New Publicly Verifiable Data Deletion Supporting Efficient Tracking for Cloud Storage

Changsong Yang[1], Xiaoling Tao[1], and Qiyu Chen[2]
(Corresponding author: Changsong Yang and Xiaoling Tao)
School of Computer Science and Information Security, Guilin University of Electronic Technology[1]
No. 1 Jinji Road, Guilin, China, 541004
College of Information and Computer Engineering, Northeast Forestry University[2]
No. 26 HeXing Road, Harbin, Heilongjiang, China
(Email: csyang02@163.com and txl@guet.deu.cn)
(Received Apr. 26, 2019; Revised and Accepted Nov. 16, 2019; First Online Feb. 15, 2020)

Abstract

With the rapid development of cloud storage, an increasing number of data owners are willing to outsource their data to cloud server to greatly reduce local storage overhead. However, in cloud storage, the ownership of the outsourced data is disconnected from the management, which makes the outsourced data deletion become a crucial security challenge: the cloud server might reserve the data maliciously for economic interests, and return wrong deletion results to cheat the data owners. To solve this problem, we design a novel outsourced data deletion scheme. If the cloud server does not execute deletion command honestly, the data owner can detect the dishonest data reservation by checking the returned deletion evidence. Additionally, we adopt Merkle hash tree to achieve public verifiability in outsourced data deletion without requiring any trusted third party. Meanwhile, the proposed scheme is able to achieve efficient data leakage source tracking to prevent the data owner and the cloud server from slandering each other. Finally, we prove that our scheme can satisfy the desired security requirements.

Keywords: Cloud Storage; Data Deletion; Efficient Tracking; Merkle Hash Tree; Public Verifiability

1 Introduction

Cloud computing, a newly-developing and promising computing paradigm, can connect large-scale computing resources, network resources and storage resources together through the Internet [7]. Thanks to the rapid development of computer software and hardware technology, cloud computing can utilize its plenty of resources to provide many attractive services, for instance, data storage and sharing service, outsourcing service, verifiable databases service, and so on. These services have been widely applied by the public, especially for cloud storage service. The cloud storage service provider can offer on-demand data storage service to the tenants [9]. By employing the high-quality cloud storage service, all the resource-constraint data owners could upload their personal data to remote cloud server for saving heavy local storage overhead. Because of the attractive advantages, an increasing number of data owners, including individuals and corporations prefer to embrace cloud storage service.

Despite a number of advantages, cloud storage service inescapably suffers from a few novel security challenges [3,6]. First of all, the outsourced file might contain some data owner's privacy information, which should be kept secret. Therefore, data confidentiality has become a particularly austere security challenge for cloud storage. Generally speaking, the traditional encryption technique can be seen as a solution to this issue. However, it can only offer a partial solution because it is very difficult to execute significative operations over the ciphertext. The fully homomorphic encryption algorithm seems a potential solution, but the existing protocols are not efficient and practical. Secondly, both the data owner and the cloud server may be dishonest. Both of them might expose the data maliciously to slander each other. Therefore, how to precisely trace the data leakage source is a challenge which requires to be solved. Last but not least, the data owner cannot execute any operation over the outsourced data directly because he will lose the direct control over the data. All the operations over the outsourced data, such as data deletion operation, might be executed by the cloud server. However, the cloud server might not remove the data sincerely for economic interests. Hence, how to securely remove outsoutced data is also a security threat.

Although plenty of solutions have been proposed to realize data deletion, there are still some security challenges in processing the outsourced data deletion. Firstly, plenty of existing data deletion schemes reach deletion by

overwriting the physical disks [8, 13, 18]. To be specific, they use some random data to overwrite the data which needs to be deleted. To make the data deletion operation more secure, some researchers suggest that the disk should be overwritten more than one times. Although overwriting the disk can theoretically solve the problem of data deletion, it still has some inherent limitations. On the one hand, overwriting the physical medium isn't efficient for real applications. Especially for distributed storage system, it is very difficult to overwrite every disk which maintains the data copy. On the other hand, there may be some physical remanence of the overwritten data left on the disk. The attacker who equips with advanced microsoping tools can recover the overwritten data with the physical remanence [5]. Therefore, it is desired to improve the efficiency and the security of the data deletion schemes.

Boneh and Lipton [2] firstly utilized cryptography technique to delete the data instead of protecting them, which can make the data deletion operation more secure and efficient, and resulting in plenty of follow-up schemes [10, 14, 16]. To be specific, these schemes should firstly use encryption key to encrypt the data before storing. Then they destroy a very short decryption key to make a large amount of related ciphertext unavailable, and return an erasure outcome to the data owner. This approach can efficiently delete the digital data. However, a lot of existing cryptography-based data deletion schemes are not able to achieve verifiability. That is, the data owner must trust the returned deletion result since he cannot verify it conveniently. However, the storage server may reserve the data maliciously and return a wrong outcome to cheat the data owner. Therefore, the requirement of deletion result verifiability should be introduced into the data deletion schemes.

Last but not least, some schemes have been put forwarded to provide the data owner with the ability to verify the data deletion result conveniently [11, 20, 22, 25, 26]. They delete the outsourced data and then return a related data deletion proof. The data owner can check the deletion result by verifying the returned deletion proof. However, these schemes all assume that the data owner is honestly. If the deleted data is later discovered, the cloud server is deemed to reserve the data maliciously, and the data owner should be entitled to compensation. However, there are some dishonest data owners in real-world, and they may expose the data maliciously to slander the cloud server to obtain compensation. All the existing schemes are not able to judge the data leakage source under the dishonest data owner and cloud server model because both of the two entities can obtain the same data backup. Therefore, we should offer the ability to track the data leakage source if the data is exposed.

Although various schemes have been put forward to deal with the problem of data deletion, most of them have a few deficiencies. First of all, in a lot of existing solutions, the data owner must trust the returned outcome since he cannot verify it. However, the cloud server might reserve the data backup maliciously and return a wrong outcome to cheat the data owner, but the data owner cannot detect the malicious behavior. Secondly, although some existing deletion schemes provide the data owner with the ability to verify the deletion outcome, they cannot achieve traceability. When the data is leaked, they cannot trace the data leakage source. To the best of our knowledge, it seems that there is not research work on publicly verifiable data deletion scheme that supports data leakage source tracking. Hence, we design a new scheme to delete the outsourced data and trace the data leakage source.

1.1 Our Contributions

In this paper, we put forward a novel publicly verifiable data deletion scheme for cloud storage, which can simultaneously achieve data leakage source tracking. The main contributions of our proposed scheme are as follow:

- We put forward a new Merkle hash tree-based publicly verifiable outsourced data deletion scheme, which can simultaneously achieve data leakage source tracking. To be specific, after executing data deletion operation, the cloud server can utilize Merkle hash tree to generate a deletion proof. If the cloud server reserves the data backup maliciously, the data owner can detect the cloud server's dishonest behavior by verifying the proof.

