{1D})$ to $\mathcal{A}_2$. The new tuple $(V_{1D}, \sigma_{1D}, \omega_{1D}, \lambda', h')$ is added to $L_3$.

Encrypt query. For any plaintext $m$, $\mathcal{A}_1$ adaptively requests the encrypt query with identity$ID^*$. $\mathcal{B}$ searches list $L_2$ for the tuple $(ID^*, spk_{1D}, skk_{1D}, \sigma_{1D}, r^*)$.

(1) If the tuple $(ID^*, spk_{1D}, skk_{1D}, \sigma_{1D}, r^*)$ cannot be found in list $L_2$, $\mathcal{B}$ first requests the Hash-1 query and key query until the tuple $(ID^*, Q^*, r^*)$ and the tossing coin value $r^*$ become existent in $L_1$ and $L_2$, respectively. Then $\mathcal{B}$ calculates the ciphertext $c_{1D} = m_{1D} \cdot \omega_{1D}$ and updates the tuple $(ID^*, spk_{1D}, skk_{1D}, \sigma_{1D}, r^*)$ in $L_2$, where $\sigma_{1D} = \sigma_{1D} \cdot \omega_{1D}$. If $r^* = 1$, $\mathcal{B}$ halts and returns $\perp$; otherwise, $\mathcal{B}$ outputs $\sigma_{1D}$ to $\mathcal{A}_1$.

(2) If the tuple $(ID^*, spk_{1D}, skk_{1D}, \sigma_{1D}, r^*)$ is stored in $L_2$ and $r^* = 0$, then $\mathcal{B}$ directly returns $\sigma_{1D}$ to $\mathcal{A}_1$; otherwise, $\mathcal{B}$ halts and returns $\perp$.

Tag query. For ciphertext $\sigma_{1D}$ associated with plaintext $m^*$, adversary $\mathcal{A}_1$ adaptively performs the tag query with $(\omega_{1D}, \sigma_{1D}, ID^*)$. $\mathcal{B}$ first checks the value of $r^*$ in $L_2$: if $r^* = 1$, $\mathcal{B}$ halts and outputs $\perp$. Otherwise, based on the values of $\omega_{1D}$ in $L_1$ and $\sigma_{1D}$ in $L_2$, $\mathcal{B}$ calculates $D_{1D}$ to generate the tag for the tuple $(\omega_{1D}, \sigma_{1D}, ID^*)$ by step $TagGen$ and then returns it to $\mathcal{A}_2$.

Forgery. Eventually, a forgery tag $T'$, which is relevant to plaintext $m'$ on identity$ID^*$ with the private keys $skk_{1D}$', is forged by $\mathcal{A}_2$. If $r^* = 0$, $\mathcal{B}$ halts and outputs $\perp$. Otherwise, based on the aforementioned operations, $\mathcal{B}$ holds the following values: $P_0 = g^\gamma$, $H_2(ID^*) = g^\omega_{1D}$, $H_2(\omega_{1D}) = g^\omega_{1D}$, $H_2(\omega_{1D}) = g^{\omega_{1D}}$, and then it can output $g^{\omega_{1D}} = (T')^{\epsilon(1+k+q_{td}+q_{td}+q_{td}+q_{td})}$, solving the proposed CDH problem.

Analysis. In this game, there are three times of aborting for $\mathcal{B}$ on key query, encrypt query, and tag query. Therefore, the probability of $\mathcal{B}$ implementing the queries for $\mathcal{A}_2$ without abortion is higher than $1 - \gamma \epsilon (1+q_{td}+q_{td}+q_{td}+q_{td})$. Thus, the probability of occurrence of output of the right value of $g^{\omega_{1D}}$ for $\mathcal{B}$ is $\geq \epsilon' \cdot \gamma \epsilon (1+q_{td}+q_{td}+q_{td}+q_{td}) + \epsilon (1 + q_{td} + q_{td} + q_{td} + q_{td}) \cdot t_0 + q_{td}$.

**Theorem 3.** As long as the DL assumption holds, the probability that adversary $\mathcal{A}_3$ wins Game3, that is, to forge the tag and pass the verification in the scheme, is computationally negligible.

