Improved Outsourced Provable Data Possession for Secure Cloud Storage

Haibin Yang, Zhengge Yi, Ruifeng Li, Zheng Tu, Xu An Wang, Yuanyou Cui, and Xiaoyuan Yang

1Engineering University of People's Armed Police, Xi'an, China
2Guizhou Provincial Key Laboratory of Public Big Data, Guizhou University, Guiyang, China

Correspondence should be addressed to Xu An Wang; wangxazjd@163.com

Received 1 May 2021; Accepted 8 July 2021; Published 23 July 2021

1. Introduction

Since 2007, as one of the most interesting topics in the computer field, cloud computing has experienced rapid development and has become a key research direction for large-scale enterprises and institutions. Its high flexibility, scalability, high performance ratio, and other characteristics make it serve storage, medical, financial, education and other aspects [1–3]. Among them, cloud storage is an emerging technology developing in cloud computing in terms of data storage [4]. Compared with traditional data storage methods, cloud storage has the advantages of high performance and low cost. Cloud storage uses data storage and data management as its basic functions, allowing users to connect at any location and store local data and information on the cloud, facilitating users’ management of resources.

However, with the widespread application of cloud storage technology, its security has received more and more attention from users and has gradually become the key to the sustainable development of cloud storage technology. On the one hand, cloud service providers (CSPs) may delete users’ data stored in order to free up storage space for their interests or may want to obtain users’ data privacy [5]. On the other hand, the CSP has great openness and complexity, and it is easy to become the central target of various malicious attacks, leading to the loss, leakage, tampering, or damage of users’ data. Therefore, cloud storage integrity audits have emerged to solve the problem. Users regularly audit the integrity of their own data information stored in the cloud, discover whether their data have been discarded or tampered with, and take corresponding remedial measures.

1.1. Related Work. In the early years, cloud audit-related research was mainly about the integrity verification of remote data. Users do not own the original data and can only verify the integrity of the data stored on the cloud server through the protocol. In 2003, Deswarte et al. [6] proposed the first audit scheme that supports remote data integrity verification. The scheme is based on the Diffie–Hellman key exchange protocol using the homomorphic characteristics of RSA signatures and the difficulty of calculating discrete logarithms as a security basis. The entire file is represented by a large number and then subjected to modular exponentiation to achieve remote data integrity audit. However,
this solution will generate a great computing overhead, which is a heavy burden for users. In 2006, Filho et al. [7], based on RSA’s homomorphic hash function, used the hash function to compress large data files into small hash values before performing operations. This scheme reduces the expense of calculation, but it is also not suitable for large-scale data storage in a cloud storage environment. The scheme put forward the important role of homomorphic hash function in remote data integrity verification, which is the biggest contribution of it. In 2008, Sebe et al. [8] based on the idea of partitioning to improve the previous scheme. The scheme divides the large data file into blocks and then each data block is calculated, which greatly reduces the computational expense. But the prover still needs to access all the data when generating the evidence, so this scheme is also not suitable for large data files.

The above schemes all require the user as a verifier to maintain a metadata set for verification. On the one hand, it is easy for users to lose or leak these metadata, which leads to the disclosure of private data. On the other hand, for users with limited computing resources, huge outsourcing data will increase the computing overhead in the audit process. In addition, in the event of a data corruption accident, the user or CSP will shirk each other’s responsibilities and cannot provide effective evidence to confirm who should be responsible for the accident. Thus, scholars have introduced an absolutely impartial third-party auditor to audit on behalf of users. Auditors are more professional than users in terms of data preservation and computing performance, and in the event of an accident, they can be held accountable for solving problems in a fair manner. Therefore, the audit scheme has gradually changed from a private audit between users and CSP to a public audit between users, CSP, and third-party auditors (TPAs). In 2007, Shah et al. [9] proposed a public audit scheme based on the difficulty of discrete logarithm calculation to audit ciphertext data and key integrity. The scheme uses a hash function with a key to precalculate a certain number of response values stored by the auditor. During the audit process, the auditor only needs to match the evidence provided by the server with the prestored response value. However, the number of audits in this scheme will be limited by the number of prestored response values.

The amount of calculation required for the integrity audit of all data is not a small expense even for professional third-party auditors. Scholars have been studying how to increase audit efficiency to reduce computational overhead, but from another aspect, reducing the data content that needs to be audited can also achieve the goal. In 2005, Noar and Rothblum [10] proposed an online memory detection scheme. The scheme studied the sublinear authentication and proposed related authentication protocols. The basic idea of the sublinear authentication is to verify the integrity of all the original data by verifying the integrity of a small part of the data block specified randomly. In 2007, Ateniese et al. [11] proposed the first probabilistic provable data possession (PDP) auditing scheme with both safety and practicality. The scheme is based on RSA’s homomorphic authentication label, which realizes the audit of outsourced data. The metadata of multiple data blocks can be aggregated into one value, which effectively reduces the communication overhead, and a random sampling strategy is adopted to check the user’s remote data instead of verifying all the user’s data, so the calculation cost is effectively reduced.

With the continuous improvement of the audit program, some expansion requirements are constantly raised, for example, audit programs that support privacy protection or batch audits. In 2010, Wang et al. [12] proposed an audit scheme supporting privacy protection through the integration of homomorphic authentication tags and random mask technologies for the first time, in which the bilinear signature is used to support batch audits. In 2013, Yang et al. [13] proposed an audit solution based on the index table technology that supports dynamic data update, and the tag aggregation technology is used to process multiple audit requests from different users to support batch audits in a multi-user multi-cloud environment. In 2015, Hui et al. [14] proposed a public audit scheme based on dynamic hash table (DHT), which can record the attribute information of data blocks to support dynamic data update and improve efficiency. The program also supports privacy protection and batch auditing.

In 2007, Juels and Jr [15] proposed an original proof of retrievability (POR) scheme. Different from the PDP scheme described above, the POR scheme can repair the corrupted data when data are detected to be damaged. The scheme uses sampling and error correction codes to perform fault-tolerant preprocessing on outsourced data files, which can restore data with a certain probability when the data are damaged. In 2008, Shacham and Waters [16] proposed a compact POR scheme. The scheme draws on the idea of homomorphic authentication tags and effectively aggregates the evidence into a smaller value, allowing the verifier to perform any number of audits, while also reducing the communication overhead in the verification process. POR and PDP have their own application scenarios. The former can recover damaged data, and the latter is more flexible which can be applied to privacy protection, dynamic auditing, and batch auditing. Cloud auditing schemes are constantly being improved based on users’ needs. When scholars are studying how to reduce computing and communication costs, they also try to expand functions horizontally or combine them with different technologies for innovation. In 2013, Zhao et al. [17] proposed the first identity-based cloud audit scheme, which uses random mask technology to achieve privacy protection. In the identity-based cloud auditing scheme, only the private key generator (PKG) holds a public-private key pair and its public key certificate. The public keys of other users can be calculated based on the identity information, and the private key is generated by PKG, which will reduce the calculation and communication overhead of the scheme. In 2015, Zhang and Dong [18] proposed the first certificateless cloud audit scheme that can resist malicious auditors. Thus, the concept of malicious auditors was introduced into the cloud audit program for the first time. The certificateless cloud audit scheme can solve the certificate management problem in the certificate-based cloud audit solution and the key escrow problem in the identity-based audit solution. In 2016, Xin
et al. [19] combined the transparent watermarking technology with the auditing scheme, proposing a scheme to audit the integrity of static multimedia data, which can greatly save multimedia data calculation and storage costs.