- Our proposed scheme can satisfy the property of traceability, which is different from the previous solutions. That is, if the data is leaked, the proposed scheme can trace the data leakage source, which can prevent data owner and cloud server from exposing the data maliciously to slander each other. Additionally, the proposed scheme is also very efficient in computation and communication.

This paper is an extension of our previous work that was presented at SICBS [24]. In the following, we show the main differences between this paper and the conference version. Firstly, we demonstrate the related work more detailedly in Section 1. Secondly, we put forward a more detailed scheme, and add the high description of the proposed scheme in Section 4. We also identify the main security properties for the proposed scheme in Section 3.3, and we prove that our new scheme can satisfy these design goals in Section 5.1. Finally, we will add the experimental simulation and performance comparison between our scheme and two previous schemes in Section 5.3.

1.2 Related Work

Data deletion has been studied for a long time. Perlman *et al.* [12] utilized a trusted third party (TTP) to solve the data deletion problem. First of all, they encrypt the data with a data key, then the TTP further encrypts the data key with a control key. When the data owner will not need the file anymore, the TTP will make

the data key unavailable by destroying the control key, thus the corresponding ciphertext cannot be decrypted anymore. In 2010, Tang *et al.* [15] designed a practical and implementable file assured deletion (FADE) system. They firstly use a data key to encrypt the file. After that the data key will be encrypted with a control key which associated with a policy. Besides, one or multiple TTPs maintain the policies together. Finally, when they want to delete the file, they can remove the related policy, and instruct the TTP to delete the corresponding key.

To offer the data owner the ability to verify the deletion outcome, Hao *et al.* [5] presented a novel data deletion protocol. In their protocol, they store the private key in a trusted platform module's protected memory. Then they reach data deletion by destroying the private key and finally return a signature as an evidence. In 2016, Luo *et al.* [8] presented a permutation-based data deletion scheme. They suppose that the cloud server could merely maintain the latest version of the data. Additionally, all the backups will be consistent when they are updated. Then they reach deletion by updating them with random data. Finally, the data owner is able to verify the deletion outcome through a challenge-response protocol. In 2018, Yang *et al.* [21] used blockchain to design a novel scheme to achieve publicly verifiably data deletion. In their scheme, they utilize blockchain to reach public verifiability without requiring any TTP, which is quite different from a lot of the previous schemes. After executing deletion operation, the cloud server can generate a deletion proof, which will be published on the blockchain. Finally, the data owner can check the deletion result by verifying the proof.

Xue *et al.* [19] put forward a verifiable data deletion method, which could also achieve provable data transfer and data integrity verification. Their scheme gives the data owner the ability to move the outsourced data between two different clouds. Moreover, the data owner is able to check the transferred data integrity on the target cloud through provable data possession (PDP) protocol. Then the original cloud deletes the transferred data blocks, and utilizes Rank-based Merkle hash tree (RMHT) to generate a deletion proof. Wang *et al.* [17] presented a similar method in 2018. Recently, Yang *et al.* [23] put forward a novel verifiable outsoutced data transfer and deletion scheme. In their scheme, the data owner is able to migrate the outsourced data between two different clouds, and then delete the transferred data from the original cloud server. Additionally, they utilize the primitive of vector commitment (VC) to realize public verifiability without requiring any TTP.

1.3 Organization

The remainder of this paper is organized as follows: We describe the preliminary of Merkle hash tree in Section 2. In Section 3, we describe the problem statement in detail. To be specific, we firstly formalize the system model of our novel scheme. Then we present the main security challenges. Finally, the security goals will be identified. In Section 4, we put forward our new publicly verifiable data deletion scheme in detail. A brief analysis of the proposed scheme, and the performance evaluation are presented in Section 5. Finally, we will conclude the proposed scheme in Section 6.

2 Merkle Hash Tree

Merkle hash tree (MHT), a specific binary tree, is always used to authenticate digital data [1, 4]. By using MHT, the communication and computation overhead during the verification process will be decreased greatly. In MHT, each leaf node maintains a hash value of the data block which needs to be authenticated, and every internal node keeps a hash value of the concatenation of its two children. For example, if we want to authenticate data set $D = \{d_1, d_2, d_3, d_4\}$, the MHT is illustrated as Figure 1, $h_{2\_i} = H(d_i)$, where $i \in [1, 4]$, $h_{1\_2} = H(h_{2\_1}||h_{2\_2})$, H is a secure one-way hash function. Finally, the public key signature technique is used to sign the root node.

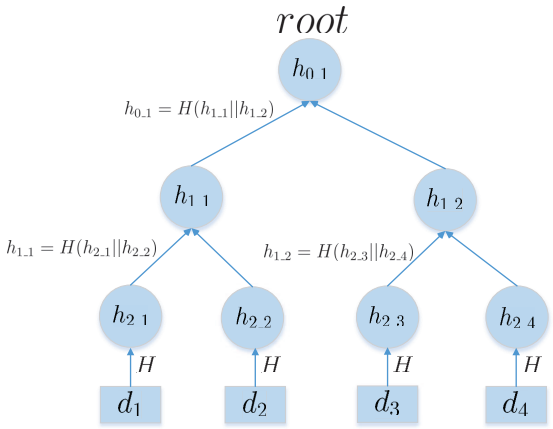

Figure 1: An example of MHT

The verifier can verify any subset of D by utilizing the verification object Φ, which is a set of all sibling nodes on the path from the authenticated leaf node to the root node. For instance, to verify d_3, Φ contains h_{1_1} and h_{2_4}. The verifier computes $h'_{0\_1} = H(h_{1\_1}||H(h_{2\_3}||h_{2\_4}))$ firstly. Then he checks that whether $h'_{0\_1} = h_{0\_1}$ holds, and verifies the validity of the signature. If both the verifications pass, it means that d_3 is valid; otherwise, it means that d_3 has been tampered with maliciously.

3 Problem Statement

3.1 System Model

In the following, we formalize the system model of our new scheme, which involves three entities: a data owner O, a cloud server S and a trusted agency TA, as illustrated in Figure 2.

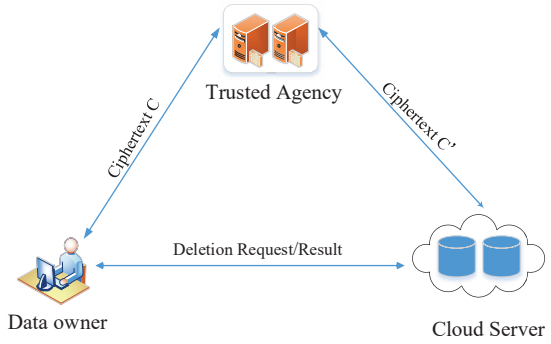

Figure 2: The system model

- **The data owner O.** It is a resource-constraint entry, who prefers to outsource his personal data to the cloud server for saving local storage overhead. When O will not need the data anymore, he sends a data deletion command to the cloud server to delete the outsourced data permanently. Finally, O can check the deletion result by verifying the returned deletion proof.
- **The cloud server S.** It refers to an entry which has large-scale computing resources, network resources and storage resources, thus, can maintain a large amount of data for resource-constraint O. When O will not need the outsourced data anymore, S will be required to delete the outsourced data from the disk. After that S may generate a deletion evidence for O to check the deletion outcome.
- **The trusted agency TA.** It is a trusted third party, which is absolutely righteous. That is to say, TA will never collude with O or S to cheat the other maliciously. TA always behaves honestly and righteously, therefore, both S and O fully trust TA unconditionally.