**Proof.** If $\mathcal{A}_3$ wants to win the game, it has to generate the forged integrity proof $P' = (T', F')$ according to the challenge information $chal = (c, k_1, k_2)$ and satisfy the following equations with the nonnegligible probability:

$$e \left( \prod_{k=1}^{z'} T_k, g \right) = \prod_{k=1}^{z'} e \left( \prod_{\gamma_k = \gamma_k} \left( H_2(\omega_{\gamma_k}), \left( X_{\ell_k}, s_{\gamma_k} \right) \right) \right) \cdot e \left( \prod_{k=1}^{z'} H_1(ID_{\ell_k}), P' \right),$$

where $z'$ denotes the count of the group member participating in the challenge and $\ell_k$ represents the number of encrypted data blocks participating in the challenge.
In terms of the assumption in the game, there must be at least one component of vectors satisfying the condition is at most \( q^{-1} \). Clearly, we can find that the probability of \( \sum_{k=1}^{n} a_k \) is less than \( q^{-1} \) which is negligible for a large prime \( q \). We can find that the probability of solving the DL problem is a nonnegligible probability \( 1 - 1/n \); thereby adversary \( \mathcal{A}_3 \) wins Game3 at the negligible probability. 

\[ \text{Theorem 4. The adversary cannot pass the integrity proof by leveraging forged ciphertext.} \]

**Proof.** If \( \mathcal{A}_3 \) wants to win the game, it tries to generate the forged ciphertext \( c' = (\sigma_{ID'_2}, \sigma_{ID'_2}) \) with the forgery identity \( ID'_2 \) and legitimate ciphertext \( \sigma \) in the reprove a participant phase. Suppose that adversary \( \mathcal{A}_4 \) generates its parameters for its identity \( ID'_2 \) through the aforementioned games, for example, \( H_1(ID'_2) \), the private key \( \text{sk}_{ID'_2} = (\mathcal{X}_{ID'_2}, \mathcal{Y}_{ID'_2}) \), the public key \( \text{spk}_{ID'_2} = (\mathcal{X}_{ID'_2}, \mathcal{Y}_{ID'_2}) \), and \( H_2(\omega_{ID'_2}) \). If the tag generating by \( \mathcal{A}_4 \) utilizing these parameters still passes the integrity proof by CSP, then adversary \( \mathcal{A}_4 \) wins this game. Otherwise, it fails.

(1) Assume that the revoked member \( u_i \) calculates the reencryption key \( r_{j 	o i} = H_2(e(H_1(ID'_2), Y_{ID'_2}^i) \cdot e(H_1(ID), Y_{ID}^i)) \) with \( \mathcal{A}_4 \)'s identity \( ID'_2 \) and its public key \( \text{spk}_{ID'_2} \) and then returns \( r_{j 	o i} \) to \( \mathcal{A}_4 \).

(2) \( \mathcal{A}_4 \) reencrypted ciphertext \( \sigma_{ID'_2} = (\sigma_{ID'_2}, \sigma_{ID'_2}) \), where \( \sigma_{ID'_2} = (H_1(ID'_2), Y_{ID'_2}^i) \) and \( \sigma_{ID'_2} = r_{j 	o i} \). Then \( \mathcal{A}_4 \) randomly picks \( x_{ID'_2} \) as the partial key and outputs the forgery tag \( T_{ID'_2} = (H_2(\omega_{ID'_2}) \cdot \sigma_{ID'_2}) \cdot D_{ID'} \).

Through the aforementioned operations, the forgery tag is generated by attacker \( \mathcal{A}_4 \). If the tag passes the integrity proof, the equation \( e(T_{ID'_2}, g) = e(H_2(\omega_{ID'_2}) \cdot \sigma_{ID'_2}, \mathcal{X}_{ID'_2}) \cdot e(H_1(ID'_2)^{x_{ID'_2}}, P_b) \) holds. However, Theorems 1 and 2 have pointed that the tag with the forged private/public key pairing has a negligible probability to win Game1 and Game2; thereby, without the real ciphertext reencrypted by the legitimate private key, the adversary could output the correct integrity proof only with negligible probability. 

**7. Performance Analysis**

In this section, we first show the computation and communication cost of our scheme by theory and then represent the experiment results of the scheme.