1.2. Our Contribution. Recently, an outsourced dynamic provable data possession scheme with batch update for secure cloud storage (ODPDP) was proposed by Wei et al. [20]. However, we find that there are security problems in their scheme. The adversary can easily forge authentication labels. Even if all the outsourced data have been deleted by the cloud server, CSP can still give a correct data possession proof. And malicious auditors do not carry out auditing work but can conspire with the cloud server forging audit log to deceive client. Finally, we propose an improved secure auditing protocol, and roughly analysis shows that our new protocol is secure and can be used in practical settings.

1.3. Organization. This paper is organized as follows. In Section 2, we describe the system model of our scheme. In Section 3, we review Guo et al.’s outsourced dynamic provable data possession scheme with batch update for secure cloud storage. In Section 4, we give our attacks to the original scheme to show that it is not secure. In Section 5, we give our improved secure auditing scheme and roughly analyze its security. Finally, in Section 6, we draw some conclusions.

2. System Model

First of all, for the convenience of understanding, the notations and their corresponding meanings of this paper are described in Table 1.

There are three entities in the system model of ODPDP scheme: CSP (cloud service provider), client, and auditor, as depicted in Figure 1. The following three entities are involved:

1. CSP (cloud service provider): the service provider, which has abundant computing power and physical storage capacity, realizes the maintenance and management of the received data from client. This part is honest and curious.

2. Client: the data owner, which outsources the data that needs to be calculated and stored to the CSP, concern the integrity of the outsourced data, and checks whether the auditor is honest in the audit work regularly.

3. Auditor: the third-party auditor accepts audit task from the client and is responsible for ensuring the integrity of the data of the client stored in CSP.

The protocols used in the ODPDP scheme are as follows:

1. Setup \( (t^3) \rightarrow \{\text{client: sk}_c, v_k, sk, pk; \ \text{auditor: sk}_a, v_k; \\text{CSP: sk}_{CSP}, v_{kCSP}\} \): random key generation protocol. The users input a security parameter \( K \) and then it generates pairs of signing-verifying keys (SKP, VKP) for each participant. For the convenience of expression, we assume that all the participants involved in each subsequent protocol always take the owners’ public key and its own secret key as input.

2. Store \( (\text{client: M}) \rightarrow v(\text{client: P,C; auditor: P, CT; CSP: P M}) \): the interactive protocol among the three parties. It takes the keys of the three participants as input and the data \( M \) owner by client, and then outputs the processed data \( M = \{M, \Sigma\} \) for the CSP. \( \Sigma \) is generated by the client through the secret key \( sk \) as the tag vector of \( M \). And for the auditor, it outputs a RBMTT based on \( M \). Besides, it outputs a public parameter \( P \) that is confirmed by three participants and a contract \( C \) between the client and the auditor.

3. AuditData \( (\text{auditor: Q, T; CSP: M}) \rightarrow \{\text{auditor: dec}_a, L\} \): the interactive protocol between the CSP and auditor to make the auditor to be sure that the integrity of \( M \) in CSP is in good condition. The auditor takes the functionality of Bitcoin to extract pseudo-random challenge \( Q \) and then sends it to the CSP. The CSP computes a proof of data possession based on \( Q \) and \( M \) and then sends it to the auditor for verification. The auditor verifies the proof from the CSP through \( Q, T, pk \) and then outputs a binary value \( \text{dec}_a \) as a response to indicate whether the auditor accepts the proof or not and a log entry \( L \) to record the auditing behavior.

4. AuditLog \( (\text{client: B; auditor: T, A}) \rightarrow \{\text{client: dec}: \text{interactive protocol between the client and auditor, which can help the client to audit a log file A consisting of the log entries recorded by the auditor. The aim of this protocol is to check whether the auditor accomplished the auditing task or not. After the auditor receives the random subset \( B \) of the Bitcoin block index released by the client, it calculates the proof of the specified log based on \( B, T, A \) and sends the proof to the client. The client checks the received proof and then outputs a binary value \( \text{dec}_a \), which indicates if it admits the proof. Compared with the AuditData protocol, the frequency is much lower and the computational efficiency is much higher in this protocol.

3. Review of Guo et al.’s Scheme

In Guo et al.’s scheme, three parties are involved, which are the user, the auditor, and the CSP. In their scheme, they used the rank-based Merkle tree (RBMT) to protect the integrity of data block hashes, while the hash values and tags protect the integrity of data blocks. Then, they proposed a multi-leaf-authenticated (MLA) solution for RBMT to authenticate multiple leaf nodes and their indices all together without storing status value and height value. At the same time, they proposed an efficient homomorphic verifiable tag (EHVT) based on BLS signature to reduce clients’ log verification effort. For the specific implementation of these technologies, one can refer to the original paper [20]. Concretely, the following algorithms are involved in their scheme.
3.1. Setup Protocol. Each participant $P \in \{\text{CSP}, \text{client}, \text{auditor}\}$ performs Key Gen to obtain $sk_P$ and $vk_P$. In addition, the client samples $s + 1$ random elements $\alpha_1, \alpha_2, \ldots, \alpha_s, x \in \mathbb{Z}_q$ and computes $g_1 = g^{\alpha_1}, g_2 = g^{\alpha_2}, \ldots, g_s = g^{\alpha_s}, y = g^x \in G$. Then, the client chooses a random element $\lambda \in G$, and the secret key and public key are denoted as $(sk, pk) = (\alpha_1, \alpha_2, \ldots, \alpha_s, x)$ and $(g, \lambda, g_1, g_2, \ldots, g_s, y)$.

3.2. Store Protocol. The data file is divided into $M = (m_1, m_2, \ldots, m_n)$, and each data block consists of $s$ sectors and has the form $m_i = m_{i1}m_{i2} \ldots || m_{is} (1 \leq i \leq n)$ such that each sector $m_{iz} \in \mathbb{Z}_q (1 \leq z \leq s)$, where $||$ denotes concatenation.

