Hindawi
Security and Communication Networks
Volume 2021, Article ID 6686281, 15 pages
https://doi.org/10.1155/2021/6686281

Research Article

Aggregation-Based Tag Deduplication for Cloud Storage with Resistance against Side Channel Attack

Xin Tang[1], **Linna Zhou**[2], **Bingwei Hu**[1], and **Haowen Wu**[1]

[1] *School of Information Science and Technology, University of International Relations, Beijing 100091, China*
[2] *School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China*

Correspondence should be addressed to Xin Tang; xtang@uir.edu.cn

Received 1 December 2020; Revised 23 December 2020; Accepted 2 January 2021; Published 16 February 2021

Academic Editor: Jinwei Wang

Tag deduplication is an emerging technique to eliminate redundancy in cloud storage, which works by signing integrity tags with a content-associated key instead of user-associated secret key. To achieve public auditability in this scenario, the linkage between cloud users and their integrity tags is firstly re-established in current solutions, which provides a potential side channel to malicious third-party auditor to steal the existence privacy of a certain target file. Such kind of attack, which is also possible among classic public auditing schemes, still cannot be well resisted and is now becoming a big obstacle in using this technique. In this paper, we propose a secure aggregation-based tag deduplication scheme (ATDS), which takes the lead to consider resistance against side channel attack during the process of public verification. To deal with this problem, we define a user-associated integrity tag based on the defined content-associated polynomial and devise a Lagrangian interpolation-based aggregation strategy to achieve tag deduplication. With the help of this technique, content-associated public key is able to be utilized instead of a user-associated one to achieve auditing. Once the verification is passed, the TPA is just only able to make sure that the verified data are correctly corresponding to at least a group of users in cloud storage, rather than determining specific owners. The security analysis and experiment results show that the proposed scheme is able to resist side channel attack and is more efficient compared with the state of the art.

1. Introduction

With the rapid development of cloud storage, efficiency has gradually become an important issue since growing amount of redundant data are generated and outsourced to cloud in the big data era. Considering the privacy, most of them are encrypted before outsourcing, which brings a huge challenge to cloud data deduplication. To achieve ciphertext deduplication, convergent encryption (CE) [1] is proposed as an effective manner. It works by encrypting data with a content-associated key, such as hash value of the plaintext, to ensure that the same plaintext corresponds to the identical ciphertext. However, taking the requirement of integrity audit into account in this scenario, deduplication of verification tags which are generated using user-associated signing key remains a big challenge.

Motivated by convergent encryption, in order to achieve tag deduplication, content-associated key could be used instead of user-associated signing key in the process of tag generation. In this way, ciphertext chunks with the same content, no matter signed by which owner, correspond to the same tag. However, this naïve solution may break the linkage between tags and its owners during the process of integrity auditing performed by the third party auditor (TPA) [2]. In order to enable the public auditability for cloud data, users have to generate their public keys and leave them to the TPA for bilinear pairing-based verification, which makes the aforementioned solution infeasible since in this case verification tags should still be signed by secret keys of users. Moreover, such kind of public verification actually creates a side channel for TPA, which is able to steal data existence privacy of a specific user according to the result of integrity auditing.

In the literature, the existing work for tag deduplication relies on proxy re-signature technique to establish the linkage between data owners and their tags during the auditing process [2]. In [2], verification tags are able to be deduplicated in cloud storage since universal content-associated key is utilized in the process of tag generation instead of user-associated signing key. To keep the linkage of users and their tags during the process of integrity auditing, the verification tag is transformed to the one signed by user-associated key using the re-signature key generated by the data owner before integrity verification is performed by the TPA. However, re-signature keys still bring about a huge amount of storage and management overhead to the CSP. And the possible side channel attack launched by the TPA is not considered. As a potential way to achieve secure deduplication, in [3], a threshold tag aggregation scheme is proposed. In their design, verification tags are firstly generated with the defined user-associated key and then aggregated into a uniform one with the help of Lagrangian interpolation [4] if the number reaches a certain threshold. According to their scheme, the verification is performed utilizing aggregated public key. Thus, even though TPA knows existence of the verified data according to the verification result, it still cannot determine the ownership since the data have already been a popular one. This work is able to achieve tag deduplication and side channel attack resistance simultaneously. However, to facilitate signature aggregation and public key generation, specific parameters need to be transmitted among a constant group of users, and thus, it does not work for the open deduplication system.

Therefore, in this paper, we are going to tackle how to enable an efficient tag deduplication scheme in an open cloud storage system with security guaranteed. To the best of our knowledge, ATDS is the first threshold tag deduplication scheme that allows resistance to side channel attack. With the same level of security guaranteed, our scheme surpasses existing ones in feasibility during the process of deduplication. Specifically, we utilize the technique of Lagrangian interpolation, which requires every one of the cloud users to generate a verification tag for their data chunk based on the newly defined secret key which is associated with both content and unique index of the user before outsourcing to cloud storage. With the help of Lagrangian interpolation, the CSP is able to aggregate any group of tags for identical data chunk once the number involved reaches the threshold value, which in turn makes threshold deduplication for verification tags in an open deduplication system possible. Moreover, to resist side channel attack during the integrity auditing process, we use content-associated public key instead of user-associated one to achieve auditing. Once the verification is passed, the TPA is just only able to make sure that the verified data are correctly corresponding to at least a group of users in cloud storage, rather than determining specific owners. In addition, we also provide an integrity checking mechanism for data owners once the number of tags in cloud storage does not reach the threshold value. The main contributions for this paper are summarized as follows:

(1) We put forward a side channel attack-resistant public auditing framework based on aggregation strategy with tag deduplication supported. Under the proposed framework, for a certain data chunk, tags generated by different owners can be aggregated into a uniform one, which makes deduplication possible. In terms of security, side channel attack launched by the TPA is able to be resisted since public auditability is achieved by utilizing the content-associated public key instead of user-associated one.

