Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 5127499, 10 pages
https://doi.org/10.1155/2022/5127499

Research Article

Improved Cloud Auditing Protocol and Its Application for Pandemic Data Management

Xu An Wang[1,2], **Zhengge Yi**[1], **Xiaoyuan Yang**[1], **Jindan Zhang**[3], **Yun Xie**[4], **Manman Zhang**[4] and **Guixin Wu**[5]

[1]*Engineering University of PAP, Xi'an, China*
[2]*State Key Laboratory of Public Big Data, Guizhou University, Guiyang, China*
[3]*Xianyang Vocational Technical College, Xianyang, China*
[4]*Nanjing University of Posts and Telecommunications, Nanjing, China*
[5]*Chongqing University Cancer Hospital, Chongqing, China*

Correspondence should be addressed to Xu An Wang; 1261510059@qq.com and Guixin Wu; wuguixin123456@126.com

Received 11 May 2021; Revised 23 November 2021; Accepted 13 January 2022; Published 26 February 2022

Academic Editor: Ximeng Liu

Data integrity verification mechanisms play an important role in cloud environments. Recently, a lightweight identity-based cloud storage audit scheme has been proposed; this paper points out security vulnerabilities of their OffTagGen algorithm. That is, the attackers such as malicious cloud servers can forge the tags, which can destroy data integrity. By improving the construction of OffTagGen algorithm, an improved security cloud auditing protocol is proposed in this work to better protect user's privacy. The analysis shows that the new protocol is effective and resistant to attacks.

1. Introduction

In the last two years, the COVID-19 pandemic has become a major disaster in the world. As COVID-19 has a certain fatality rate and spreads very fast, prevention and control of this virus have become a top priority worldwide. However, compared to the prevention and control regarding SARS in 2003, information related technique is being widely used in all aspects regarding prevention and control of this COVID-19 pandemic. Hence, extensive collection, processing, and investigation of personal data has become an important part of the anti-pandemic work. Given the huge amount of data collected, it is necessary to store these data in the cloud to reduce the storage burden.

As a new storage paradigm, cloud storage collects different storage devices to provide users with massive data storage. Hospitals and patients can easily access data by connecting to the cloud anytime, anywhere, and through any networked device whenever needed.

The infrastructures supporting cloud storage are distributed and virtual. This brings some threats to users' data security, such as network virus propagation, unauthorized access, denial of service attacks, information leakage, data loss, as well as network infrastructure that could damage data integrity during data transmission, etc. For pandemic prevention and control, it is clear that the loss of data, such as the patient's recent whereabouts, will certainly cause enormous trouble and could even lead to further virus spread and may even favour the situation of successive waves of the new coronavirus.

Due to the threat of internal or external attacks, data stored in the cloud are easily damaged. In addition, the cloud service provider (CSP) may not notify the user of this event in consideration of its own reputation. The user has a mechanism to detect data corruption only after accessing the data [1–3]. Therefore, in order to improve the reputation of cloud storage and let users know the integrity of the hosted data in a timely manner, a mechanism is needed to verify the

data integrity in the cloud. Hence, an integrity verification mechanism is very important in the cloud environment.

1.1. Related Works. Traditional methods need to download the entire data from the cloud when verifying data integrity, which brings unacceptable communication and computing costs and greatly consumes users' resources. In order to satisfy the user's remote checking of data integrity, the cloud data remote integrity check solution should need not to download the complete data in the cloud storage environment. Thus, the following solutions have been proposed.

Ateniese et al. [4] first proposed a provable data possession (PDP) scheme, an effective technology to audit the cloud storage. In the PDP protocol, data are encoded as blocks, and the user processes the block data to generate a verifiable authenticator, and then it outsources the data blocks and authenticators to the cloud. A public verifier with sufficient resources is also called a third-party auditor (TPA) and is trusted by users to check the data integrity. TPA creates a challenge to the server by randomly selecting a small group of block indexes. The server returns a proof that proves the integrity of the challenged blocks. TPA can effectively verify the proof without downloading data block. PDP has laid the foundation for the design of cloud storage audit schemes. In recent years, many researchers have conducted extensive and in-depth explorations around PDP [5–7].

In order to obtain better efficiency and performance, several improved PDP protocols have been proposed [8, 9]. The previously proposed schemes mostly use traditional public key cryptography, so a trusted certificate authority (CA) is required to issue a certificate to bind certain user identities and their public keys. Heavy certificate management, including certificate generation, distribution, and revocation, requires a lot of computing and storage resources. As the number of users increases, certificate management becomes extremely difficult. In addition, the verifier must retrieve the certificate from the CA and then check the validity of the public key certificate, which also brings heavy calculation and communication costs to the verifier. Therefore, the certificate-based PDP protocol is very inefficient when used in actual situations.

In order to overcome this problem, researchers considered applying identity-based cryptography to the PDP protocol and therefore proposed many ID-PDP solutions. Wang et al. [10] first introduced the concept of identity-based PDP (ID-PDP), which uses user names or emails instead of public keys. Then, ID-PDP is further extended to the multicloud storage environment [11] to check the integrity of remote data. In order to improve performance, Wang et al. [12] added a proxy server to the remote data integrity check scheme. The proxy server processes data instead of users. In the scheme, incentive and unconditional anonymous ID-PDP was first proposed to protect and encourage criminal whistleblowers. Yu et al. [13] used RSA signature technology to design an ID-based integrity cloud data check protocol. The protocol supports variable size file blocks and public verification. In order to further improve security, Yu et al. [14] combined the key homomorphic encryption technology in the cryptographic cloud audit system and proposed an improved scheme with perfect data privacy protection capabilities. The ID-based privacy-preserving integrity verification of shared data over untrusted cloud scheme is proposed, which can support users to update the data in cloud and protect users' privacy in untrusted cloud servers. Li et al. [15] proposed identity-based privacy-preserving remote data integrity checking for cloud storage scheme, which uses homomorphic verifiable tags to reduce the computational complexity and uses random integer addition to mask the original data to protect the verifier from obtaining any knowledge about the data during the integrity checking process.

1.2. Contribution. Currently, using cloud storage audit protocols is regarded as an important cloud service. However, the existing audit protocols have certain shortcomings. On the one hand, most of them rely on expensive public key infrastructure (PKI), so certificate management/verification is very complicated. On the other hand, most cloud users have limited resources. Nowadays, ID-based cloud audit protocols have attracted the attention of researchers, but most of them require users with limited resources to perform expensive operations.