**Constructing RBMT.** With all data blocks, the client first computes $h_i = H_2(m_i) (1 \leq i \leq n)$. Then, the client constructs RBMT $T$ on top of the ordered hash values, meaning that each leaf node $w_i$ stores the corresponding hash value $h_i$.

**Computing EHVT.** Based on $g, \lambda$ and secret key $sk$, the client computes

$$
\sigma_i = \left(\lambda^{h_i} \cdot g^\sum_{z=1}^{s} m_{iz} \right)^x \in G (1 \leq i \leq n).
$$

**Figure 1:** System model.

| Table 1: Notations. |
|---------------------|
| Notations | Descriptions |
| $G$ | A multiplicative cyclic group |
| $H$ | A secure hash function such that $H(\cdot): \{0,1\}^* \rightarrow \mathbb{Z}_p$ |
| $\mathbb{Z}_p$ | A prime field |
| $g$ | The generator of group $G$ |
| $M$ | The user’s data file |
| $m_i\{m_{i1}, \ldots, m_{is}\}$ | The user’s data file with $n$ blocks and $s$ slices |
| $\alpha_i, x$ | The secret random values of user |
| $\lambda$ | A random value that makes up the public key |
| $sk$ | The secret key of user |
| $pk$ | The public key of user |
| $\sigma_i$ | The authentication label for the $i$-th data block |
| $\Sigma$ | The collection of authentication labels |
| $M$ | The processed data of user which include $M$ and $\Sigma$ |
| $Q$ | The challenge set sent by the auditor to CSP |
| $T$ | A rank-based Merkle tree built from user’s data |
| $\Lambda$ | The local log file of the auditor |
| $L$ | A record of an auditor’s auditing work |
| $\rho$ | The proof generated by the CSP or auditor |
| $\cup_p$ | A multi-proof based on multiple challenged leaf nodes |
3.2.1. Outsourcing Data. The client sends $\mathcal{M}$ and $\text{Sig}_{\text{ Blake}}(\mathcal{M})$ to CSP. CSP verifies $\text{Sig}_{\text{ Blake}}(\mathcal{M})$, and if the verification is passed, CSP accepts $\mathcal{M}$.

3.2.2. Outsourcing Auditing Work. Auditing work is outsourced to the auditor and CSP sends $T$ with $\text{Sig}_{\text{ Blake}}(T)$ to the auditor. Then, the auditor verifies $\text{Sig}_{\text{ Blake}}(T)$.

3.2.3. Agreeing Parameters. A public parameter $\mathcal{P} = \{n, h_{\text{ root}}\}$ needs to be agreed on by three participants, where $n$ denotes the number of data blocks and $h_{\text{ root}}$ denotes the Merkle root of $T$. In addition, the client and auditor also need to agree on a contract $C = \{B, F, I\}$, where $C$ denotes the auditor’s checking policy. The auditing work will start from a Bitcoin block index $B$, the auditing frequency depends on $F$, and $I$ dictates the number of challenged data blocks for each checking.

Then, the client deletes $\mathcal{M}$ and $T$ from its local storage, and she only maintains a constant amount of metadata.

3.3. AuditData Protocol. The scheme leverages the Bitcoin blockchain as a time-dependent pseudo-random source to generate periodic challenges. The auditor inputs the time $t \in \tau$ to obtain a hash value $\text{hash}^{(b)} \in \{0, 1\}^{80}$ of the latest block that has appeared since time $t$ in Bitcoin blockchain. Then, PRBG is invoked on the input hash $(b)$ to acquire pseudo-random bits, which will be used by the auditor to select a pair of keys $k_{\pi}^{(b)}, k_{\eta}^{(b)}$. At last, the auditor generates a challenge $Q^{(b)} = \{b, k_{\pi}^{(b)}, k_{\eta}^{(b)}\}$ and sends it to CSP, where the block $b$ corresponds to the timer.

Upon receiving the challenge $Q^{(b)}$, CSP first computes the challenged indices and coefficients as follows:

$$i_{\pi} = \pi_{k_{\pi}^{(b)}}(\eta),$$

$$a_{\eta} = f_{k_{\eta}^{(b)}}(\eta) \quad (1 \leq \eta \leq l). \tag{2}$$

Then, CSP computes the proof of data possession to prove the integrity of the challenged data blocks as follows:

$$\mu_{z}^{(b)} = \sum_{p=1}^{l} a_{\eta} m_{i_{\pi}^{(b)}} \in \mathbb{Z}_{q}, \quad 1 \leq z \leq s, \tag{3}$$

$$\sigma^{(b)} = \prod_{p=1}^{l} \sigma_{\eta}^{(b)} \in G. \tag{4}$$

Finally, CSP responds the auditor with the proof $\rho^{(b)} = \{\mu_{1}^{(b)}, \mu_{2}^{(b)}, \ldots, \mu_{s}^{(b)}, \sigma^{(b)}\}$. Then, the auditor verifies the correctness of $\rho^{(b)}$. First, the auditor computes the challenged indices and coefficients. Second, the auditor computes the value with $T$ as follows:

$$e^{(a^{(b)}), g} = e\left(h^{(b)} \cdot \prod_{z=1}^{s} g_{z}^{(b)}, y^{(b)}\right). \tag{5}$$

If the equation holds, the auditor assures that the challenged data blocks are intact. Lastly, the auditor saves the log entry in the log file $\Lambda$ to record the auditing work as follows:

$$L^{(b)} = \{t, Q^{(b)}, h^{(b)}, \rho^{(b)}, \text{Sig}_{\text{ Blake}}(\rho^{(b)})\}. \tag{6}$$

3.4. AuditLog Protocol. The client chooses a random subset $B$ of indices of Bitcoin blocks and sends it to the auditor. Once receiving $B$, the auditor finds $Q^{(b)}$, $h^{(b)}$, and $\rho^{(b)}$ from his log file $\Lambda$ for each $b \in B$ and computes.

---

In addition, for each $b \in B$, the auditor reads $k_{\pi}^{(b)}$ from $Q^{(b)}$ and computes the challenged indices $i_{\eta}(1 \leq \eta \leq l)$ by invoking $\pi_{k_{\pi}^{(b)}}(\eta)$. After eliminating the repetitive indices, the last ordered challenge index vector is denoted by
C = (i_1, i_2, ..., i_c). Then, the auditor obtains the corresponding multi-proof \( \mathcal{U}_p \). At last, the auditor generates the proof of appointed logs as follows:

\[
\rho^{(B)} = \{ \mathcal{U}_p, h^{(B)}, \mu_1^{(B)}, \mu_2^{(B)}, ..., \mu_{\lambda}^{(B)}, \sigma^{(B)} \},
\]

and sends it to the client with \( \text{Sig}_{\text{sk}}(\rho^{(B)}) \).