3.2 Security Threats

We assume that the cloud server S is "semi-honest-but-curious". As a result, S may not follow the protocol honestly for economic interests. Besides, the attackers, such as hackers or malicious users may try their best to access the outsourced data illegally. Therefore, we seriously consider the following two types of attacks: the internal attacks and the external attacks. The internal attacks are launched by the internal attackers, such as the dishonest cloud administrators, who would try to dig some sensitive information from the outsourced data. Furthermore, the dishonest S may share the outsourced data with others for financial incentives. The external attackers (*e.g.*, hackers and illegal users) might try to access the outsourced file and dig privacy data. Therefore, we should consider the following three security challenges.

- **Data privacy disclosure**. Privacy disclosure is a very common and serious security threat in cloud storage. On the one hand, the internal attackers are so curious that they may dig some sensitive information from the outsourced data. Moreover, the selfish cloud server moves the outsourced data to other subcontractors for saving storage overhead, or shares them with some other corporators for economic interests. On the other hand, the external attackers may try their best to access the file to find some privacy data.
- **Data corruption**. The outsourced data may be polluted for the following reasons. First of all, the manager performs erroneous operations, software or hardware malfunctions all may cause data loss. Secondly, the external attackers (*e.g.*, hackers) may modify or delete the data arbitrarily. Last but not least, when the data owner downloads the file, the cloud server sends part of the data for saving bandwidth, or delivers some unrelated data to cheat the data owner.
- **Malicious data reservation**. When the data owner will not need the data anymore, he will send a deletion command to the cloud server to delete the data permanently. However, the selfish cloud server might not execute the data deletion operation honestly for the following factors: (1) it needs some computational cost to delete the data from the physical medium; (2) the cloud server might try to reserve the data to dig some privacy data.

3.3 Design Goals

In our new scheme, we aim to achieve publicly verifiable data deletion in cloud storage. Meanwhile, when the data is leaked, we can trace the data leakage source efficiently. Therefore, our scheme should realize the following four goals.

- **Data confidentiality**. To ensure the outsourced data confidentiality, it should prevent the attackers from accessing the data directly because the data may contain some privacy information. That is, it is necessary to use cryptography algorithm to encrypt the file before uploading it to the cloud server. Moreover, the corresponding decryption key should be maintained secretly.
- **Data integrity**. To prevent the outsourced data from being polluted, the data owner should be given the ability to verify the data integrity and availability. If the outsourced data has been polluted, the data owner should be able to detect the malicious manipulation.
- **Verifiable data deletion**. To make the cloud server delete the data from physical medium sincerely, the data owner should be given the ability to verify the deletion result. If the cloud server reserves the data dishonestly, the data owner can detect the malicious data reservation by verifying the returned proof.

- **Accountable traceability**. To prevent the data owner and the cloud server from slandering each other, it should satisfy the property of accountable traceability. To be specific, we can trace the data leakage source precisely when the data is leaked. Additionally, upon the data owner and the cloud server executing some operations, they cannot deny their performances anymore.

4 Our Construction

4.1 High Description

In this paper, we study the problem of publicly verifiable cloud data deletion with efficient tracking under the commercial mode, which is very similar to schemes [5,19]. In our system model, there is a trust problem between the cloud server S and the data owner O: both of them might not fully believe each other. On the one hand, O might not believe that S will execute data deletion operation honestly. On the other hand, S thinks that O may reserve the data deliberately, and expose them to slander that S did not sincerely delete the data after deletion. Now plenty of data deletion methods have been proposed, but all of them assume that O is honest, thus O will not slander S maliciously. However, this assumption is not realistic. Therefore, we propose a novel scheme, which aims to make the data deletion publicly verifiable and the data leakage source traceable.

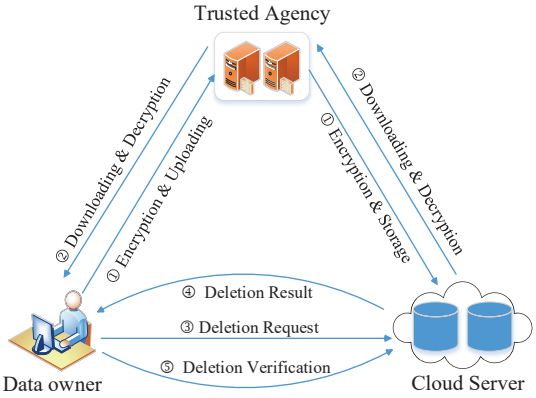

Figure 3: The main processes of our scheme

Our proposed scheme not only can achieve publicly verifiable data deletion but also can realize efficient data leakage source tracking, and Figure 3 describes the main steps of the proposed scheme. First of all, O encrypts the file to protect the privacy, and then sends the ephemeral ciphertext to TA. Then TA further encrypts the received ephemeral ciphertext and sends the final ciphertext to S. After that TA verifies the storage result, and O deletes the local backup. When O wants the outsourced file, he downloads the corresponding ciphertext and decrypts it to obtain the plaintext. If O will not need the file anymore, he is willing to send a deletion command to delete the data from S. Upon receiving the deletion request, S deletes the related data and returns a deletion evidence to O. Finally, O can check the data deletion result by verifying the proof. In our scheme, we utilize MHT to realize public verifiability, and the verification process does not need any TTP.

4.2 The Concrete Construction

In this part, we put forward our new scheme in detail. First of all, we define a few notations. Before embracing cloud storage service, the data owner O must pass the authentication of cloud server S. For simplicity, we assume that O has been authenticated and become a legal user of S. Then O can set a unique identity id, which is maintained by O and TA secretly. Besides, we suppose that O, S, TA respectively has a ECDSA key pair (pk_o, sk_o), (pk_s, sk_s) and (pk_t, sk_t). $H_1(\cdot)$ and $H_2(\cdot)$ are two secure hash functions. Furthermore, we assume that every file is named with a secret and unique name, and the name is so secure that it can resist brute-force attack. Without loss of generality, we can assume that O wants to upload file F to S, and n_f is the name of F.