Recently, Rabaninejad et al. [16] proposed a lightweight identity-based provable data ownership cloud storage audit scheme, which supports privacy and traceability of user identities. They also proposed an online/offline ID-based PDP scheme [17]. However, we discovered that there are security flaws in the digital signature (OffTagGen) of their scheme. Attackers, such as malicious cloud servers, can destroy the privacy of user's identity privacy and damage data privacy and integrity. In order to get a more secure protocol, based on the scheme presented in [16], we propose an improved one and discuss its application in pandemic data management. The main contributions of this paper are summarized as follows:

(1) We firstly point out the insecurity of Rabaninejad et al.'s lightweight identity-based provable data ownership cloud storage audit scheme. We give two attacks to show that data tags can be easily forged.

(2) We provide an improved secure cloud audit protocol that protects user privacy. This new protocol is effective yet resistant to attacks.

(3) Finally, we show how our scheme can be applied to the pandemic data management.

1.3. Organization. The rest of this article is organized as follows. In Section 2, we describe the system framework. In Section 3, we review the cloud audit scheme proposed by Rabaninejad et al. [17]. In Sections 4 and 5, we introduce the attack. In Section 6, we provide an improved privacy protection cloud audit protocol and conduct a rough analysis of its security. In Section 7, we apply our scheme to pandemic data management. Finally, in Section 8, we conclude the work and point out some directions for future work.

2. System Framework

In this section, we first describe the system model. Then, we give the goals of the design. After that, we introduce some necessary definitions. Finally, we show the security model.

2.1. System Model. The system model includes four entities, as shown in Figure 1, which involves the key generation center (KGC), the users, the third-party auditor (TPA), and the cloud server. The functions of each entity are summarized as follows:

(1) *KGC*. Based on the user's identity, KGC generates its private key.

(2) *The Users*. When the users want to store data files remotely in the cloud, they first divide the file into several blocks, use their own private key to generate a label or tag on each data block, and then outsource (block, label) to the cloud server.

(3) *TPA*. TPA performs public audits delegated by the users.

(4) *Cloud Server*. The cloud server stores user-managed data and generates a proof verified by TPA.

As shown in Figure 1, the workflow of the four parties can be described as follows:

(1) The users generate the offline tags and store them locally.

(2) The users send their identity information to KGC.

(3) KGC uses the master key and the users' identity information to generate the users' private key and returns it to them.

(4) When users need to store data files in the cloud server, they generate online tags by using some lightweight computations based on offline tags.

(5) Users outsource the (block, online tag) pair to the cloud server.

(6) The user sends an audit request to TPA with some audit information attached.

(7) TPA sends the challenge message to the cloud server.

(8) The cloud server generates a proof based on the challenge message and sends it to TPA.

(9) TPA sends the results of the audit as the auditing report to the users.

2.2. Design Goal. The design goals are roughly as follows.

(i) *Correctness*. If TPA honestly follows the agreement, it can correctly audit the integrity of outsourced data.

(ii) *Soundness*. If an untrusted cloud server has not completely stored the outsourcing data, it cannot pass the audit.

(iii) *Public Audit*. TPA can replace the user to remotely audit the integrity of the data.

(iv) *Scalability*. TPA can simultaneously support the effective verification of multiple audit requirements.

(v) *Lightweight*. TPA provides low-cost label generation algorithms for users with limited computing resources.

2.3. Definition. The ID-PDP scheme includes the following algorithms:

(i) $Setup(1^k) \longrightarrow (param, msk)$. KGC executes this algorithm. The input is the security parameter 1^k, and the output is the master key msk and the public parameter $param$.

(ii) $Extract\ (ID, param, msk) \longrightarrow k_{ID}$. KGC receives the inputs, including msk, and identity ID and then outputs the secret key k_{ID}.

(iii) $TagGen(param\ , k_{ID}\ , F) \longrightarrow \sigma$. The data owner with the key k_{ID} executes this algorithm, inputs the parameters and the data file, and first splits F into n blocks $F = (m_1, \ldots, m_n)$. Next, the data owner generates a corresponding label σ_i for each block m_i and outsources the label $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ and the data blocks $F = (m_1, \ldots, m_n)$ to the cloud together.

(iv) $Challenge\ (param, F_{name}; ID) \longrightarrow chal$. TPA outputs a challenge on behalf of the data owner whose identity is ID to challenge the integrity of the F_{name} file. The parameters, the name F_{name} of the data file F, and the identity ID of the data owner are taken as input.

(v) $ProofGen\ (param, ID, chal, F, \sigma) \longrightarrow proof$. The cloud server executes $ProofGen$, inputs $param$, $chal$, owner's identity ID, and the file F with label σ, and outputs a certificate proving the integrity of the challenge block.

(vi) $ProofVerify\ (param, F_{name}, R, ID, chal, proof) \longrightarrow \{0, 1\}$. TPA executes $ProofVerify$ to verify the proof of challenge reported by the cloud server. The input are $param$, F_{name}, the identity ID of the data owner, and the pair $(chal, proof)$. If the verification is passed, 1 is output, otherwise 0.

2.4. Security Model. For maintaining their own reputation, cloud servers are generally unwilling to disclose data loss/damage to the verifier, so they are not completely credible in