7.1. Computation Cost. In our scheme, the computation cost is mainly concentrated on those operations that are computationally complex and time-consuming, such as pairing operation, exponentiation operation, and multiplication operations. For the simplicity of presentation, we use symbols \( C_{pr}, C_{exp}, C_{mul} \), and \( C_{mul} \) to represent the cost of one pairing operation in \( G_1 \times G_1 \rightarrow G_2 \), one exponentiation operation in \( G_1 \), one multiplication in group \( G_1 \), and one multiplication in group \( G_2 \). In addition, the computational overhead of some other operations (such as pseudorandom parameter selection, hash operation, addition, and pseudorandom permutation) is negligible, so these operations are not analyzed. Continue to leverage the above symbols, and let \( n, c, \alpha, \text{and} \alpha' \) denote the number of projects divided into subprojects, members participating in the project, the challenge subprojects, and the members involved in the challenge, respectively. According to the processes of Encrypt and TagGen, the computation costs for \( n \) data blocks are \( (C_{pr} + 2C_{exp}) \) and \( n(2C_{mul} + 2C_{exp}) \), respectively.

In the process of ProveGen, we ignore the generation operation cost of challenge information challenge and focus on the proof information, so the computation cost of this procedure is \( C_{exp} + (c - \alpha')C_{mul} \). In this scheme, we also give the computation cost on the revocation step RevGen for a member of the project, in which the revoked member costs \( 2(C_{pr} + C_{exp}) \), the specified recipient costs \( 2(C_{exp} + C_{mul}) \), and the verification computation cost for the CSP is \( (2l - 1)C_{mul} + 3C_{pr} + 2C_{exp} + C_{mul} \). Table 1 shows the detailed comparison of computation cost and data blocks types among the scheme of papers [7, 39] and ours. As can be seen from Table 1, in step TagGen, our scheme is one more \( C_{mul} \) than [7] and much less than [39]. After all, the cost of [39] is related to the number of participants \( \alpha' \). The cost amount of the step ProveGen is the same as that of [7] but is less than that of paper [39]. In addition, the step VerifyProof is used to verify the correctness of proof information and its computation cost is \( (z + 2)C_{pr} + (2c + z - 2)C_{mul} + (c + 2z)C_{exp} + zC_{mul} \), and the costs of [7, 39] are \( 3C_{pr} + (2c + z)C_{exp} + (2c + z)C_{mul} \) and \( (z + 2)C_{pr} + (c + z)C_{exp} + (c + 2z)C_{mul} + zC_{mul} \), respectively. Compared with the schemes in [7, 39], the cost of our scheme is slightly higher. The reason is that our scheme performs tag generation, verification, and update of ciphertext, and the computational cost is obviously higher than that in literature.

7.2. Communication Cost. In this scheme, the communication cost mainly arises from the challenge information generation phase and proof generation phase. To audit the integrity of the data stored in the cloud service, a verifier sends the challenge
information \((c, k_1, k_2)\) to the CSP, and then proof \(P = (T, F)\) is returned to the verifier by CSP. The communication cost for an integrity proof challenge is \(|n| + 2|q|\) bits, and the communication cost of proof information response is \(2(c + z')|G_1| + |n| + 2|q|\) bits, where \(|q|\) is the element length in \(\mathbb{Z}_q\) and \(|n|\) is the length of the element in set \([1, n]\). In addition, the communication cost for the revocation phase is \((2(l + 1)|G_1|)\), where \(l\) is the number of ciphertext data blocks owned by the revoked member.

### 7.3. Experimental Results

In this experiment, we utilized the Ubuntu Kylin 16.04 LTS (64-bit) operation system equipped with the VMware Workstation 10 with Intel Core i7-8700 3.2 GHz processor and 16 G RAM of the host computer with Win10 operation system using C language to simulate the scheme implementation environment. The Pairing Based Cryptography (PBC) [51] library (version 0.5.14) has been used to execute pairing steps and the Openssl library [52] (version 1.1.1k) is deployed to implement two hash (SHA 256) operations. For the choice of experimental parameters, we used the file params/a.param provided by PBC for type A pairing and constructed a 256-bit order elliptic curve [53] group of type A. To obtain more accurate results, all experiments were run 50 times to get an average.