- *Encrypt*. To guarantee data confidentiality, the data owner O should encrypt the file F before uploading, and the detailed processes are as follow.
  - First of all, O encrypts the file F: $C_o = Enc_{k_o}(F)$, where $k_o = H_1(sk_o||id||n_f)$, and Enc is an IND-CPA secure symmetric encryption algorithm. Then O computes a file tag $tag_f = H_1(n_f)$ and a hash value $h_o = H_1(C_o||id||tag_f)$. Finally, O sends the ephemeral ciphertext C_{f_o} to TA, where $C_{f_o} = (C_o, tag_f, h_o)$.
  - Upon receiving C_{f_o}, TA verifies that whether the equation $h_o = H_1(C_o||id||tag_f)$ holds. If $h_o \neq H_1(C_o||id||tag_f)$, TA aborts and returns failure; otherwise, TA further encrypts C_{f_o}: $C_t = Enc_{k_t}(C_{f_o})$, where $k_t = H_1(sk_t||id||tag_f)$. Then TA computes a hash value $h_t = H_1(C_t||id||tag_f)$. Finally, TA sends the final ciphertext C_{f_t} to S, where $C_{f_t} = (C_t, tag_f, h_t)$.
- *StoreCheck*. On receipt of C_{f_t}, the cloud server S maintains the data and returns a storage proof. For simplicity, assume that m files are stored in MHT.
  - Upon receipt of C_{f_t}, S stores the data in the leaf node of MHT. Here we can take $m = 8$ for example, and C_{f_t} is stored in the leaf node 6, as illustrated in Figure 4. Then S computes a signature on the hash value of the root node: $sig_r = Sign_{sk_s}(h_{0\_1})$, where $Sign$ is a ECDSA signature generation algorithm. Finally, S returns storage proof $\lambda = (sig_r, \Phi)$ to TA, where Φ is the verification object $(h_{1\_1}, h_{2\_4}, h_{3\_5})$.

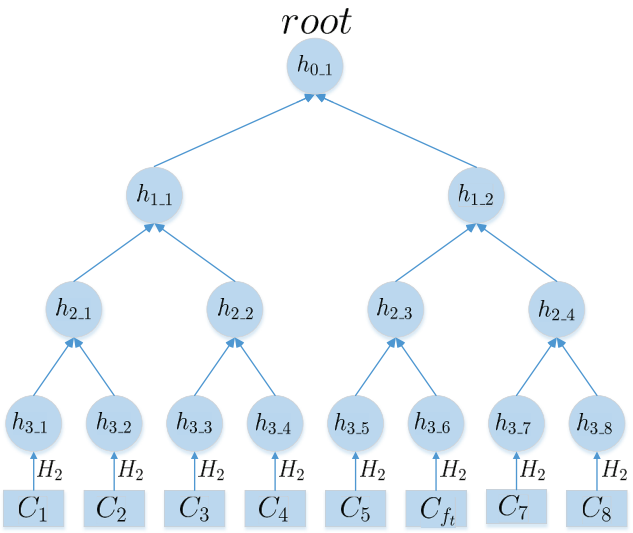

Figure 4: MHT for storage proof

– On receiving λ, TA checks that whether S stores the file and generates the storage proof honestly. To be specific, TA computes the following equations:

$$h'_{3\_6} = H_2(C_{f_t});$$
$$h'_{2\_3} = H_2(h_{3\_5}||h'_{3\_6});$$
$$h'_{1\_2} = H_2(h'_{2\_3}||h_{2\_4});$$
$$h'_{0\_1} = H_2(h_{1\_1}||h'_{1\_2});$$

Then TA checks that whether the equation $h'_{0\_1} = h_{0\_1}$ holds. If $h'_{0\_1} \neq h_{0\_1}$, TA quits and outputs failure; otherwise, TA verifies that if sig_r is a valid signature on h_{0_1}. If sig_r is not valid, TA quits and outputs failure; otherwise, TA trusts that S stores the data honestly, and sends λ to O, then O deletes the local backup.

- *Decrypt*. When data owner O needs the file F, he should download the ciphertext from the cloud server S and decrypt it to obtain the plaintext.

  – In order to download the ciphertext, O needs to generate a download request R_d. First of all, O computes a signature $sig_d = Sign_{sk_o}(download||tag_f||T_d)$, where T_d is a timestamp. Then O sends download request R_d to S, where $R_d = (download, tag_f, T_d, sig_d)$. Upon receiving R_d, S checks the validity of R_d through signature verification. If R_d is not valid, S quits and outputs failure; otherwise, S sends $C_{f_t} = (C_t, tag_f, h_t)$ and R_d to TA.

  – Upon receipt of C_{f_t} and R_d, TA firstly verifies the validity of R_d. If R_d is not valid, TA quits and outputs failure; otherwise, TA checks that if $h_t = H_1(C_t||id||tag_f)$ holds. If $h_t \neq H_1(C_t||id||tag_f)$, TA quits and outputs failure; otherwise, TA decrypts C_t to obtain the ephemeral ciphertext $C_{f_o} = Dec_{k_t}(C_t)$, where Dec represents a traditional symmetric decryption algorithm, and $k_t = H_1(sk_t||id||tag_f)$. Finally, TA sends $C_{f_o} = (C_o, tag_f, h_o)$ to O.

  – Upon receiving $C_{f_o} = (C_o, tag_f, h_o)$, O checks that if the equation $h_o = H_1(C_o||id||tag_f)$ holds. If $h_o \neq H_1(C_o||id||tag_f)$, O quits and outputs failure; otherwise, O executes decryption operation to obtain the corresponding plaintext $F = Dec_{k_o}(C_o)$, where $k_o = H_1(sk_o||id||n_f)$.

- *Delete*. When data owner O will not need the file F anymore, he wants to permanently delete the file from the cloud server S.

  – To delete the outsourced data, O needs to generate a deletion request R_e. Firstly, O computes a signature $sig_e = Sign_{sk_o}(delete||tag_f||T_e)$, where T_e is a timestamp. Then O generates a deletion request $R_e = (delete, tag_f, T_e, sig_e)$, and sends R_e to S.

  – Upon receipt of R_e, S firstly checks the validity of R_e. If R_e is not valid, S aborts and returns failure; otherwise, S deletes the data and computes a signature $sig_s = Sign_{sk_s}(delete||tag_f||T_e||R_e)$. Then S utilizes sig_s to replace C_{f_t} to re-construct the MHT, as illustrated in Figure 5. Finally, S computes a new signature on the hash value of the new root node $sig^*_r = Sign_{sk_s}(h^*_{0\_1})$, and returns a deletion proof τ to O, where $\tau = (R_e, sig_s, sig^*_r, \Phi)$, and $\Phi = (h_{1\_1}, h_{2\_4}, h_{3\_5})$.