After verifying \( \text{Sig}_{\text{sk}_s}(\rho^{(B)}) \), for each \( b \in B \), the client first invokes PRBG(\( h^{(b)} \)) to get \( Q^{(b)} \) and reconstructs the challenged indices and coefficients \( i_{\eta}, a_{\eta} \) \((1 \leq \eta \leq \lambda)\). Then, the client verifies the correctness of \( \mathcal{U}_p \). If the verification is passed, it means that all the challenged leaf nodes \( \omega_j \) \((1 \leq j \leq c)\) in \( \mathcal{U}_p \) are authenticated, and then the corresponding hash value \( h_j \) stored in leaf node \( \omega_j \) can be accepted by the client. Finally, with \( \lambda \) and all authenticated \( h_j \), the client verifies \( h(B) \) by checking the following equation:

\[
s^{(B)} = \prod_{b \in B} \prod_{\eta=1}^{\lambda} a_{\eta}^{x_n} B_{\eta}^{x} = \prod_{b \in B} \prod_{\eta=1}^{\lambda} \left( \lambda^{g_{\sum_{i=1}^{\lambda} a_{\eta} m_{\eta}}} \right)^{a_{\eta} x_n} = \prod_{b \in B} \left( \prod_{\eta=1}^{\lambda} a_{\eta} h_{\eta} \right) \left( \sum_{\eta=1}^{\lambda} a_{\eta} m_{\eta} \right)^{x_n} = \left( \lambda^{g_{\sum_{i=1}^{\lambda} a_{\eta} m_{\eta}}} \right)^{x_n} = \left( h^{(B)} \cdot g_{\sum_{i=1}^{\lambda} a_{\eta} m_{\eta}}^{x_n} \right)^{x_n}.
\]

4. Our Attack

In Guo et al.'s auditing protocol, their security model indicates that the malicious CSP cannot forge false proof to pass the challenger’s verification and the client can resist malicious CSP and auditor collusion attacks. However, we find that we can extract some key information from the client’s \( pk \), data blocks, and their corresponding tags which are known to CSP. In this section, we firstly show how CSPs extract key information and how to use this information to forge "correct" data blocks and their corresponding tags. Then, we will show how malicious CSP and auditor collude to use false proof to pass the client’s verification.

4.1. Attack I. Our attack is based on the following observation: the public key of the client is

\[
\begin{align*}
\text{pk} = (g, \lambda, g_1, g_2, ..., g_s, y),
\end{align*}
\]

and this public key is known to all, and thus the adversary can easily use it to forge authentication label. Concretely, the adversary launches the following attack:

(1) In the Store Protocol, CSP can receive \( M \) from client, which includes the data of the client and its corresponding authentication tags. The adversary can get a large number of authentication tags. The adversary can get a large number of authentication tags as follows:

\[
\begin{align*}
\sigma_1 &= \left( \lambda^{h_{1}} \cdot g_{\sum_{i=1}^{\lambda} a_{\eta} m_{\eta}} \right)^{x_n}, \\
\sigma_2 &= \left( \lambda^{h_{2}} \cdot g_{\sum_{i=1}^{\lambda} a_{\eta} m_{\eta}} \right)^{x_n}, \\
&\vdots \\
\sigma_n &= \left( \lambda^{h_{n}} \cdot g_{\sum_{i=1}^{\lambda} a_{\eta} m_{\eta}} \right)^{x_n}.
\end{align*}
\]

(2) The above equations can be rewritten as follows:
\[ \sigma_1 = (\lambda^x)^{h_1} \cdot (g^{a_1x})^{m_{11}} \cdot (g^{a_2x})^{m_{12}} \cdots (g^{a_nx})^{m_{1n}} \]
\[ \sigma_2 = (\lambda^x)^{h_2} \cdot (g^{a_1x})^{m_{21}} \cdot (g^{a_2x})^{m_{22}} \cdots (g^{a_nx})^{m_{2n}} \]
\[ \vdots \]
\[ \sigma_n = (\lambda^x)^{h_n} \cdot (g^{a_1x})^{m_{n1}} \cdot (g^{a_2x})^{m_{n2}} \cdots (g^{a_nx})^{m_{nn}}. \]

The CSP knows the data blocks of client, and it can calculate the corresponding hash values through \( H(m_i) \):
\[ h_1, h_2, \ldots, h_n. \]

In order to simplify the attack process, let \( s = 2 \), \( A = \lambda^x \), \( B = g^{a_1x} \), \( C = g^{a_2x} \), and take three linear irrelevant tags \( \sigma_1, \sigma_2, \sigma_3, \sigma_i \) as follows:
\[ \sigma_1 = A^{h_1} \cdot B^{m_{11}} \cdot C^{m_{12}} \]
\[ \sigma_2 = A^{h_2} \cdot B^{m_{21}} \cdot C^{m_{22}} \]
\[ \sigma_3 = A^{h_3} \cdot B^{m_{31}} \cdot C^{m_{32}}. \]

For \( s = 2 \), the adversary can compute \( A, B, \) and \( C \) as follows:
\[ \sigma_1^{h_1} = A^{h_1} \cdot B^{m_{11}} \cdot C^{m_{12}} \]
\[ \sigma_1^{h_2} = A^{h_2} \cdot B^{m_{21}} \cdot C^{m_{22}} \]
\[ \sigma_1^{h_3} = A^{h_3} \cdot B^{m_{31}} \cdot C^{m_{32}}. \]

For \( s = 3 \), the adversary computes
\[ \left( \frac{\sigma_{12}}{\sigma_{11}} \right)^{m_{21}h_3 - m_{11}h_1} = B^{(m_{11}h_2 - m_{11}h_1)} \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1), \]
\[ \left( \frac{\sigma_{12}}{\sigma_{11}} \right)^{m_{21}h_3 - m_{11}h_1} = B^{(m_{11}h_2 - m_{11}h_1)} \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1), \]
\[ \left( \frac{\sigma_{12}}{\sigma_{11}} \right)^{m_{21}h_3 - m_{11}h_1} = B^{(m_{11}h_2 - m_{11}h_1)} \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1), \]
\[ \left( \frac{\sigma_{12}}{\sigma_{11}} \right)^{m_{21}h_3 - m_{11}h_1} = B^{(m_{11}h_2 - m_{11}h_1)} \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1), \]