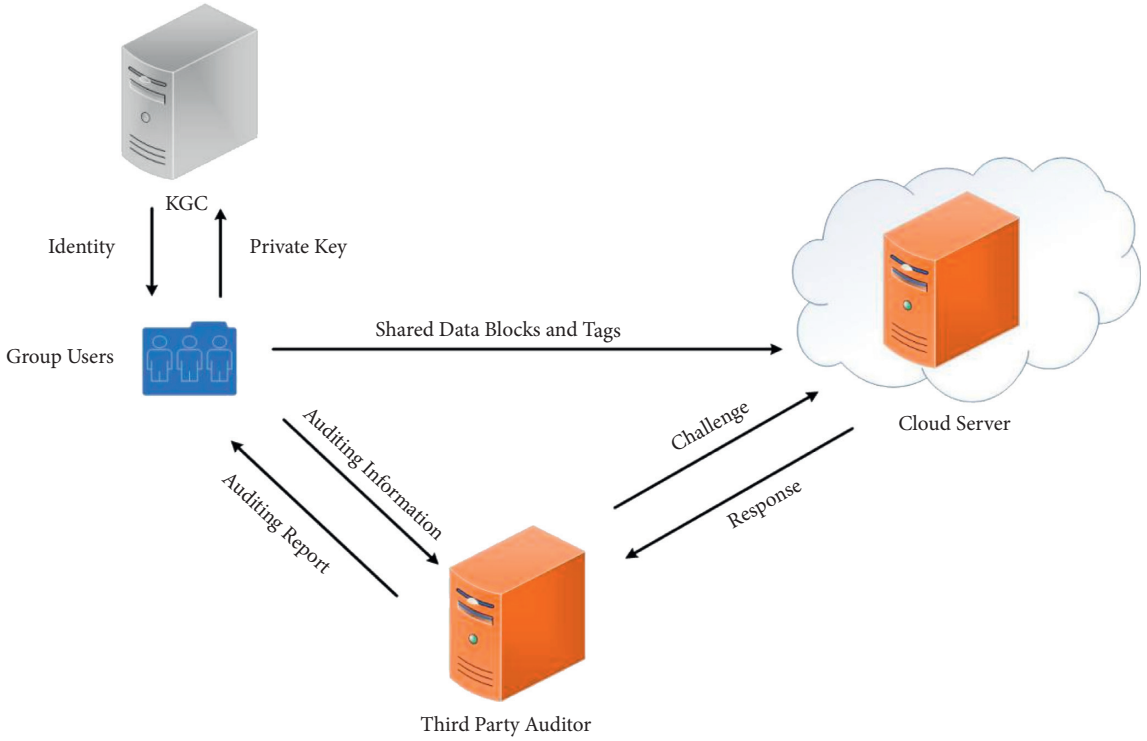

Figure 1: System model.

the PDP scheme. Here, we focus on the security model of auditing soundness of the cloud storage auditing protocol. The security model is described as follows: a game between an adversary server A and a challenger B.

(i) *Setup*. The algorithm *Setup* is run by challenger B, and the master key *msk* and public parameters *param* are obtained. Then, challenger B forwards the parameters to the server A but keeps *msk* secret.

(ii) *Query*. The adversary server A adaptively makes the following queries to the challenger B:

(a) *HashQuery*. A performs a hash function query, and B uses the hash value as a response.

(b) *ExtractQuery*. A can query any user's key through it. B obtains the key by running the *Extract* algorithm and sends the obtained result to A.

(c) *TagQuery*. A queries the tags on the input pair (ID, m). B responds to the query by running the *TagGen* algorithm and feeds the results back to A.

(iii) *Challenge*. Challenger B runs the *Challenge* algorithm on the file F of the user with ID, and B sends a challenge to A. Note that ID has never been queried from the *Extract* oracle, and all blocks of the file F have been queried from the *tag* oracle.

(iv) *ProofGen*. Adversary A executes the algorithm and computes a *proof* based on the received challenge.

(v) *ProofVerify*. If the proof is verified, A wins the game. Also, part of the proof represented by μ in the protocol is not equal to the aggregate value coming from the challenger B based on the ProofGen algorithm.

3. Review of Rabaninejad's Scheme

In this section, we will review the specific scheme of Rabaninejad et al. [17]. First, we review the concept of a bilinear map. G_1 and G_2 denote a cyclic additive group and a cyclic multiplicative group of the same prime order q, where g is the generator of G_1. The bilinear map $e: G_1 \times G_1 \longrightarrow G_2$ is a function with the following properties:

(i) *Bilinearity*. $e(aP, bQ) = e(P, Q)^{ab}$, for all $P, Q \in G_1$ and $a, b \in Z_q$.

(ii) *Non – Degeneracy*. $e(P, P) \neq 1$, where 1 denotes the identity element of G_2.

(iii) *Computability*. There exists an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in G_1$.

Rabaninejad et al. [17] proposed an online/offline ID-based PDP scheme, which consists of the following algorithms. In addition to the definition and notation in the bilinear map, two hash functions $H: \{0,1\}^* \longrightarrow G_1$ and $h: \{0,1\}^* \longrightarrow Z_q$ are used in the scheme.

(1) *Setup*. The KGC chooses a random value $\alpha \in Z_q$ as the master secret key *msk* and sets the master public key as $mpk = g^{\alpha}$. So, the system public parameters are $param = (e, q, G_1, G_2, g, mpk, h, H)$.

(2) *Extract*. The KGC uses $param = (e, q, G_1, G_2, g, mpk, h, H)$ and $mpk = \alpha$ to generate the secret key $k_{ID} = H(ID)^{\alpha}$ for user ID.

(3) $OffTagGen$. The user with identity ID chooses a secret random value $x \in Z_q$ as the trapdoor key and sets $\gamma = mpk^x$. Then, it generates an offline tag σ_i^{off} for $i \in [1, B]$ by choosing two random value $(m_i^{'}, r_i^{'})$ from Z_q as follows:

$$\sigma_i^{off} = \left(k_{ID}^x\right)^{m_t'} k_{ID}^{r_t'} = k_{ID}^{xm'+r_t'}. \tag{1}$$

At last, the offline tags $\{(m_i^t, r_i^t, \sigma_i^{\text{off}})\}_{i\in[1,B]}$ are locally stored.

(4) $OnTagGen$. The user with identity ID owns the file F with F_{name}. First, it divides F into n blocks as $F = (m_1, ..., m_n)$, where $m_i \in Z_q$. Then, it generates the online tag (r_i, σ_i) on block m_i based on an unused offline tag $(m_i^t, r_i', \sigma_i^{off})$ and uses trapdoor key as follows:

$$\begin{aligned} r_i &= r_i^t + x\left(m_i^t - m_i\right) \bmod q, \\ \sigma_i &= \sigma_i^{off}. \end{aligned} \tag{2}$$

Finally, the online $tag\{(r_i, \sigma_i)\}_{i\in[1,n]}$ together with the data blocks $F = (m_1, \ldots, m)_n$ is outsourced to the cloud server. The data owner also creates an MHT on the ordered hash value $\{h(r_i)\}_{i\in[1,n]}$ with the root node root and generates $\sigma_{root} = IDS(root)$. At the same time, the pair $(root, \sigma_{root})$ together with (γ, F_{name}), $IDS(\gamma \| F_{name})$ is sent to the server and the TPA. Here, IDS is a secure ID-based signature. When the server receives the tags $\{(r_i, \sigma_i)\}_{i\in[1,n]}$ and the data blocks $F = (m_1, \ldots, m_n)$, it first checks whether $Verify(m_i, r_i, \sigma_i, ID, mpk, \gamma) = 1$ passes for all $i \in [1, n]$. If so, it stores $\{(m_i, r_i, \sigma_i)\}_{i\in[1,n]}$ in its storage; otherwise, it outputs $\perp$. Furthermore, if σ_{root} and $IDS(\gamma \| F_{name})$ are valid signatures, the server and the TPA save the values' root, γ for the file name F_{name}.