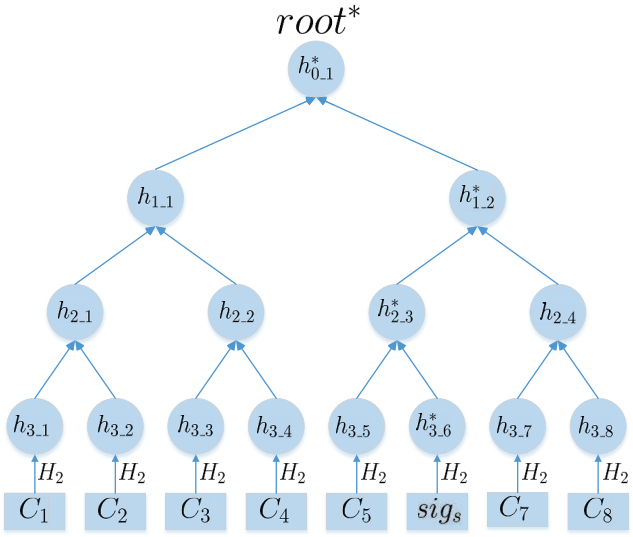

Figure 5: The MHT for deletion proofs

- *DelCheck*. After receiving τ, the data owner O can check the deletion result by verifying τ.

  – Firstly, O checks that whether the signature sig_s is valid. If sig_s is not valid, O quits and

outputs failure; otherwise, O computes the following equations:

$$h^{*'}_{3\_6} = H_2(sig_s);$$
$$h^{*'}_{2\_3} = H_2(h_{3\_5}||h^{*'}_{3\_6});$$
$$h^{*'}_{1\_2} = H_2(h^{*'}_{2\_3}||h_{2\_4});$$
$$h^{*'}_{0\_1} = H_2(h_{1\_1}||h^{*'}_{1\_2});$$

– Then O checks that whether the equation $h^{*'}_{0\_1} = h^{*}_{0\_1}$ holds. If $h*'_{0\_1} \neq h^{*}_{0\_1}$, O quits and outputs failure; otherwise, O verifies that whether the signature sig^{*}_{r} is a valid signature on $h^{*}_{0\_1}$. If sig^{*}_{r} cannot pass the verification, O quits and outputs failure; otherwise, O can trust that the deletion proof τ is valid. If the deleted data discovered again, based on the evidence, O should be entitled to compensation.

5 Scheme Analysis and Implementation

In the following section, we give a brief analysis of the proposed scheme. Firstly, we analyze the proposed scheme's security properties in detail. Secondly, we present the comparison among our scheme and some previous schemes in theory. Finally, we evaluate the performance through the simulation experiments.

5.1 Security Analysis

In this part, we analyze the security of our proposed scheme, including the data confidentiality, data integrity, public verifiability, traceability and non-repudiation.

5.1.1 Data Confidentiality

To protect the sensitive information, the data owner uses IND-CPA secure AES algorithm to encrypt the file before uploading. Additionally, the data owner keeps the encryption key and decryption key secret. That is, any attacker cannot acquire the encryption key and decryption key maliciously. In other word, the malicious attacker cannot obtain any plaintext information from the ciphertext. Hence, our novel scheme can reach data confidentiality.

5.1.2 Data Integrity

Our proposed scheme is able to ensure the outsourced data integrity. In the decryption process, the cloud server S will firstly send the final ciphertext C_{f_t} and the download request R_d to the trusted agency TA, where $C_{f_t} = (C_t, tag_f, h_t)$ and $R_d = (download, tag_f, T_d, sig_d)$. On receipt of C_{f_t} and R_d, TA will check R_d and C_{f_t} before decrypting. If R_d is not valid, it means that O does not require to download the file, and TA aborts; otherwise, TA checks C_{f_t}. To be specific, TA checks that whether the equation $h_t \stackrel{?}{=} H_1(C_t||id||tag_f)$ holds. The id is kept secret by O and TA. Therefore, S cannot forge a new C'_t to make equation $h_t = H_1(C'_t||id||tag_f)$ hold. That is, if and only if C_t is intact can the verifications pass, and TA decrypts C_t to obtain C_{f_o}. Therefore, TA always can detect the malicious operation if S falsifies C_t.

Besides, upon receiving $C_{f_o} = (C_o, tag_f, h_o)$ from TA, O will check it before executing decryption operation. To be specific, O verifies that whether the equation $h_o \stackrel{?}{=} H_1(C_o||id||tag_f)$ holds. The attacker cannot falsify a new C'_o to make equation $h'_o = H_1(C'_o||id||tag_f)$ hold because the id is maintained secretly by O and TA. That is, the attacker cannot forge C'_o to cheat O successfully. Therefore, if and only if C_{f_o} is intact can the verification pass.

As the analysis described above, the proposed scheme can achieve data integrity.

5.1.3 Public Verifiability

Our new scheme can reach publicly verifiable data deletion in cloud storage. After executing data deletion operation, the cloud server S generates a deletion proof τ to prove that he has performed data deletion honestly. Note that $\tau = (R_e, sig_s, sig^{*}_{r}, \Phi)$, where $\Phi = (h_{1\_1}, h_{2\_4}, h_{3\_5})$. Then anyone who given τ (called verifier) can check the data deletion result by verifying the evidence τ. Firstly, the verifier checks the validity of the deletion request R_e. If R_e is not valid, the verifier aborts and returns failure; otherwise, the verifier checks the validity of the signature sig_s. If sig_s is not valid, the verifier aborts and returns failure; otherwise, the verifier utilizes $H_2(sig_s)$ and Φ to re-compute $h^{*}_{0\_1}$. Finally, the verifier checks that whether the signature sig^{*}_{r} is a valid signature on $h^{*}_{0\_1}$. If and only if all verifications pass will the verifier trust that the deletion proof τ is valid. Note that the verification phases do not involve any private information, and any verifier can check the deletion outcome. Therefore, we think that our scheme is able to realize the property of public verifiability.

5.1.4 Traceability

The proposed scheme can trace the data leakage source precisely when the data is leaked. In our scheme, the data owner O owns the plaintext F and the ephemeral ciphertext C_{f_o}. The trusted agency TA further encrypts the ephemeral ciphertext C_{f_o} to obtain the final ciphertext C_{f_t}. Then the cloud server S maintains the final ciphertext C_{f_t}. Besides, O cannot access to the final ciphertext C_{f_t}, and S cannot access to the plaintext F and the ephemeral ciphertext C_{f_o}. That is, only TA and O can obtain the ephemeral ciphertext C_{f_o}, and only TA and S can obtain the final ciphertext C_{f_t}. Note that TA is absolutely impartial, and it will never collude with O (or S) to cheat S (or O). Hence, on the one hand, the data leakage source must be O if C_{f_o} is exposed. That is,

Table 1: Functionality comparison among three schemes

Scheme	Scheme [5]	Scheme [22]	Our Scheme
Trusted Third Party	Yes	Yes	Yes
Public Verifiability	Yes	Yes	Yes
Data Confidentiality	Yes	Yes	Yes
Data Integrity	No	Yes	Yes
Non-repudiation	No	Yes	Yes
Traceability	No	No	Yes