From this equation, we can know
\[ C = \left( \frac{(\sigma_{12}^{h_1} / \sigma_{11}^{h_1})^{m_{21}h_3 - m_{11}h_1} \cdot (m_{11}h_2 - m_{11}h_1) \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1) \cdot (m_{11}h_1 - m_{21}h_1)}{(\sigma_{12}^{h_1} / \sigma_{11}^{h_1})^{m_{21}h_3 - m_{11}h_1}} \right). \]
The value of $B$ can be obtained by substituting the value of $C$ into formula (3):

$$B = \left\{ \begin{array}{l}
\sigma^a_1 \cdot \sigma^a_3 \cdot \left\{ \frac{m_{i_1}h_2 - m_{i_2}h_1}{(m_{i_2}h_2 - m_{i_2}h_1)} \right\} \cdot \frac{1}{\mu(m_{i_1}h_2 - m_{i_2}h_1)(m_{i_1}h_2 - m_{i_2}h_2)} \\
\frac{m_{i_1}h_2 - m_{i_2}h_1}{(m_{i_2}h_2 - m_{i_2}h_1)} \cdot \frac{\left(\frac{\sigma^a_1}{\sigma^a_3}\right)^h_1}{\frac{\sigma^a_3}{\sigma^a_1} \cdot m_{i_2}h_2 - m_{i_2}h_1} \cdot \left\{ \frac{1}{\mu(m_{i_1}h_2 - m_{i_2}h_1)(m_{i_1}h_2 - m_{i_2}h_2)} \right\} \cdot \frac{1}{\mu(m_{i_1}h_2 - m_{i_2}h_1)(m_{i_1}h_2 - m_{i_2}h_2)} \end{array} \right\} \cdot \frac{1}{\mu(m_{i_1}h_2 - m_{i_2}h_1)(m_{i_1}h_2 - m_{i_2}h_2)}.
\right.$$ (21)

According to formula (3), the following results can be obtained:

$$A = \left( \frac{\sigma^a_1}{b^m_{i_1} \cdot C^m_{i_2}} \right)^{\mu(h_1)}. \tag{22}$$

The value of $A$ can be obtained by substituting the value of $B$ and $C$ into the above equation.

Through the above process, the adversary can obtain the value of $\lambda^a, g^{a^a}, g^{a^a}$. Significantly, when $s > 2$, the value of $g^{a^a}, g^{a^a}, \ldots, g^{a^a}$ can be calculated in the same way. In this way, the adversary can get the key parameters of tags.

(4) Now, the malicious cloud server modifies data blocks

$$m_{i_1}, m_{i_2}, \ldots, m_{i_1}, m_{i_2}, \ldots, m_{i_1}, m_{i_2}, \ldots, m_{i_1}$$

to be any other data blocks

$$m_{i_1}, m_{i_2}, \ldots, m_{i_1}, m_{i_2}, \ldots, m_{i_1}, m_{i_2}, \ldots, m_{i_1}.$$

(23)

(5) The adversary knows the value of

$$\lambda^a, g^{a^a}, g^{a^a}, \ldots, g^{a^a}.$$

and thus it can compute the forged authentication label for modified data blocks as follows:

$$\sigma_1 = (\lambda^a)^{\mu_{i_1}}, (g^{a^a})^{m_{i_1}}, (g^{a^a})^{m_{i_2}}, \ldots, (g^{a^a})^{m_{i_1}},$$

$$\sigma_2 = (\lambda^a)^{\mu_{i_2}}, (g^{a^a})^{m_{i_2}}, (g^{a^a})^{m_{i_2}}, \ldots, (g^{a^a})^{m_{i_2}},$$

$$\vdots$$

$$\sigma_n = (\lambda^a)^{\mu_{i_n}}, (g^{a^a})^{m_{i_n}}, (g^{a^a})^{m_{i_n}}, \ldots, (g^{a^a})^{m_{i_n}}.$$ (26)

4.2. Attack II. Our attack II is based on the following observation: even if the CSP does not store any data blocks, the auditor does not need to carry out the audit work and store RBMT $T$, and the CSP and the auditor can conspire to generate the correct log file, which makes the client believe that CSP stores data integrally and the auditor performs the audit work honestly. Concretely, the attack is the following:

(1) Store Protocol: after receiving the client’s data $M$ and its corresponding tag collection $T$, the CSP first verifies the correctness of their signatures according to the original scheme. The malicious cloud server can get the value of $\lambda^a, g^{a^a}, g^{a^a}, \ldots, g^{a^a}$ through doing the same as the attack I. Then, CSP deletes the client’s data and its corresponding tags. After receiving the auditing work and $T$ from client, the auditor does the same as original scheme but deletes $T$.

(2) Audit Data Protocol: in this step, the malicious cloud server and the auditor complete the interactive process of challenge and response, but the CSP does not have the real data, and the auditor does not need to complete the verification work.

(a) The auditor generates a challenge $Q^{(b)} = (b, k^{(b)}, k_j^{(b)})$ as original scheme and sends it to CSP.

(b) After receiving the challenge $Q^{(b)}$, CSP first computes the challenged indices as follows:

$$i_\eta = \pi^{(b)}_{k^{(b)}}(\eta),$$

$$a_\eta = f^{(b)}_{k^{(b)}}(\eta) (1 \leq \eta \leq l).$$ (27)

(c) To generate the proof, CSP randomly chooses $gg$ and $h_\eta \in Z_q(j \in i_\eta)$ and computes a combination of the challenged blocks as

$$\mu^{(b)} = \sum_{\eta=1}^{\mu} a_\eta m_{i_\eta} \in Z_q(1 \leq \eta \leq s).$$

(d) For any $m_{i_\eta} \in Z_q(j \in i_\eta)$, the malicious cloud server computes the forged tags as

$$\sigma_\eta = (\lambda^a)^{\mu_{i_\eta}} (g^{a^a})^{m_{i_\eta}} \ldots (g^{a^a})^{m_{i_\eta}},$$

aggregates the tags as $\sigma^{(b)} = \prod_{\eta=1}^{\mu} a_\eta \in G$.

(e) CSP responds the auditor with the proof $\widehat{\rho}^{(b)} = \{\mu^{(b)}, \pi^{(b)}_1, \pi^{(b)}_2, \ldots, \pi^{(b)}_s, \sigma^{(b)}\}$ and its signature $\operatorname{Sig}_{\operatorname{decSP}}(\widehat{\rho}^{(b)})$. Note that CSP also need to send the forged hash value $\overline{h}_\eta \in Z_q(j \in i_\eta)$ to the auditor.

(f) The auditor verifies the validity of $\operatorname{Sig}_{\operatorname{decSP}}(\widehat{\rho}^{(b)})$, and if it is correct, the auditor does the next
\[
\alpha_n
\]

5.1. Setup Protocol.

Each participant protocol. In this section, we give an improved secure auditing calculation. First, with the value of \( h_i \in Z_q \) received from CSP, the auditor computes the value as follows:

\[
\tilde{\mu}_z^{(b)} = \sum_{b \in B} \tilde{\mu}_z \in Z_q \quad (1 \leq z \leq s). 
\]

Finally, the auditor does not need expand a lot of computational expanse to verify the proof \( \tilde{\rho}^{(b)} \), but creates the following log directly:

\[
\sum_{b \in B} \tilde{\mu}_z \in Z_q \quad (1 \leq z \leq s). 
\]

(3) Audit Log Protocol: here we show that the malicious auditor has the ability to generate correct log file, which can convince the client that he has honestly performed the auditing work and that CSP has honestly stored all data.