(5) *Challenge*. In order to challenge the integrity of the file F_{name} which is owned by the user with identity ID, the TPA runs the following process:

(a) Choose a random subset $J \subset [1, n]$ as the block indices to be challenged in the auditing process, and for each $j \subset J$, choose a random value $y_j \in Z_q$.

(b) Send the challenge $chal = (F_{name}, \{(j, y_j)\}_{j\in J})$ to the server.

(6) $ProofGen$. The server generates an auditing proof according to the received challenge $chal = (F_{name}, \{(j, y_j)\}_{j \subset J})$, through the following procedure:

(a) Computes a combination of the challenged blocks as $\mu^t = \sum_{j\in J} y_i m_j$ and sets $\mu = H(ID)^{\mu}$.

(b) Aggregates the tags as $\sigma = \sum_{j\in J} \sigma_j^{y_j}$.

(c) Sends back $(\mu, \sigma, \{r_j.\Delta_j\}_{j\in J})$ as the auditing proof to the TPA. Here, r_j is the first term in tag of block m_j and Δ_j is the corresponding AAI in MHT.

(7) *ProofVerify*. When TPA receives the proof $(\mu, \sigma, \{r_j.\Delta_j\}_{j\in J})$, it first computes $root'$ from $\{h(r_j) j \in J\}$ and the corresponding $AAI\{\Delta_j\} j \in J$. If $root' = root$, it then computes $R = \sum_{j\in J} y_j r_j$ and checks equation (3). which indicates that the cloud storage is good or not

$$e(\sigma, g) \stackrel{?}{=} e(\mu, \gamma) \cdot e(H(ID), mpk)^R. \tag{3}$$

4. Attack I on the OffTagGen Algorithm

The attack I is as follows:

(1) The adversary (which can be the malicious cloud server) can obtain many block-signature pairs, such as $(M_1, \sigma_1), (M_2, \sigma_2), \ldots, (M_n, \sigma_n)$, and the following holds:

$$\begin{aligned} \sigma_1 &= k_{ID}^{xm_1'+r_1'} = k_{ID}^{xm_1+r_1}, \\ \sigma_2 &= k_{ID}^{xm_2'+r_2'} = k_{ID}^{xm_2+r_2}, \\ &\cdots \\ \sigma_n &= k_{ID}^{xm_n'+r_n'} = k_{ID}^{xm_n+r_n}. \end{aligned} \tag{4}$$

(2) Let $k_{ID}^x = A, k_{ID} = B$, and $r_1, r_2, ..., r_n$ be all known to the adversary; thus, the above equations can be rewritten as follows:

$$\begin{aligned} \sigma_1 &= k_{ID}^{xm_1+r_1} = A^{m_1} B^{r_1}, \\ \sigma_2 &= k_{ID}^{xm_2+r_2} = A^{m_2} B^{r_2}, \\ &\cdots \\ \sigma_n &= k_{ID}^{xm_n+r_n} = A^{m_n} B^{r_n}. \end{aligned} \tag{5}$$

(3) With these equations, the adversary can compute A and B. We must point out that actually two equations are enough to compute A and B. Concretely, the adversary first computes

$$\begin{aligned} \sigma_1^{m_2} &= A^{m_1 m_2} B^{r_1 m_2}, \\ \sigma_2^{m_1} &= A^{m_2 m_1} B^{r_2 m_1}, \\ \sigma_1^{r_2} &= A^{m_1 r_2} B^{r_1 r_2}, \\ \sigma_2^{r_1} &= A^{m_2 r_1} B^{r_2 r_1}, \end{aligned} \tag{6}$$

and then computes

$$\begin{aligned} \frac{\sigma_1^{m_2}}{\sigma_2^{m_1}} &= \frac{A^{m_1 m_2} B^{r_1 m_2}}{A^{m_2 m_1} B^{r_2 m_1}} = \frac{B^{r_1 m_2}}{B^{r_2 m_1}} = B^{r_1 m_2 - r_2 m_1}, \\ \frac{\sigma_1^{r_2}}{\sigma_2^{r_1}} &= \frac{A^{m_1 r_2} B^{r_1 r_2}}{A^{m_2 r_1} B^{r_2 r_1}} = \frac{A^{m_1 r_2}}{A^{m_2 r_1}} = A^{m_1 r_2 - m_2 r_1}. \end{aligned} \tag{7}$$

(4) Let $C = \sigma_1^{m_2}/\sigma_2^{m_1}$ and $D = \sigma_1^{r_2}/\sigma_2^{r_1}$:

$$C = \frac{\sigma_1^{m_2}}{\sigma_2^{m_1}} = B^{r_1 m_2 - r_2 m_1},$$
$$D = \frac{\sigma_1^{r_2}}{\sigma_2^{r_1}} = A^{m_1 r_2 - m_2 r_1}. \tag{8}$$

For the exponential prime modular, q is publicly known to all; thus, the adversary can compute A and B as follows:

$$C^{1/r_1 m_2 - r_2 m_1} = B,$$
$$D^{1/m_1 r_2 - m_2 r_1} = A. \tag{9}$$

(5) With A and B, the adversary can forge any offline and online tags; concretely, the offline tags and online tags are generated as follows:

(a) $OffTagGen$. The adversary forges offline tags σ^{off} for $i \in [1, B]$, and by choosing random values $(\overline{m}_i', \overline{r}_i')$ from Z_q, it computes

$$\sigma_i^{\text{off}} = A^{\overline{m}_i'} B^{\overline{r}_i'} = \left(k_{ID}^x\right)^{\overline{m}_i'} k_{ID}^{\overline{r}_i'} = k_{ID}^{x\overline{m}_i' \overline{r}_i'}. \tag{10}$$

Because the adversary knows A and B, it can compute σ^{off}. At last, the adversary locally stores the offline tags $\{(\overline{m}_i', \overline{r}_i', \sigma_i^{\text{off}})\}_{i \in [1,B]}$.

(b) $OnTagGen$. For the file F with F_{name} where $= (m_1, \ldots, m_n)$, assume the adversary wants to modify $F = (m_1, \ldots, m_n)$ to be $F = (\overline{m}_1, \overline{m}_2, \ldots, \overline{m}_n)$ and then forge the tags. It generates the online tag $(\overline{r}_i, \overline{\sigma}_i)$ on block $\overline{m}_i$ as follows:

$$\overline{r}_i = r_i \bmod q,$$
$$\overline{\sigma}_i = A^{\overline{m}_i} B^{r_i} = \left(k_{ID}^x\right)^{\overline{m}_i} k_{ID}^{r_i} = k_{ID}^{x\overline{m}_i + r_i}. \tag{11}$$

Finally, the online tags $\{(\overline{r}_i, \overline{\sigma}_i)\}_{i \in [1,n]}$ together with the data blocks $F = (\overline{m}_1, \overline{m}_2, \ldots, \overline{m}_n)$ are outsourced to the cloud server. At the same time, the original pair $(root, \sigma_{root})$ together with (γ, F_{name}), $I\ DS(\gamma || F_{name})$ is also sent to the server and the TPA. Here, $I\ DS$ is a secure ID-based signature.