Table 2: Computational complexity comparison

Scheme	Scheme [5]	Scheme [22]	Our Scheme
(Encrypt)	$2\mathcal{E}+4\mathcal{H}$	$1\mathcal{E}+m\mathcal{H}$	$2\mathcal{E}+6\mathcal{H}$
(Decrypt)	$1\mathcal{E}+1\mathcal{D}+3\mathcal{H}$	-	$1\mathcal{S}+1\mathcal{V}+2\mathcal{D}+4\mathcal{H}$
(Store)	-	$1\mathcal{S}+1\mathcal{V}+46m\mathcal{H}$	$1\mathcal{S}+1\mathcal{V}+(2^{n+1}+n)\mathcal{H}$
(Delete)	$1\mathcal{S}$	$2\mathcal{S}+1\mathcal{V}+23\mathcal{H}$	$3\mathcal{S}+1\mathcal{V}+(n+1)\mathcal{H}$
(DelCheck)	$1\mathcal{V}$	$1\mathcal{V}+20\mathcal{H}$	$2\mathcal{V}+(n+1)\mathcal{H}$

the dishonest O cannot reserve C_{f_o} and then expose it to successfully slander that S did not delete the data honestly. On the other hand, the data leakage source must be S if C_{f_t} is leaked. Therefore, if S reserves C_{f_t} maliciously and resulting in data leakage, S cannot deny his dishonest data reservation. That is, the proposed scheme can reach the data leakage source traceability.

5.1.5 Non-repudiation

In our scheme, we assume that both the data owner O and the cloud server S may deny their behaviors thus slander the other. Without loss of generality, we analyze the non-repudiation when S is malicious and O is dishonest, respectively.

Case 1: Malicious cloud server S. The malicious cloud server S may slander the data owner O. First of all, the malicious S deletes the outsourced data arbitrarily to save storage overhead, and then slanders that he performed the data deletion operation as O's command. For this scenario, O can require S to present the data deletion request $R_e = (delete, tag_f, T_e, sig_e)$, where sig_e is a signature generated by O with private key sk_o. On the one hand, O had never generated and sent R_e to S to delete the data. On the other hand, S cannot forge a valid R_e since S does not has the private key sk_o. Therefore, S cannot present R_e to prove that he deleted the outsourced data as O's command. Secondly, S reserves the data dishonestly and slanders that O had not required him to remove the outsourced data. Here, O can demonstrate the data deletion proof $\tau = (R_e, sig_s, sig_r^*, \Phi)$, where sig_s is a signature generated by S with private key sk_s. The signature sig_s can be seen as a proof, which can prove that O has required S to delete the data, and S has responded to this request. That is, the dishonest S cannot successfully slander O.

Case 2: Dishonest data owner O. The dishonest data owner O denies his behavior and slanders the cloud server S maliciously. Firstly, O had asked S to delete the data, and S had done it honestly. However, O declares that he had never asked S to delete the data, and slanders that S deleted the data arbitrarily. Here, S can show the deletion request $R_e = (delete, tag_f, T_e, sig_e)$, which contains a signature sig_e generated by O. No one else can forge a valid signature sig_e to further forge a valid deletion request R_e. Therefore, R_e can be seen as an evidence which can prove that O had required S to delete the data. Secondly, O had never required S to delete the data. Nevertheless, O declares that he had required S to delete the file and S did not do it sincerely. In this case, S can require O to show the deletion proof $\tau = (R_e, sig_s, sig_r^*, \Phi)$ which generated by S. However, S did not generate τ at all. In addition, O cannot forge a valid τ. Therefore, O cannot slander S successfully.

5.2 Comparison

In the following, we compare the functionality and computational complexity among our new scheme and two previous schemes [5, 22] in theory, then Table 1 and Table 2 demonstrate the comparison results, respectively.

From Table 1 we can have the following findings. Firstly, all of these three schemes need to introduce a trusted third party to achieve publicly verifiable data deletion. Secondly, all of them can guarantee data confidentiality, which can protect the sensitive information that contained in the outsourced file. Thirdly, Hao *et*

al. scheme [5] cannot satisfy the data integrity and non-repudiation, which is different from our scheme and Yang *et al.* scheme [22]. Last but not least, only our new scheme is able to achieve traceability, which can prevent data owner and cloud server from slandering each other. Overall, our new scheme is more attractive than the other two schemes.

Then we compare the performance of the three schemes in theory, and the results are listed as time complexities in Table 2. For simplicity, we use symbols $\mathcal{E}$ and $\mathcal{D}$ to represent a symmetry encryption and decryption, respectively. Moreover, we denote by $\mathcal{H}$ a hash computation, $\mathcal{S}$ a signature generation operation, and $\mathcal{V}$ a signature verification operation. Meanwhile, we assume that the MHT has $m = 2^n$ leaf nodes and the number of data blocks in scheme [22] is m. Finally, we ignore the other overhead, such as multiplication and communication overhead.

5.3 Performance Evaluation

In this part, we simulate our proposed scheme and two previous schemes [5, 22], then provide the performance evaluation. The related algorithms are implemented with PBC library and the OpenSSL library on an Unix machine, which equips with Intel(R) Core(TM) i5-6200U processors running at 2.4 GHz and 8 GB main memory. For simplicity, we simulate all the entities on this Linux machine and ignore the communication overhead.

The outsourced file always contains some sensitive information, which should be kept secret. Therefore, the data owner needs to use secure encryption algorithm to encrypt the file before uploading. We increase the size of the file from 0.125 MB to 1 MB with a step for 0.125 MB, and the number of data blocks in Yang *et al.* scheme [22] is fixed in 1024. Then the approximate time cost is shown in Figure 6. We can find that although the time overhead will increase with the size of the file, the encryption operation is one-time. Moreover, our proposed scheme and Hao *et al.* scheme [5] cost almost the same time cost to encrypt the same size of file. Meanwhile, Yang *et al.* scheme [22] needs more time cost than the other two schemes. Hence, we think our proposed scheme is efficient to encrypt file.

After uploading the file to the cloud server, the data owner wants to check that whether the cloud server maintains the data honestly. The main computation comes from storage proof generation and storage result verification. In our scheme, the cloud server needs to compute $(2m - 1)$ hash values and a ECDSA signature to generate storage proof, where $m = 2^n$. Then the data owner needs to execute $(n + 1)$ hash calculations and a signature verification operation to verify the storage result. In Yang *et al.* scheme [22], the computation consists of a signature generation and a signature verification, and $46m$ hash computations. We increase the number n from 1 to 8 with a step for 1, and then Figure 7 presents the efficiency comparison. From Figure 7 we can realize that although the computational overhead increases with

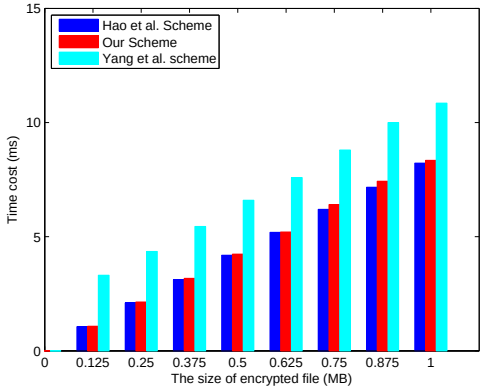

Figure 6: Time cost of encryption

n, the growth rate of our scheme is relatively lower than that of Yang *et al.* scheme [22]. Meanwhile, our proposed scheme costs less time overhead. Therefore, our proposed scheme is more efficient than Yang *et al.* scheme [22].