(a) The auditor finds \( Q^{(b)}, \tilde{\rho}^{(b)}, \) and \( \tilde{\rho}^{(b)} \) from his log file \( \Lambda \) for each \( b \in B \) with the random subset \( B \) of indices of Bitcoin blocks and computes

\[
\tilde{\rho}^{(b)} = \prod_{b \in B} \tilde{\rho}_b \cdot \sum_{b \in B} \tilde{\mu}_b \cdot \tilde{\mu}_b \quad (1 \leq z \leq s). 
\]

Then, the auditor generates the proof of appointed logs as original scheme as follows:

\[
\tilde{\rho}^{(b)} = \left\{ \sum_{b \in B} \tilde{\mu}_z \right\}, \quad (1 \leq z \leq s) 
\]

and sends it to the client with \( \text{Sig}_{\text{CSP}}(\tilde{\rho}^{(b)}) \).

(b) The client first verifies the correctness of \( \text{Sig}_{\text{CSP}}(\tilde{\rho}^{(b)}) \) and \( \sum_{b \in B} \tilde{\mu}_b \), and then verifies the following equations:

\[
\tilde{\mu}_z^{(b)} = \prod_{b \in B} \tilde{\mu}_b \cdot \tilde{\mu}_b \quad (1 \leq z \leq s). 
\]

Here we can verify that the forged proof \( \tilde{\sigma}^{(b)} \) is a valid one if the following equation holds:

\[
\tilde{\sigma}^{(b)} = \prod_{b \in B} \tilde{\sigma}_b \cdot \sum_{b \in B} \tilde{\mu}_b \cdot \tilde{\mu}_b \quad (1 \leq z \leq s). 
\]

5. Improved Secure Auditing Protocol

In this section, we give an improved secure auditing protocol.

5.1. Setup Protocol. Each participant \( \mathcal{P} \in \{ \text{CSP}, \text{client}, \text{auditor} \} \) performs Key Genss to obtain skp and vkp. In addition, the client chooses \( s + 1 \) random elements \( \alpha_1, \alpha_2, \ldots, \alpha_s, x \in Z_q \) and computes \( g_1 = g^{\alpha_1}, g_2 = g^{\alpha_2}, \ldots, g_s = g^{\alpha_s}, y = g^x \in G \). The client's secret keys and public keys are denoted as sk = \((\alpha_1, \alpha_2, \ldots, \alpha_s, x)\) and pk = \((g, g_1, g_2, \ldots, g_s, y)\).

5.2. Store Protocol. The data file held by the client is divided into \( n \) data blocks as \( M = (m_1, m_2, \ldots, m_n) \), and each \( m_i \) consists of \( s \) sectors. More precisely, the data block has the form \( m_i = m_{i1}m_{i2} \ldots \| m_{in} \) (1 \( \leq i \leq n \)) such that each sector \( m_{iz} \in Z_q \) (1 \( \leq z \leq s \)), where \( \| \) denotes concatenation.

Constructing RBMT. With all data blocks, the client computes hash values \( h_i = H_2(m_i) \) (1 \( \leq i \leq n \)). Then, it constructs RBMT \( T \) on top of the ordered hash values, meaning that each leaf node \( \omega_i \) stores the corresponding hash value \( h_i \).

Computing EHVT. Based on \( g \) and sk, the client computes

\[
\sigma_i = \left( h_i \cdot g^{\sum_{z=1}^{s} a_z m_{iz}} \right) \quad (1 \leq i \leq n). 
\]

Then, the client generates the processed data \( \mathcal{M} = [M, \Sigma] \), where \( \Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) \).
5.2.1. Outsourcing Data. The client sends $\mathcal{M}$ and $\text{Sig}_{skc}(\mathcal{M})$ to CSP. CSP verifies $\text{Sig}_{skc}(\mathcal{M})$; if $\text{Sig}_{skc}(\mathcal{M})$ passes, CSP accepts $\mathcal{M}$.

5.2.2. Outsourcing Auditing Work. The client outsources auditing work to the auditor by sending $T$ with $\text{Sig}_{skc}(T)$. If $\text{Sig}_{skc}(T)$ is passed, the auditor accepts $T$.

5.2.3. Agreeing Parameters. A public parameter $\mathcal{P} = \{n, h_{\text{root}}\}$ needs to be agreed on by three participants, where $n$ is the total number of data blocks and $h_{\text{root}}$ is the Merkle root of $T$. In addition, the client and auditor need to further agree on a contract $C = \{B, F, l\}$ that specifies the checking policy for the auditor. $B$ denotes a Bitcoin block index from which the client chooses a random subset $b_{i}^{(l)}$. Then, PRBG is invoked on the hash $(b_{i})$ to obtain long enough pseudo-random bits, which will be sequentially used by the auditor to select a pair of keys $k_{s}^{(l)}$, $k_{f}^{(l)}$. At last, the auditor generates a challenge $Q^{(l)} = \{b, k_{s}^{(l)}, k_{f}^{(l)}\}$ and sends it to the CSP, where the block $b$ corresponds to the time $t$.

Upon receiving the challenge $Q^{(l)}$, the CSP first computes the challenged indices and coefficients as follows:

$$i_{\eta} = \pi_{k_{s}^{(l)}}(\eta),$$

$$a_{\eta} = f_{k_{f}^{(l)}}(\eta) (1 \leq \eta \leq l).$$

Then, the CSP computes the proof of data possession to verify the integrity as follows:

$$\mu_{z}^{(l)} = \sum_{\eta=1}^{l} a_{\eta} m_{z_{\eta}} \in \mathbb{Z}_{q} (1 \leq z \leq s),$$

$$\sigma_{z}^{(l)} = \sum_{\eta=1}^{l} a_{\eta} m_{z_{\eta}} \in G.$$  \hspace{1cm} (36)

Finally, the CSP responses the auditor with the proof \(\rho^{(l)} = \{\mu_{1}^{(l)}, \mu_{2}^{(l)}, \ldots, \mu_{l}^{(l)}, \sigma^{(l)}\}\), and then the auditor verifies the correctness of $\rho^{(l)}$. First, the auditor computes the challenged indices and coefficients. Second, with the corresponding hash values stored in his local $T$, the auditor computes the value

$$h^{(l)} = \prod_{\eta=1}^{l} (h_{\eta})^{a_{\eta}} \in G.$$  \hspace{1cm} (37)