We can check that the forged tag $\overline{\sigma}_i$ a valid one because the below equation holds:

$$\begin{aligned} e(\overline{\sigma}_i, g) &= e\left(k_{ID}^{x\overline{m}_i + r_i}, g\right) \\ &= e\left(\left(H_{ID}\right)^{\alpha x \overline{m}_i} \cdot \left(H_{ID}\right)^{\alpha r_i}, g\right) \\ &= e\left(\left(H_{ID}\right)^{\alpha x \overline{m}_i}, \gamma\right) \cdot e\left(\left(H_{ID}\right)^{r_i}, mpk\right). \end{aligned} \tag{12}$$

Thus, OffTagGen algorithm is not secure. Even with two block-signature pairs, anyone can first modify the contents of the blocks and then forge the offline and online tags correspondingly.

5. Attack II on the Cloud Auditing Protocol

In our attack II, we show that the adversary (which can be the malicious cloud server) can forge proof while it can even delete all the outsourced data blocks. Concretely, the attack is the following:

(1) The first four steps of the attack are the same as attack I. The malicious cloud server can get $= k_{ID}^x$, $B = k_{ID}$ after these steps. The below steps follow the framework of the cloud auditing protocol

(2) When the cloud server receives the tags $(r_i, \sigma_i)_{i \in [1,n]}$ and the data blocks $F = (m_1, m_2, \ldots, m_n)$ outsourced by the data owner, it first checks whether $Verify(m_i, r_i, \sigma_i, I\ D, mpk, \gamma) = 1$ passes for all $i \in [1, n]$. If so, it stores $\{(m_i, r_i, \sigma_i)\}_{i \in [1,n]}$ in its storage; otherwise, it outputs $\perp$. Furthermore, if σ_{root} and $I\ DS(\gamma || F_{name})$ are valid signatures, the server and the TPA save the values' root, γ for the file name F_{name}.

(3) $Challenge$. In order to challenge the integrity of the file F_name which is owned by the user with identity ID, the TPA runs the following process:

(a) Choose a random c−element subset $J \subset [1, n]$ as the block indices to be challenged in the auditing process, and for each $j \subset J$, choose a random value $y_i \in Z_q$.

(b) Send the challenge $chal = (F_{name}, \{(j, y_j)\}_{j \in J})$ to the server.

(4) $ProofGen$. Here we show that the malicious cloud server can even delete all the outsourced data blocks but still has the ability to return the correct auditing proof to the TPA. Note here that the malicious cloud server still stores all the tags $\{(r_i, \sigma_i)\}_{i \in [1,n]}$. Concretely, the server generates an auditing proof according to the received challenge $chal = (F_{name}, \{(j, y_j)\}_{j \subset J})$, through the following procedure:

(a) First, it randomly chooses $\widehat{m}_j \in Z_q\ (j \in J)$ and computes a combination of the challenged blocks as $\mu' = \sum_{j \in J} y_j \widehat{m}_j$ and sets $\mu = H(ID)^{\mu'}$.

(b) For any $\widehat{m}_j \in Z_q\ (j \in J)$, the malicious cloud server computes the forged tags as $\widehat{\sigma}_j = A^{\widehat{m}_j} B^{r_j} = \left(k_{ID}^x\right)^{\widehat{m}_j} k_{ID}^{r_j} = k_{ID}^{x\widehat{m}_j + r_j}$ and aggregates the tags as $\sigma = \prod_{j \in J} \widehat{\sigma}_j^{y_i}$.

(c) It sends back $(\mu, \sigma, \{r_j, \Delta_j\}_{j \subset J})$ as the auditing proof to the TPA. Here, r_j is the first term in the original tags of corresponding to the deleted block m_j and Δ_j is the corresponding AAI in MHT.

(5) $ProofVerify$. When TPA receives the proof $(\mu, \sigma, \{r_j, \Delta_j\}_{j \subset J})$, it first computes $root'$ from $\{h(r_j)_{j \in J}\}$ and the corresponding $AAI\{\Delta_j\}_{j \in J}$. If $root = root'$, it then computes $R = \sum_{j \in J} y_j r_j$ and

checks equation (13). If the equation holds, the TPA outputs 1, which means that the verification passes; otherwise, it outputs 0.

$$e(\sigma, g) \stackrel{?}{=} e(\mu, \gamma) \cdot e(H(ID), mpk)^R. \tag{13}$$

Here, we can verify that the forged proof, is a valid one because the below equation holds:

$$\begin{aligned} e(\sigma, g) &= e\left(\prod_{j\in J} \widehat{\sigma}_j^{y_j}, g\right) \\ &= e\left(\prod_{j\in J} \left((k_{ID}^x)^{\widehat{m}_j} \right)^{y_j}, g\right) e\left(\left(k_{ID}^{r_j}\right)^{y_j}, g\right) \\ &= e\left(\prod_{j\in J} \left((H_{ID}^{x\alpha})^{\widehat{m}_j} \right)^{y_j}, g\right) e\left(\left((H_{ID}^{\alpha})^{r_j} \right)^{y_j}, g\right) \\ &= e\left(\prod_{j\in J} \left((H_{ID})^{\widehat{m}_j} \right)^{y_j}, g^{x\alpha}\right) e\left(\left(H_{ID}^{r_j}\right)^{y_j}, g^{\alpha}\right) \\ &= e\left(\prod_{j\in J} (H_{ID})^{y_j \widehat{m}_j}, g^{x\alpha}\right) e\left(H_{ID}^{r_j y_j}, g^{\alpha}\right) \\ &= e\left(\prod_{j\in J} (H_{ID})^{y_j \widehat{m}_j}, g^{x\alpha}\right) e\left(H_{ID}, g^{\alpha}\right)^{r_j y_j} \\ &= e\left((H_{ID})^{\sum_{j\in J} y_j \widehat{m}}, g^{x\alpha}\right) e\left(H_{ID}, g^{\alpha}\right)^{\sum_{j\in J} r_j y_j} \\ &= e(\mu, \gamma) \cdot e(H(ID), mpk)^R. \end{aligned} \tag{14}$$

6. Our Improved Cloud Auditing Protocol

(1) *Setup*. The KGC chooses a random value $\alpha \in Z_q$ as the master secret key msk and sets the master public key as $mpk = g^{\alpha}$. So, the system public parameters are $param = (e, q, G_1, G_2, g, mpk, h, H)$.