The step Encrypt needs to execute two time-consuming calculations, namely, pairing operation and exponentiation on group \(G_1\), totally costing almost 602.024 ms for 100 members. We utilize the file with the size of 32 M for experimental demonstration, so the total number of blocks is \(10^6\) which is bounded by the order of the 256-bit group. Suppose that all blocks are averagedly distributed to project members; thereby the number of members getting the blocks is \(10^4\). Figure 2 depicts the time cost result for the members varying from 1 to 100 to generate all ciphertext blocks. Through observation, it is found that the time consumption of encrypting operation is proportional to the number of users. It takes 5.83 ms for a single member to encrypt all its data blocks, while all members can accept the fact that it takes 602.02 ms to encrypt all data blocks. Moreover, the operation that all data blocks are clustered together and encrypted only occurs at the beginning of the project, in the distribution phase of the subproject.

Based on the cost of ciphertext generation, we now evaluate the cost of tag generation experimentally. We still leverage the \(10^6\) ciphertext blocks for the experiment. To carry out the experiment demonstration, we utilize the participant ciphertext number ranging from \(10^5\) to \(10^6\) with an increment of \(10^5\) for each test. From the experimental results in Figure 3, we can observe that the time consumption for tag generation is linear with the increase of the number of blocks, and it takes about 173.9 s to generate tags for all \(10^6\) blocks. As observing the proposed scheme, the entire Encrypt and TagGen processes are executed by only the project manager; thereby the project members just need to download the ciphertext and tags within the appointed time.

We set \(|q| = 256\) bits, \(|n| = 20\) bits, and \(z' = 100\) as in previous work [39]. Based on the previous conclusions [5, 7, 39], if 1% of all blocks are corrupted, 460 challenge blocks picked randomly can achieve 99% error detection.

### Table 1: Comparison of computation cost.

| Schemes            | Tag generation | Proof generation | Verification proof | Data block type |
|--------------------|----------------|------------------|--------------------|-----------------|
| Scheme in [39]     | \((z' + 1)\)   | \(cC_{exp} + cC_{mul_i}\) | \(3C_p + (2c + z')C_{exp} + (2c + z')C_{mul_i} + C_{mul_i}\) | Plaintext       |
| Scheme in [7]      | \(2C_{exp} + C_{mul_i}\) | \(cC_{exp} + (c - z')C_{mul_i}\) | \((z' + 2)C_p + (c + z')C_{exp} + (c + 2z')C_{mul_i} + z'C_{mul_i}\) | Plaintext       |
| Our scheme         | \(2C_{exp} + 2C_{mul_i}\) | \(cC_{exp} + (c - z')C_{mul_i}\) | \((z' + 2)C_p + (c + 2z')C_{exp} + (2c + z' - 2)C_{mul_i} + z'C_{mul_i}\) | Ciphertext      |
probability and 300 challenge blocks chosen at random can successfully achieve 95% malpractice detection probability. In Figure 4, we can see that when the size of project group varies from 500 to 1000 and the number of challenge members ranges from 0 to 200, our scheme can achieve an auditing task with the maximum verification time of 14.664 s and 42 KB by choosing $c = 460$.

8. Conclusion

In this paper, a remote encrypted data integrity auditing scheme stored on a cloud service provider is presented. This scheme addresses the integrity auditing issue for the encrypted data which is shared with numerous members of a group. In our scheme, the sponsor of a shared data group is the project manager who is responsible for the initialization of system parameters, the selection of partial private keys for project members, and the generation of original ciphertext blocks for subprojects. Meanwhile, with the help of certificateless signature idea, the synchronization change between the ciphertext block and the tag is realized, and the problem of auditing the integrity of the ciphertext block is transformed into an equation verification related to the tag. Therefore, based on the above two measures, the key escrow and certificate management in PKI naturally do not exist. With regard to the revocation of the member, our scheme utilizes the homomorphic hash function to transform the ciphertexts of the revoked members into the ciphertexts of the existing members without leaking the information of ciphertext. Finally, the protocol has been proven secure to satisfy adaptively selective ciphertext attack assuming the stability of CDH and DL in bilinear pairing. From the results of the experiment, our scheme is efficient in both computation and communication cost and more secure in a shared group in cloud storage.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61772008, in part by the Science and Technology Major Support Program of Guizhou Province, China, under Grant 20183001, in part by the Key Program of the National Natural Science Union Foundation of China under Grant U1836205, in part by the Science and Technology Program of Guizhou Province under Grant ZK[2021]325, in part by the Science and Technology Program of Guiyang under Grant [2021]1–5, and in part by the Science and Technology Planning Project of Tongren Municipality under Grant [2020]78.