Third, the auditor checks the following equation to verify the proof:

$$e(\sigma^{(l)}, g) = e(h^{(l)} \cdot \prod_{z=1}^{s} g_{z}^{\mu_{z}^{(l)}}, y).$$  \hspace{1cm} (38)

If the equation holds, it means that the challenged data blocks are intact. Lastly, the auditor creates the following log entry that records his auditing work:

$$L^{(b)} = \{t, Q^{(b)}, h^{(b)}, \rho^{(b)}, \text{Sig}_{skc}\rho^{(b)}\},$$

and saves it in his local log file $\Lambda$. The correctness of the equality can be elaborated as follows:

$$e(\sigma^{(b)}, g) = e\left(\prod_{\eta=1}^{l} (h_{\eta})^{a_{\eta}} \cdot \prod_{z=1}^{s} g_{z}^{\mu_{z}^{(b)}}, y\right)$$

$$= e\left(\prod_{\eta=1}^{l} (h_{\eta})^{a_{\eta}} \cdot \prod_{z=1}^{s} g_{z}^{\mu_{z}^{(b)}}, y\right)$$

$$= e\left(h^{(b)} \cdot \prod_{z=1}^{s} g_{z}^{\mu_{z}^{(b)}}, y\right).$$  \hspace{1cm} (40)

5.3. AuditData Protocol. The scheme leverages the Bitcoin blockchain as a time-dependent pseudo-random source to generate periodic challenges. The auditor first inputs the time $t \in \tau$ to obtain a hash value $\text{hash}^{(b)} \in \{0, 1\}^{l_{\text{hash}}}$ of the latest block that has appeared since time $t$ in Bitcoin blockchain. Then, PRBG is invoked on the hash $(b_{i})$ to obtain long enough pseudo-random bits, which will be sequentially used by the auditor to select a pair of keys $k_{s}^{(l)}$, $k_{f}^{(l)}$. At last, the auditor generates a challenge $Q^{(l)} = \{b, k_{s}^{(l)}, k_{f}^{(l)}\}$ and sends it to the CSP, where the block $b$ corresponds to the time $t$.

Upon receiving the challenge $Q^{(l)}$, the CSP first computes the challenged indices and coefficients as follows:

$$i_{\eta} = \pi_{k_{s}^{(l)}}(\eta),$$

$$a_{\eta} = f_{k_{f}^{(l)}}(\eta) (1 \leq \eta \leq l).$$

Then, the CSP computes the proof of data possession to verify the integrity as follows:

$$\mu_{z}^{(l)} = \sum_{\eta=1}^{l} a_{\eta} m_{z_{\eta}} \in \mathbb{Z}_{q} (1 \leq z \leq s),$$

$$\sigma^{(l)} = \sum_{\eta=1}^{l} a_{\eta} m_{z_{\eta}} \in G.$$  \hspace{1cm} (36)

Finally, the CSP responses the auditor with the proof $\rho^{(l)} = \{\mu_{1}^{(l)}, \mu_{2}^{(l)}, \ldots, \mu_{l}^{(l)}, \sigma^{(l)}\}$, and then the auditor verifies the correctness of $\rho^{(l)}$. First, the auditor computes the challenged indices and coefficients. Second, with the corresponding hash values stored in his local $T$, the auditor computes the value

$$h^{(l)} = \prod_{\eta=1}^{l} (h_{\eta})^{a_{\eta}} \in G.$$  \hspace{1cm} (37)

Third, the auditor checks the following equation to verify the proof:

$$e(\sigma^{(l)}, g) = e(h^{(l)} \cdot \prod_{z=1}^{s} g_{z}^{\mu_{z}^{(l)}}, y).$$  \hspace{1cm} (38)

If the equation holds, it means that the challenged data blocks are intact. Lastly, the auditor creates the following log entry that records his auditing work:

$$L^{(b)} = \{t, Q^{(b)}, h^{(b)}, \rho^{(b)}, \text{Sig}_{skc}\rho^{(b)}\},$$

and saves it in his local log file $\Lambda$. The correctness of the equality can be elaborated as follows:

$$e(\sigma^{(b)}, g) = e\left(\prod_{\eta=1}^{l} (h_{\eta})^{a_{\eta}} \cdot \prod_{z=1}^{s} g_{z}^{\mu_{z}^{(b)}}, y\right)$$

$$= e\left(\prod_{\eta=1}^{l} (h_{\eta})^{a_{\eta}} \cdot \prod_{z=1}^{s} g_{z}^{\mu_{z}^{(b)}}, y\right)$$

$$= e\left(h^{(b)} \cdot \prod_{z=1}^{s} g_{z}^{\mu_{z}^{(b)}}, y\right).$$  \hspace{1cm} (40)

5.4. AuditLog Protocol. The client chooses a random subset $B$ of indices of Bitcoin blocks and sends it to the auditor. Once receiving $B$, the auditor finds $Q^{(b)}$, $h^{(b)}$, and $\rho^{(b)}$ from his log file $\Lambda$ for each $b \in B$ and computes

$$h_{b} = \prod_{b \in B} h^{(b)} \in G,$$

$$\sigma_{b} = \prod_{b \in B} \sigma^{(b)} \in G,$$  \hspace{1cm} (41)

$$\mu_{z}^{(b)} = \sum_{b \in B} \mu_{z}^{(b)} \in \mathbb{Z}_{q} (1 \leq z \leq s).$$

In addition, for each $b \in B$, the auditor reads $k_{s}^{(b)}$ from $Q^{(b)}$ and computes the challenged indices $i_{\eta} (1 \leq \eta \leq l)$ by invoking $\pi_{k_{s}^{(b)}}(\eta)$. After eliminating the repetitive indices, the last ordered challenge index vector is denoted by $C = (i_{1}, i_{2}, \ldots, i_{l})$. Then, the auditor runs Gen Multi Proof $(T, C)$ to obtain the corresponding multi-proof $\cup_{p}$. At last, the auditor generates the proof of appointed logs as below:

$$\rho^{(B)} = \{\mu_{1}^{(b)}, \mu_{2}^{(b)}, \ldots, \mu_{l}^{(b)}, \rho^{(l)}\},$$

and sends it to the client with $\text{Sig}_{skc}(\rho^{(B)})$.