(2) *Extract*. The KGC uses $param = (e, q, G_1, G_2, g, mpk, h, H)$ and $mpk = \alpha$ to generate the secret key $k_{ID} = H(ID)^{\alpha}$ for user ID.

(3) *OffTagGen*. The user with identity ID owns the file F with F_{name}, chooses a secret random value $x \in Z_q$ as the trapdoor key, and sets $\gamma = mpk^x$, $\gamma' = g^x$. Then, it generates an offline tag θ_i^{off} for $i \in [1, B]$, by choosing two random values (m_i', r_i') from Z_q as follows:

$$\begin{aligned} \sigma_{\mathrm{i}}^{off} &= \left(k_{ID}^x\right)^{m_i'} k_{ID}^{r_{\mathrm{i}}'} H\left(F_{name} \| i\right)^x \\ &= \left(k_{ID}\right)^{m_{\mathrm{i}}' x + r_{\mathrm{i}}'} H\left(F_{name} \| i\right)^x. \end{aligned} \tag{15}$$

At last, it locally stores the offline tags, $\left\{(m_i', r_i', \sigma_i^{off})\right\}_{i\in[1,B]}$.

(4) *OnTagGen*. First, it divides F into n blocks as $F = (m_1, \ldots, m_n)$, where $m_i \in Z_q$. Then, it generates the online tag (r_i, σ_i) on block m_i based on an unused offline tag $(m_i', r_i', \sigma_i^{\mathrm{off}})$ and uses trapdoor key as follows:

$$\begin{aligned} r_i &= r_i' + x\left(m_i' - m_i\right) \bmod q, \\ \sigma_i &= \sigma_i^{off}. \end{aligned} \tag{16}$$

Finally, the online tags $\{(r_i, \sigma_i)\}_{i\in[1,n]}$ together with the data blocks $F = (m_1, \ldots, m_n)$ are outsourced to the cloud server. The data owner also creates an MHT on the ordered hash value $\{h(r_i)\}_{i\in[1,n]}$ with the root node $root$ and generates $\sigma_{root} = IDS(root)$. At the same time, the pair $(root, \sigma_{root})$ together with $(\gamma, F_{name}), IDS(\gamma \| F_{name})$ is sent to the server and the TPA. Here, IDS is a secure ID-based signature.

When the server receives the tags $\{(r_i, \sigma_i)\}_{i\in[1,n]}$ and the data blocks $F = (m_1, \ldots, m_n)$, it first checks whether $Verify(m_i, r_i, \sigma_i, ID, mpk, \gamma) = 1$ passes for all $i \in [1, n]$. If so, it stores $\{(m_i, r_i, \sigma_i)\}_{i\in[1,n]}$ in its storage; otherwise, it outputs $\perp$. Furthermore, if σ_{root} and $IDS(\gamma, F_{name})$ are valid signatures, the server and the TPA save the values' $root$, γ for the file name F_{name}.

(5) *Challenge*. In order to challenge the integrity of the file F_{name} which is owned by the user with identity ID, the TPA runs the following process:

(a) Choose a random c−element subset $J \subset [1, n]$ as the block indices to be challenged in the auditing process, and for each $j \subset J$, choose a random value $y_j \in Z_q$.

(b) Send the challenge $chal = (F_{name}, \{(j, y_j)\}_{j\in J})$ to the server.

(6) *ProofGen*. The server generates an auditing proof according to the received challenge $chal = (F_{name}, \{(j, y_j)\}_{j\subset J})$, through the following procedure:

(a) Computes a combination of the challenged blocks as $\mu^t = \sum_{j\in J} y_j m_j$ and sets $\mu = H(ID)^{\mu'}$.

(b) Aggregates the tags as $\sigma = \sum_{j\in J} \sigma_j^{y_j}$.

(c) Sends back $(\mu, \sigma, \{r_j.\Delta_j\}_{j\in J})$ as the auditing proof to the TPA. Here, r_j is the first term in tag of block m_j and Δ_j is the corresponding AAI in MHT.

(7) *ProofVerify*. When TPA receives the proof $(\mu, \sigma, \{r_j.\Delta_j\}_{j\in J})$, it first computes $root'$ from $\{h(r_j)_{j\in J}\}$ and the corresponding $AAI\{\Delta_j\}_{j\in J}$. If $root' = root$, it then computes $R = \sum_{j\in J} y_j r_j$ and checks equation (17).

$$e(\sigma, g) = e\left(\sum_{j\in J} (H(F_{name} \| i))^{y_j}, \gamma'\right) e(\mu, \gamma) e(H(ID), mpk)^R. \tag{17}$$

7. Security Analysis

In this section, we first prove the correctness of our improved scheme. Then, we prove that the audit proof in our proposed scheme cannot be forged, which proves that our proposed scheme can resist attacks I and II.

(1) *Correctness*. The correctness of verification equation (17) is proved below:

$$\begin{aligned} e(\sigma, g) &= e\left(\sum_{j\in J}\left(\left(k_{ID}^{x}\right)^{m_{i}\prime} k_{ID}^{r_{i}^{\prime}} H\left(F_{name}||i\right)^{x}\right)^{y_j}, g\right) \\ &= e\left(\sum_{j\in J} H\left(\left(F_{name}||i\right)^{x}\right)^{y_j}, g\right) \cdot e\left(\sum_{j\in J}\left(\left(k_{ID}^{x}\right)^{m_{i}\prime}\right)^{y_j}, g\right) \cdot e\left(\sum_{j\in J}\left(k_{ID}^{r_{i}^{\prime}}\right)^{y_j}, g\right) \\ &= e\left(\sum_{j\in J}\left(H\left(F_{name}||i\right)\right)^{y_j}, \gamma^{\prime}\right) \cdot e\left(\sum_{j\in J}\left(H_{ID}^{m_{i}^{\prime}}\right)^{y_j}, g^{x\alpha}\right) \cdot e\left(\sum_{j\in J}\left(k_{ID}^{r_{i}^{\prime}}\right)^{y_j}, g^{x}\right) \\ &= e\left(\sum_{j\in J}\left(H\left(F_{name}||i\right)\right)^{y_j}, \gamma^{\prime}\right) \cdot e(\mu, \gamma) \cdot e(H(ID), mpk)^{R}. \end{aligned} \tag{18}$$

(2) *Soundness*. In our improved scheme, a malicious CSP cannot forge a correct audit proof by using our attacks.