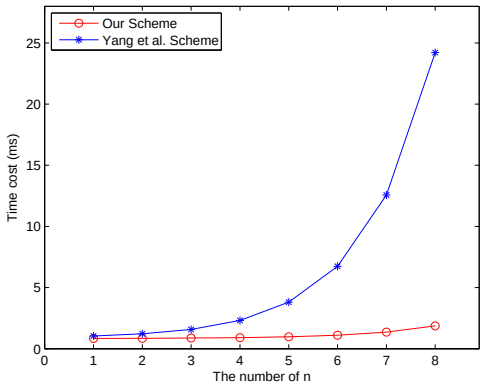

Figure 7: Time cost of storage

For saving storage overhead, the data owner does not maintain any local data backup after outsourcing the file to the remote cloud server. Therefore, when the data owner needs the file, he needs to download the corresponding ciphertext from the cloud server, and then decrypts it to obtain the plaintext. We increase the ciphertext from 0.125 MB to 1 MB with a step for 0.125 MB, and test the approximate time overhead. Then the efficiency comparison of decryption process between the two schemes is shown in Figure 8. From Figure 8 we can realize that the time cost will increase with the size of the decrypted ciphertext, and our scheme's growth rate is lower than that of Hao *et al.* scheme [5]. Moreover, although our scheme will cost a little more time when the ciphertext is less than 0.75 MB, the extra overhead is small and acceptable. Further, when the ciphertext is larger than 0.75 MB, the time cost of our scheme is less than that of Hao *et al.* scheme [5]. In real application, the ciphertext is often larger than 0.75 MB. Therefore, we can think that our scheme is more efficient than Hao *et al.* scheme [5] to

decrypt same size of ciphertext.

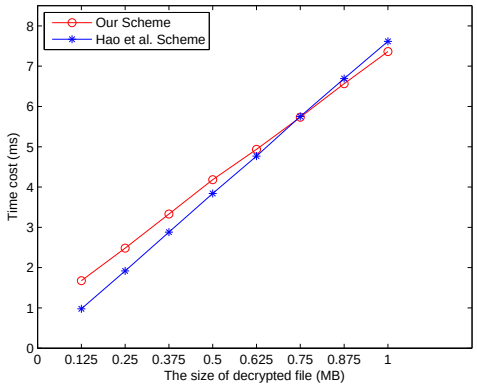

Figure 8: Time cost of decryption

When the data owner will not need the outsourced file anymore, he wants to permanently delete the data from the cloud server. To delete the outsourced file, our scheme needs to execute three signature generation operations and a signature verification computation. Moreover, our scheme also needs to perform $(n+1)$ hash calculations to update the MHT. However, Hao *et al.* scheme [5] merely needs to generate a signature. Meanwhile, Yang *et al.* scheme [22] needs to compute 23 hash values, generate two signatures and perform a signature verification operation. Then the efficiency comparison among the three schemes is shown in Figure 9. We can easily find that the time overhead of our scheme will increase with the number n, but the growth rate is very low. However, the time overhead of the other two schemes is almost constant. Moreover, although our scheme needs a little more time to delete a file, the time cost is very small. For example, when $n = 40$, the time cost is about 1.5 microseconds. Meanwhile, note that the deletion operation is one-time. Therefore, our scheme is still very efficient.

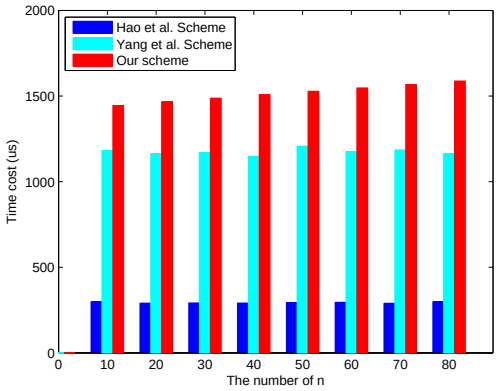

Figure 9: Time cost of deletion

After deletion, the data owner is able to check the data deletion result through verifying the returned deletion evidence. Our scheme needs to execute two signature verification operations and $(n+1)$ hash computations to verify the deletion result. However, Hao *et al.* scheme [5] only needs to verify the validity of a signature. Meanwhile, Yang *et al.* scheme [22] needs to verify a signature and compute 20 hash values. Then the time cost comparison among the tree schemes is demonstrated in Figure 10. We can easily find that the time cost of our scheme will increase with n, while the time of the other two schemes is almost constant. However, the growth rate of our scheme is very low. Additionally, although our scheme costs a litter more time than the other two schemes, the time cost is very small. Meanwhile, note that the verification operation can be finished off-line. Therefore, it will not affect the overall efficiency.

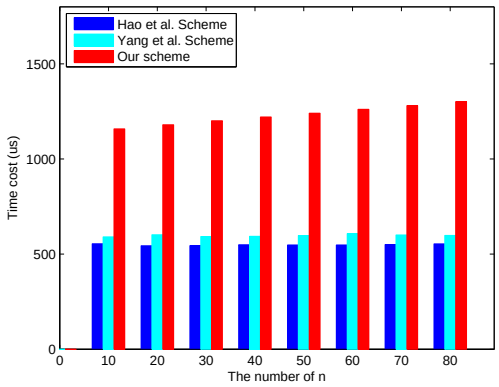

Figure 10: Time cost of deletion verification

6 Conclusions

In this paper, we put forward a novel MHT-based publicly verifiable outsourced data deletion scheme, which also supports efficient data leakage source tracking. In cloud storage, both the data owner and the cloud server might think the other is dishonest. In our new scheme, we use the cryptographic primitive of MHT to deal with this trust problem. To be more specific, the cloud server should use MHT to compute an evidence after deletion. If the cloud server reserves the data maliciously, the data owner is able to easily detect the dishonest data reservation by verifying the deletion proof. In addition, our novel scheme can satisfy the property of data leakage source traceability, which can prevent the data owner and cloud server from exposing the data to slander the other. In the future, we will study how to reach data deletion and leakage source traceability without requiring any TTP.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61962015), the Science and Technology Program of Guangxi (No.AB17195045), and the Natural Science Foundation

of Guangxi (No.2016GXNSFAA380098). Moreover, the authors gratefully acknowledge the anonymous reviewers for their valuable comments.