References

[1] Dropbox for business. [Online]. Available: https://www.dropbox.com/business.
[2] Tortoisesvn. [Online]. Available: https://tortoisesvn.net/.
[3] L. Shuib and E. Yadegaridehkordi, “Big data adoption: state of the art and research challenges,” Information Processing & Management, vol. 56, no. 6, 2019.
[4] Y. Deswarte and A. Sa’idane, “Remote integrity checking-how to trust files stored on untrusted servers,” in Integrity and Internal Control in Information Systems VI - IFIP TC11/ WG11.5, Springer, Lausanne, Switzerland, 2003.
[5] Giuseppe Ateniese, R. C. Burns, R. Curtmola et al., “Provable data possession at untrusted stores,” in Proceedings of the 2007
Y. Yu, Y. Zhang, J. Ni, M. H. Au, L. Chen, and H. Liu, "Remote data possession checking with enhanced security for cloud storage," *Future Generation Computer Systems*, vol. 52, pp. 77–85, 2015.

Y. Feng, G. Yang, and J. K. Liu, "A new public remote integrity checking scheme with user and data privacy," *International Journal of Applied Cryptography*, vol. 3, no. 3, pp. 196–209, 2017.

H. Wang, "Identity-based distributed provable data possession in multicloudbased computing," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 328–340, 2015.

A. Juels, S. Burton, and J. Kaliski Jr., "Pors: proofs of retrievability for large files," in *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007*, P. Ning, S. De Capitani di Vimercati, and P. F. Syverson, Eds., pp. 584–597, Alexandria, Virginia, USA, October 2007.

K. D. Bowers, A. Juels, and A. Oprea, "HAIL: a high-availability and integrity layer for cloud storage," in *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds., Chicago, Illinois, USA, November 2009.

B. Wang, B. Li, and H. Li, "K Knox: privacy-preserving auditing for shared data with large groups in the cloud," in *Applied Cryptography and Network Security*, F. Bao, P. Samarati, and J. Zhou, Eds., vol. 7341, pp. 507–525, 2012.

B. Wang, H. Li, and M. Li, "Privacy-preserving public auditing for shared cloud data supporting group dynamics," in *Proceedings of the IEEE International Conference on Communications, ICC*, Budapest, Hungary, June 2013.

X. Liu, Y. Zhang, B. Wang, and J. Yan, "Mona: secure multi-owner data sharing for dynamic groups in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1963–1976, 2013.

J. Yuan and S. Yu, "Public integrity auditing for dynamic data sharing with multiuser modification," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 1, pp. 1182–1191, 2015.

B. Wang, B. Li, and H. Li, "Panda: public auditing for shared data with efficient user revocation in the cloud," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 92–106, 2015.

Y. Yu, Y. Mu, J. Ni, J. Deng, and K. Huang, "Identity privacy-preserving public auditing with dynamic group for secure mobile cloud storage," in *Proceedings of the Network and System Security - 8th International Conference, NSS 2014*, M. Ho Au, B. Carminati, and C.-C. Jay Kuo, Eds., pp. 28–40, Xi’an, China, October 2014.

Giuseppe Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th International ICST Conference on Security and Privacy in Communication Networks, SECURECOMM 2008*, L. Albert, P. Liu, and R. Molva, Eds., September 2008.

Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for security storage in cloud computing," in *Proceedings of the Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security*, pp. 355–370, Saint-Malo, France, September 2009.

Y. Zhu, H. Wang, Z. Hu et al., "Dynamic audit services for integrity verification of outsourced storages in clouds," in *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC)*, W. C. Chu, W. Eric Wong, M. J. Palakal, and C.-C. Hung, Eds., TaiChung, Taiwan, March 2011.

L. Chang, J. Chen, L. T. Yang et al., "Authorized public auditing of dynamic big data storage on cloud with efficient..."
verifiable fine-grained updates,” IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 9, pp. 2234–2244, 2014.