Proof

(i) *Setup*. The challenger B chooses a random value $\alpha \in Z_q$ as the master secret key msk and sets the master public key as $mpk = g^{\alpha}$. Then, challenger B forwards the parameters to the server A but keeps msk secret.

(ii) *Query*. The adversary server A adaptively makes the following queries to the challenger B:

(a) *Hash Query*. A queries hash value based on ID_i, and B chooses random $y_i \in Z_q$ and then outputs $H_i = g^{y_i}$ as a response.

(b) *Extract Query*. A can query any user's key through its ID. By running the *Extract* algorithm, B generates $K_{ID_i} = (g^{y_i})^{\alpha}$ and sends the obtained result to A.

(c) *Tag Query*. A queries the tags on the input pair (PID, m). B responds to the query by running the *TagGen* algorithm. B chooses random $r_i, r_j \in \mathbb{Z}_q$ and generates $\sigma_{ij} = ((g^{x})^{r_i y_i \alpha})^{m_j} \cdot (g^{y_i \alpha})^{r_j} \cdot H(F_{name}||i)^{x}$. Finally, B outputs (r_j, σ_{ij}) and $(\gamma_i, I\,DS\,(\gamma_i))$ and sends them to A.

(iii) *Challenge*. Challenger B runs the *Challenge* algorithm on the file F of the user with PID and B sends a challenge $chal^{*} = (F_{\text{name}}^{*}, \{(j, y_j)\}_{j\in J^{*}})$ to A.

(iv) *ProofGen*. Adversary A executes the algorithm and computes a *Proof* $P^{*} = (\mu^{*}, \sigma, \{r_j^{*}, \Delta_j^{*}\}_{j\in J^{*}})$ based on the received challenge $chal^{*}$.

(v) *ProofVerify*. If the proof is verified, A wins the game. Also, part of the proof represented by μ^{*} in the protocol is not equal to the aggregate value μ coming from the challenger B based on the *ProofVerify* algorithm.

In the original scheme, the adversary A can compute $\mu = H(ID_i)^{\mu^{\prime}}$; therefore, he can set $\mu = \mu^{*}$.

However, in our improved scheme, the authentication tag is calculated as equation (15):

$$\sigma_i^{off} = \left(k_{ID}\right)^{m_i^{\prime}x + r_i^{\prime}} H\left(F_{name}||i\right)^{x}. \tag{19}$$

Before the malicious adversary forges an off tag, he needs to know the value of $H(F_{name}||i)^{x}$. However, for $i \in [1, n]$, the value of $H(F_{name}||i)^{x}$ is different. Even if he can get the hash value of the data $H(F_{name}||i)$, calculating $H(F_{name}||i)^{x}$ is as hard as solving the DL problem in G_1, which is computationally infeasible. If p^{*} passes the verification, we can get $(xr_i^{*})\mu^{\prime} + R = (xr_i^{*})\mu^{\prime *} + R^{*}$, where $\mu \neq \mu^{*}$. B outputs $((\mu/\mu^{*})^{\Delta R^{-1}})^{r_i^{*} y_i^{*-1}}$ which is a solution to the InvCDH problem and is not feasible. Therefore, the malicious CSP cannot forge a correct audit proof to pass the proof verify of TPA.

(3) *Data Privacy against TPA*. While providing the integrity audit service to the user, TPA cannot obtain any information about the content of the user's data from the information provided by the user or from the auditing process.

Proof. On the one hand, TPA received information from user before performing the auditing work are (root, σ_{root}) and (γ, F_{name}). The user data cannot be accessed from root due to the one-way nature of the hash function. Meanwhile, $\gamma = mpk^{x}$, and TPA cannot get user's data from it.

On the other hand, in the auditing process, the TPA gains $\mu = H(ID_i)^{\mu^{\prime}}$. However, given $H(ID_i) \in G_1$ and $\mu \in G_1$, computing $\mu^{\prime} = \sum_{j\in J} y_j m_j$ is solution to the DL problem.

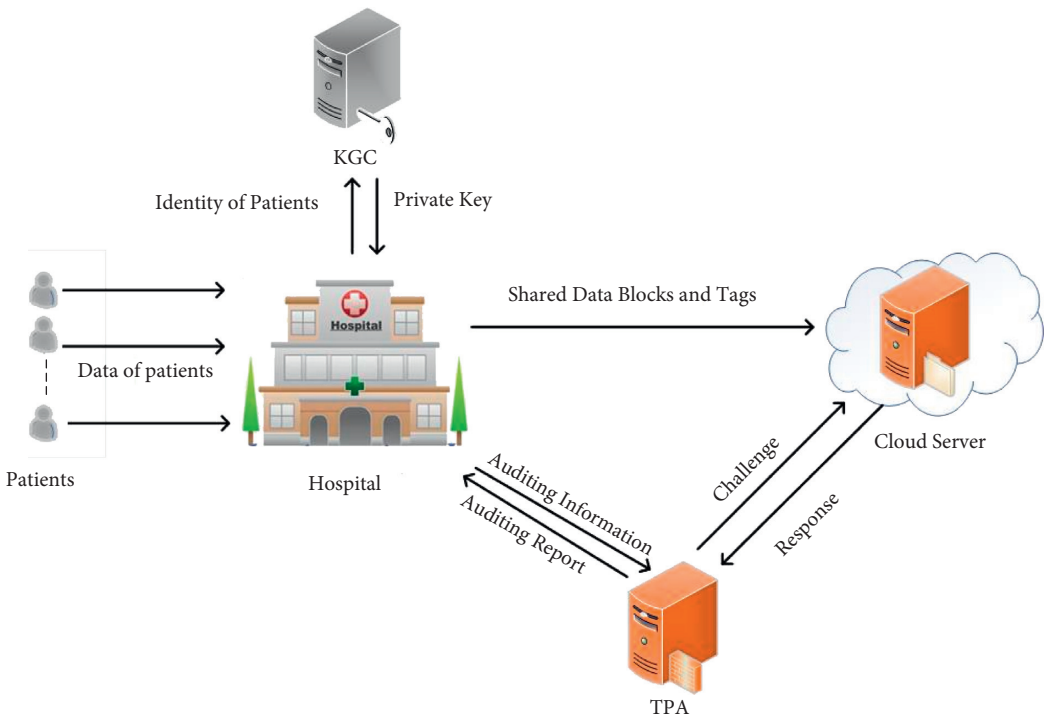

Figure 2: Pandemic data management in a hospital-based application.