References

[1] G. S. Aujla, R. Chaudhary, N. Kumar, A. K. Das, and J. J. Rodrigues, "Secsva: Secure storage, verification, and auditing of big data in the cloud environment," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 78–85, 2018.

[2] D. Boneh and R. Lipton, "A revocable backup system," in *Proceedings of the 6th Conference on USENIX Security Symposium*, pp. 91–96, pp. 22-25, 1996.

[3] K. Brindha and N. Jeyanthi, "Securing portable document format file using extended visual cryptography to protect cloud data storage," *International Journal of Network Security*, vol. 19, no. 5, pp. 684–693, 2017.

[4] N. Garg and S. Bawa, "Rits-mht: Relative indexed and time stamped merkle hash tree based data auditing protocol for cloud computing," *Journal of Network and Computer Applications*, vol. 84, pp. 1–13, 2017.

[5] F. Hao, D. Clarke, and A. F. Zorzo, "Deleting secret data with public verifiability," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 6, pp. 617–629, 2016.

[6] W. F. Hsien, C. C. Yang, and M. S. Hwang, "A survey of public auditing for secure data storage in cloud computing," *International Journal of Network Security*, vol. 18, no. 1, pp. 133–142, 2016.

[7] G. Lou and Z. Cai, "A cloud computing oriented neural network for resource demands and management scheduling," *International Journal of Network Security*, vol. 21, no. 3, pp. 477–482, 2019.

[8] Y. Luo, M. Xu, S. Fu, and D. Wang, "Enabling assured deletion in the cloud storage by overwriting," in *Proceedings of the 4th ACM International Workshop on Security in Cloud Computing*, pp. 17–23, May 2016.

[9] H. Ma, X. Han, T. Peng, and L. Zhang, "Secure and efficient cloud data deduplication supporting dynamic data public auditing," *International Journal of Network Security*, vol. 20, no. 6, pp. 1074–1084, 2018.

[10] Z. Mo, Y. Qiao, and S. Chen, "Two-party fine-grained assured deletion of outsourced data in cloud systems," in *Proceedings of the IEEE 34th International Conference on Distributed Computing Systems (ICDCS'14)*, pp. 308–317, July 2014.

[11] J. Ni, X. Lin, K. Zhang, Y. Yu, and X. X. Shen, "Secure outsourced data transfer with integrity verification in cloud storage," in *Proceedings of the IEEE/CIC International Conference on Communications in China (ICCC'16)*, pp. 1–6, July 2016.

[12] R. Perlman, "File system design with assured delete," in *Proceedings of the Third IEEE International Security in Storage Workshop (SISW'05)*, pp. 83–88, Dec. 2005.

[13] J. Reardon, D. Basin, and S. Capkun, "Sok: Secure data deletion," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'13)*, pp. 301–315, May 2013.

[14] J. Reardon, S. Capkun, and D. A. Basin, "Data node encrypted file system: Efficient secure deletion for flash memory," in *Proceedings of the 21st USENIX Conference on Security Symposium*, pp. 333–348, Aug. 2012.

[15] Y. Tang, P. P. Lee, J. C. Lui, and R. Perlman, "Fade: Secure overlay cloud storage with file assured deletion," in *Proceedings of the 6th Iternational ICST Conference on Security and Privacy in Communication Systems*, pp. 380–397, Sep. 2010.

[16] W. L. Wang, Y. H. Chang, P. C. Huang, C. H. Tu, , H. W. Wei, and W. K. Shih, "Relay-based key management to support secure deletion for resource-constrained flash-memory storage devices," in *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC'16)*, pp. 444–449, Jan. 2016.

[17] Y. Wang, X. Tao, J. Ni, and Y. Yu, "Data integrity checking with reliable data transfer for secure cloud storage," *International Journal of Web and Grid Services*, vol. 14, no. 1, pp. 106–121, 2018.

[18] M. Y. C. Wei, L. M. Grupp, F. E. Spada, and S. Swanson, "Reliably erasing data from flash-based solid state drives," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*, pp. 105–117, Feb. 2011.

[19] L. Xue, J. Ni, Y. Li, and J. Shen, "Provable data transfer from provable data possession and deletion in cloud storage," *Computer Standards & Interfaces*, vol. 54, pp. 46–54, 2017.

[20] L. Xue, Y. Yu, Y. Li, M. H. Au, X. Du, and B. Yang, "Efficient attribute-based encryption with attribute revocation for assured data deletion," *Information Sciences*, vol. 479, pp. 640–650, 2019.

[21] C. Yang, X. Chen, and Y. Xiang, "Blockchain-based publicly verifiable data deletion scheme for cloud storage," *Journal of Network and Computer Applications*, vol. 103, pp. 185–193, 2018.

[22] C. Yang, X. Tao, F. Zhao, and Y. Wang, "A new outsourced data deletion scheme with public verifiability," in *Proceedings of the 14th International Conference on Wireless Algorithms, Systems, and Applications (WASA'19)*, pp. 631–638, June 2019.

[23] C. Yang, J. Wang, X. Tao, and C. Chen, "Publicly verifiable data transfer and deletion scheme for cloud storage," in *Proceedings of the 20th International Conference on Information and Communications Security (ICICS'18)*, pp. 445–458, Oct. 2018.

[24] C. Ynag and X. Tao, "New publicly verifiable cloud data deletion scheme with efficient tracking," in *Pro-*

ceedings of the 2th International Conference on Security with Intelligent Computing and Big-data Services (SICBS'18), pp. 1–14, Dec.2018.

[25] Y. Yu, J. Ni, W. Wu, and Y. Wang, "Provable data possession supporting secure data transfer for cloud storage," in *Proceedings of the 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA'15)*, pp. 38–42, Nov. 2015.

[26] Y. Yu, L. Xue, Y. Li, X. Du, M. Guizani, and B. Yang, "Assured data deletion with fine-grained access control for fog-based industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4538–4547, 2018.

Biography

Changsong Yang biography. Changsong Yang received the B.S. degree in school of telecommunications engineering from Xidian university, Xi'an, China in 2014 and became a master degree candidate of this university in 2014. And now he is currently undergoing his Ph.D degree at the School of Cyber Engineering at Xidian university. His research interests are network security, cloud computing, verifiable data deletion and blockchian.

Xiaoling Tao biography. Xiaoling Tao received the M.S. degree in computer application technology from Guilin University of Electronic Technology, Guilin, China in 2008 and became the faculty member of this university in 2000. She is currently a professor at the school of computer science and information security, Guilin University of Electronic Technology. And now she is studing for her Ph.D degree in information and communication engineering at Guilin University of Electronic Technology. Her current research interests include network security, cloud computing and computational intelligence.

Qiyu Chen biography. Qiyu Chen is a research scholar in the College of Information and Computer Engineering, Northeast Forestry University.