[35] Li Jin, X. Tan, X. Chen, D. S. Wong, and F. X. Opor, “Enabling proof of retrievability in cloud computing with resource-constrained devices,” IEEE Transactions on Cloud Computing, vol. 3, no. 2, pp. 195–205, 2015.

[36] Z. Yang, W. Wang, Y. Huang, and X. Li, “Privacy-preserving public auditing scheme for data confidentiality and accountability in cloud storage,” Chinese Journal of Electronics, vol. 28, no. 1, pp. 179–187, 2019.

[37] N. Garg and S. Bawa, “Id-papc: identity based public auditing protocol for cloud computing,” in Proceedings of the 2017 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE), pp. 14–17, IEEE, Dehradun, India, December 2017.

[38] R. S. Bali and N. Kumar, “Secure clustering for efficient data dissemination in vehicular cyber-physical systems,” Future Generation Computer Systems, vol. 56, pp. 476–492, 2016.

[39] B. Wang, B. Li, H. Li, and F. Li, “Certificateless public auditing for data integrity in the cloud,” in Proceedings of the IEEE Conference on Communications and Network Security, CNS 2013, pp. 136–144, National Harbor, MD, USA, October 2013.

[40] M. Etemad and A. Küpcü, “Generic efficient dynamic proofs of retrievability,” in Proceedings of the 2016 ACM on Cloud Computing Security Workshop, CCSW 2016, E. R. Weippl, S. Katzenbeisser, M. Payer, S. Mangard, E. Androulaki, and M. K. Reiter, Eds., ACM, Vienna, Austria, pp. 85–96, October 2016.

[41] H. Zhuo Hao, S. Sheng Zhong, and N. Nenghai Yu, “A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability,” IEEE Transactions on Knowledge and Data Engineering, vol. 23, no. 9, pp. 1432–1437, 2011.

[42] W. Shen, J. Qin, Y. Jia, H. Rong, J. Hu, and J. Ma, “Data integrity auditing without private key storage for secure cloud storage,” IEEE Transactions on Cloud Computing, vol. 19, 2019.

[43] T. Wu, G. Yang, Y. Mu, F. Guo, R. H. Deng, and Deng, “Privacy-preserving proof of storage for the pay-as-you-go business model,” IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 2, pp. 563–575, 2021.

[44] B. Wang, B. Li, and H. Li, “Knox: privacy-preserving auditing for shared data with large groups in the cloud,” in Applied Cryptography and Network Security, F. Bao, P. Samarati, and J. Zhou, Eds., vol. 7341, pp. 507–525, Springer, 2012.

[45] B. Wang, B. Li, and H. Li, “Oruta: privacy-preserving public auditing for shared data in the cloud,” in Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing, R. Chang, Ed., Honolulu, HI, USA, June 2012.

[46] H. Yao, C. Wang, Bo Hai, and S. Zhu, “Homomorphic hash and blockchain based authentication key exchange protocol for strangers,” in Proceedings of the 2018 Sixth International Conference on Advanced Cloud and Big Data (CBD), pp. 243–248, IEEE, Lanzhou, China, August 2018.

[47] S. S. Al-Riyami and K. G. Paterson, “Certificateless public key cryptography,” in Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, pp. 452–473, Springer, Daejeon, South Korea, December 2003.

[48] J. Li, H. Yan, and Y. Zhang, “Identity-based privacy preserving remote data integrity checking for cloud storage,” IEEE Systems Journal, vol. 15, no. 1, pp. 577–585, 2021.

[49] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, “Relations among notions of security for public-key encryption schemes,” in Proceedings of the 18th Annual International Cryptology Conference, H. Krawczyk, Ed., Springer, Santa Barbara, California, USA, pp. 26–45, August 1998.

[50] N. Garg, S. Bawa, and N. Kumar, “An efficient data integrity auditing protocol for cloud computing,” Future Generation Computer Systems, vol. 109, pp. 306–316, 2020.

[51] B. Lynn. Pairing-based Cryptography Library: https://crypto.stanford.edu/pbc/download.html.

[52] Toolkit Openssl Project. Openssl library. https://www.openssl.org/docs/manmaster/man7/crypto.html.

[53] D. R. L. Brown, “Sec 2: recommended elliptic curve domain parameters,” Standards for Efficient Cryptography, vol. 20, 2010.