Therefore, our improved scheme can preserve the user's data privacy.

8. Application of Our Scheme

As an application, we consider the hospital's data management in cloud setting as an example to demonstrate the effectiveness of our scheme to the actual pandemic data management. During the prevention and control of the epidemic, many medical data of patients need to be recorded, including the patients' nucleic acid testing results, the doctor's diagnosis, and if diagnosed as COVID-19, the recent whereabouts of the patients. In fact, the amount of these data is very huge; if the hospital stores them locally, it will consume a lot of storage resources, so it is better to outsource these huge data to cloud servers for storage. However, the cloud server is not completely reliable. Many important data may be lost due to various unexpected accidents. This will cause the treatment of many patients to be delayed due to the loss of data. Furthermore, the loss of some key data of diagnosed patients may lead to inadequate control of the epidemic, which may lead to the spread of the epidemic. If there is no data integrity audit mechanism, we cannot know whether the data are completely stored. So, an integrity audit mechanism is used to ensure the integrity of cloud data. At the same time, the public key is usually used to generate the authentication information of patient data. Due to the large number of patients and the continuous increase in the number of patients, the key management is a big problem. Our scheme uses identity-based way to generate public and private keys and directly uses the user's identity information to generate public keys, which effectively avoids the difficulty of key management.

Figure 2 illustrates the system model. It includes five entities. The five entities are patients, hospital (which we termed as HSP), KGC, CSP, and TPA. Because the KGC generates the private keys for the patients, it is necessary for the hospital to select a trusted key generation server as the KGC. As the cloud server needs to store a large amount of medical data, servers with strong storage capacity are selected as the cloud storage servers. Since the TPA requires a lot of audit work, a server with powerful computing power is used as the TPA. There are three steps in this model, and they are key generation, data upload, and integrity verification. The concrete implementation is as follows:

Step 1 (key generation): first, the KGC in the HSP sets the parameters and calculates the master key in the Setup stage. When the patient comes to the hospital, the hospital generates the patient's ID based on the patient's identity information and sends the ID to KGC. Then, the KGC computes the identity-based private key for the patient according to the Extract algorithm.

Step 2 (data upload): after the hospital collected the patients' data, these data need to be uploaded to the HSP's storage server. Firstly, the hospital computes the corresponding tag for the patient based on the corresponding collected data. Then, it uploads the data blocks and the corresponding tags to the HSP's storage server. According to the policy, the tags and auxiliary information based on patient's identity are transmitted to TPA, ensuring that the TPA can implement the auditing for the data stored in the cloud server.

Step 3 (integrity verification): in order to guarantee the integrity of the data, HSP needs to check it regularly. First of all, the hospital or patients make a request for

integrity verification to TPA, which in turn uses the challenge-response auditing protocol to verify the integrity of the data stored in the cloud, as requested. If the verification is successful, the patient's data are considered to be good stored in the cloud server. Otherwise, the CSP does not store the patient's data well, and other patient's data blocks may also be lost. At this time, other important data need to be checked also, and if the data are lost, remedial measures such as backup and recovery of the lost data are needed in time.

9. Conclusions

In this paper, we review a lightweight ID-based verifiable data ownership cloud storage audit scheme proposed by Rabaninejad et al. [17]. Then, we point out the security vulnerabilities in the OffTagGen and OnTagGen part of the scheme and further demonstrate the insecurity of the original protocol by showing the attack. In order to protect the integrity of users' data, an improved secure cloud audit protocol is proposed. The security analysis shows that the new protocol is secure.

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 study was supported by the National Natural Science Foundation of China under grant nos. U1636114, 62102452, and 62172436, Open Project from Guizhou Provincial Key Laboratory of Public Big Data under grant no. 2019BDKFJJ008, Engineering University of PAP's Funding for Scientific Research Innovation Team under grant no. KYTD201805, and Engineering University of PAP's Funding for Key Researcher under grant no. KYGG202011.

References

[1] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69–73, 2012.

[2] M. Khorshed, A. B. M. Ali, and S. Wasimi, "A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing," *Future Generation Computer Systems*, vol. 28, pp. 833–851, 2012.

[3] J. Gudeme, S. Pasupuleti, and R. Kandukuri, "Review of remote data integrity auditing schemes in cloud computing: taxonomy, analysis, and open issues," *International Journal of Cloud Computing*, vol. 8, p. 20, 2019.

[4] G. Ateniese, R. Burns, R. Curtmola et al., "Provable data possession at untrusted stores," pp. 598–609, 2007.

[5] M. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," *IACR Cryptology ePrint Archive*, vol. 2008, p. 186, 2008.

[6] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 90–107, Melbourne, VIC, Australia, December 2008.

[7] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: multiple-replica provable data possession," in *Proceedings of the 28th International Conference on Distributed Computing Systems*, Beijing, China, June 2008.

[8] Z. Hao, S. Zhong, and N. Yu, "A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 1432–1437, 2011.

[9] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proceedings of the 2010 Proceedings IEEE INFOCOM*, pp. 525–533, San Diego, CA, USA, March 2010.

[10] H. Wang, Q. Wu, B. Qin, and J. Domingo-Ferrer, "Identity-based remote data possession checking in public clouds," *Information Security*, vol. 8, pp. 114–121, 2014.

[11] H. Wang, "Identity-based distributed provable data possession in multicloud storage," *IEEE Transactions on Services Computing*, vol. 8, pp. 328–340, 2015.

[12] H. Wang, D. He, and S. Tang, "Identity-based proxy-oriented data uploading and remote data integrity checking in public cloud," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1165–1176, 2016.

[13] Y. Yu, M. Au, G. Ateniese et al., "Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 767–778, 2017.

[14] Y. Yu, L. Xue, M. H. Au et al., "Cloud data integrity checking with an identity-based auditing mechanism from RSA," *Future Generation Computer Systems*, vol. 62, 2016.

[15] 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, 2020.

[16] R. Rabaninejad, S. M. Sedaghat, M. Ahmadian Attari, and M. R. Aref, "An ID-based privacy-preserving integrity verification of shared data over untrusted cloud," *Computer Society of Iran*, vol. 2020, pp. 1–6, 2020.

[17] R. Rabaninejad, M. R. Asaar, M. A. Attari, and M. R. Aref, "An identity-based online/offline secure cloud storage auditing scheme," *Cluster Computing*, vol. 23, pp. 1455–1468, 2020.