After verifying $\text{Sig}_{skc}(\rho^{(B)})$, for each $b \in B$, the client first invokes PRBG (hash $(b_{i})$) to get $Q^{(b)}$ and reconstructs the challenged indices and coefficients $i_{\eta}$, $a_{\eta} (1 \leq \eta \leq l)$. Then, the client verifies the correctness of $\cup_{p}$ by calling VerMulti Proof $(\cup_{p}, n, h_{\text{root}}(C))$, where $C$ can be obtained by utilizing her own constructed indices for all $b \in B$. If the verification is passed, it means that all the challenged leaf nodes $w_{j}$ $(1 \leq j \leq c)$ in $\cup_{p}$ are authenticated. Finally, with all
authenticaed $h_i$, the client verifies $h(B)$ by checking the following equation:

$$h^{(B)} = \prod_{b \in B} \prod_{q=1}^{I} h_i^{a_i B}.$$  

(43)

If this verification passes, the client checks the last equation by using $sk$ and $h(B)$:

$$\sigma^{(B)} = (h^{(B)} \cdot g^{\sum_{q=1}^{I} a_i B})^x.$$  

(44)

If the above equation holds, it means that the auditor audited the past challenged data blocks appointed by $B$ honestly. The correctness of the equation can be elaborated as below:

$$\zeta^{(B)} = \prod_{b \in B} \prod_{q=1}^{I} a_i B,$$

$$= \prod_{b \in B} \prod_{q=1}^{I} \left(h_i \cdot \sum_{x=1}^{I} a_i m_{x} \right)^a_i B,$$

$$= \left(\prod_{b \in B} \left(h_i \cdot \sum_{x=1}^{I} a_i \cdot g \cdot \sum_{x=1}^{I} a_i \left(\sum_{q=1}^{I} b_i q \right)\right)^x \right),$$

$$= \left(\prod_{b \in B} \left(h^{(B)} \cdot g^{\sum_{x=1}^{I} a_i B} \right) \right)^x.$$   

(45)

In our improved protocol, the construction of data tags is

$$\sigma_i = \left(h_i \cdot \sum_{x=1}^{I} a_i m_{x} \right)^x \in G(1 \leq i \leq n),$$

(46)

instead of

$$\sigma_i = \left(h_i^{h_i} \cdot g^{\sum_{x=1}^{I} a_i m_{x}} \right)^x \in G(1 \leq i \leq n).$$

(47)

Therefore, the adversary cannot forge the authentication tags as attack I. Furthermore, malicious CSP and auditor cannot conspire to deceive the client through attack II.

6. Conclusion

In this paper, we point out that Guo et al.’s outsourced dynamic provable data possession scheme with batch update for secure cloud storage is very novel but has some design flaw, and we hope similar shortcoming can be avoided in future scheme designs to improve the security of public auditing protocols.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the National Key Research and Development Program of China (grant no. 2017YFB0802000), National Natural Science Foundation of China (grant nos. U1636114 and 61572521), Foundation of Guizhou Provincial Key Laboratory of Public Big Data (no. 2019BDKFJ008), Engineering University of PAP’s Funding for Scientific Research Innovation Team (grant no. KYTD201805), and Engineering University of PAP’s Funding for Key Researcher (no. KYGG202011).

References

[1] J. Xiong, R. Bi, M. Zhao, J. Guo, and Q. Yang, “Edge-assisted privacy-preserving raw data sharing framework for connected autonomous vehicles,” IEEE Wireless Communications, vol. 27, no. 3, pp. 24–30, 2020.
[2] Y. Tian, Z. Wang, J. Xiong, and J. Ma, “A blockchain-based secure key management scheme with trustworthiness in DWSNS,” IEEE Transactions on Industrial Informatics, vol. 16, no. 9, pp. 6193–6202, 2020.
[3] J. Xiong, X. Chen, Q. Yang, L. Chen, and Z. Yao, “A task-oriented user selection incentive mechanism in edge-aided mobile crowdsensing,” IEEE Transactions on Network Science and Engineering, vol. 7, no. 4, pp. 2347–2360, 2020.
[4] J. Xiong, X. Chen, Q. Yang, L. Chen, and Z. Yao, “A task-oriented user selection incentive mechanism in edge-aided mobile crowdsensing,” IEEE Transactions on Network Science and Engineering, vol. 7, no. 4, pp. 2347–2360, 2019.
[5] J. Xiong, M. Zhao, M. Bhuiyan, L. Chen, and Y. Tian, “An ai-enabled three-party game framework for guaranteed data privacy in mobile edge crowdsensing of iot,” IEEE Transactions on Industrial Informatics, vol. 17, no. 2, pp. 922–923, 2019.
[6] Y. Deswarte, J. J. Quisquater, and A. Saidane, Remote Integrity Checking, Springer, New York, NY, USA, 2004.
[7] D. L. G. Filho and P. S. L. M. Barreto, “Demonstrating data possession and uncheatable data transfer,” IACR, Lyon, France, Report 2006/150, 2006.
[8] F. Sebè, J. Domingo-Ferrer, A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, “Efficient remote data possession checking in critical information infrastructures,” IEEE Transactions on Knowledge and Data Engineering, vol. 20, no. 8, pp. 1034–1038, 2008.
[9] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, “Auditing to keep online storage services honest,” in Proceedings of the Hotos07: Workshop on Hot Topics in Operating Systems, DBLP, San Diego, CA, USA, May 2007.
[10] M. Naor and G. N. Rothblum, “The complexity of online memory checking,” *Journal of the ACM*, vol. 56, no. 1, pp. 1–46, 2007.

[11] G. Ateniese, R. Burns, R. Curtmola et al., “Provable data possession at untrusted stores,” in *Proceedings of the ACM Conference on Computer & Communications Security*, October 2007.

[12] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for secure cloud storage,” *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013.

[13] K. Yang and X. Jia, “An efficient and secure dynamic auditing protocol for data storage in cloud computing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717–1726, 2012.

[14] T. Hui, Y. Chen, C. C. Chang et al., “Dynamic-hash-table based public auditing for secure cloud storage,” *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 710–714, 2017.

[15] A. Juels and B. S. K. Jr, “Proofs of retrievability for large files,” in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pp. 584–597, ACM, Alexandria, VA, USA, October 2007.

[16] H. Shacham and B. Waters, *Compact Proofs of Retrievability*, Springer, Berlin, Germany, 2008.

[17] J. Zhao, C. Xu, F. Li, and W. Zhang, “Identity-based public verification with privacy-preserving for data storage security in cloud computing,” *IEICE - Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E96.A, no. 12, pp. 2709–2716, 2013.

[18] J. Zhang and Q. Dong, “Efficient id-based public auditing for the outsourced data in cloud storage,” *Information Sciences*, vol. 344, pp. 1–14, 2016.

[19] T. Xin, Y. Qi, and Y. Huang, *Fragile Watermarking Based Proofs of Retrievability for Archival Cloud Data*, Springer, New York, NY, USA, 2016.

[20] G. Wei, Z. Hua, S. Qin et al., “Outsourced dynamic provable data possession with batch update for secure cloud storage,” *Future Generation Computer Systems*, vol. 95, pp. 309–322, 2019.