(2) Next, we focus on designing an aggregation scheme for tags under the proposed framework. Specifically, we devise a novel secret key for users with the help of the defined polynomial and present a Lagrangian interpolation-based scheme to aggregate tags for identical chunk once the number reaches a predefined threshold value. Moreover, a corresponding content-associated public key, which is only able to be generated by data owners, is defined to achieve public auditability.

(3) We perform security analysis for the proposed scheme and take experiments to evaluate the performance. Both theoretical and experimental results show that the proposed scheme is able to resist side channel attack launched by the TPA, with just only limited overhead required.

2. Related Work

As an effective way to check the correctness of data in cloud storage, integrity auditing [5, 6] has already attracted a lot of attention from a growing number of researchers. Ateniese et al. [5] proposed the first integrity auditing scheme defined as “provable data possession (PDP).” In their scheme, RSA-based homomorphic tags are generated for each one of the data chunks at first and then outsourced to the cloud to relieve local storage of the data owner, which can be used to check the correctness of corresponding chunks in cloud storage. However, due to the property of RSA signature, the length of a tag in this scheme is at least 1024 bits long considering the security, which occupies a large amount of cloud storage. Moreover, once the verifier is a third-party entity, the privacy of data would be leaked during the process of auditing. As an improvement, BLS signature [7] is employed instead of RSA signature to restrict the length of integrity tags to 160 bits. Based on this work, Wang et al. [6] proposed the first privacy-preserving public auditing scheme for secure cloud storage. In their design, once a verifier wants to check the correctness of the corresponding chunks, it generates a challenge and receives a proof returned from the CSP, which consists of combined chunks masked with randomness, and aggregated homomorphic verifiable tags. From the received proof, the TPA is no longer able to derive the content of challenged chunks; thus, data privacy is considered to be protected. As a follow-up work, Wang et al. [8] extended the classic PDP model to support verification of dynamic updates to the stored data by presenting a Merkle hash tree- (MHT-) based construction. Tian et al. [9] improved the scheme by presenting a new data structure named dynamic hash table, which is maintained by the TPA to achieve efficient data dynamic verification. Moreover, Yang et al. [10] considered this problem in the scenario of multiple owners. Zhu et al. [11] extended provable data possession to multicloud storage. And Shen et al. [12] developed the similar scheme to achieve verification taking data sharing into account. However, the schemes introduced above only focused on the problem of data verification in cloud storage, rather than the storage efficiency. For integrity tags, even though they correspond to the same chunk, they cannot be simply deduplicated since they are actually signed by secret keys of different owners, respectively. Moreover, according to the principle of integrity verification introduced above, the process of public verification actually creates a side channel for TPA to steal data existence privacy of a specific user according to the result of integrity auditing, which is a big security threat that has not been resolved until now. Once the returned proof is verified to be correct based on the public key of a user, the TPA is able to infer that the user is the owner of challenged data instantly.

The two problems mentioned above suffer from the same dilemma as ciphertext deduplication as well as its corresponding side channel attack resistance strategy, which have been well studied in recent years. For either integrity tags or ciphertext chunks, even though they correspond to the same plaintext chunk, their content may be divergent since they are signed or encrypted with different keys. To achieve ciphertext deduplication, Douceur et al. [1] proposed a novel convergent encryption (CE) scheme, which is able to encrypt data in a deterministic way, thus ensuring users with the same plaintext to generate the identical ciphertext. A straightforward way to realize CE scheme is to employ hash value as the encryption key. Thus, every one of the data owners could easily work out the convergent encryption key. A lot of successive works [13–17] focused on how to resist possible side channel attack in this scenario. Among them, some works [14, 18] relied on an independent key server (KS) to introduce randomness into the generation of convergent key, where inefficient interactive blind signature protocol is usually needed. To improve efficiency, Liu et al. [13], Yu et al. [19], and Tang et al. [20] eliminated the need of KS and achieved the same level of security meanwhile. Even though randomness is introduced, these works are still confronted with side channel attack since attackers are completely able to forge legal identities to acquire such randomness, such as by launching Sybil attack.

As another way to deal with side channel attack, Harnik et al. [16] proposed a threshold deduplication scheme. It works by setting a random threshold value for every file and performing cloud side deduplication only if the number of replicas corresponding to the same file reaches the value. However, the scheme is designed for unencrypted file, and it relies on an independent server to keep and maintain threshold values for different files, which suffers from the problem of single point failure. As an improvement, Stanek et al. [17] developed the work to ensure the security of ciphertext deduplication. Specifically, they presented a novel framework, in which deduplication is triggered only when the corresponding file has already been a popular one; thus, there is no risk of privacy leakage. As a follow-up work, Zhang et al. [15] resorted to the k-anonymity technique to achieve a concrete deduplication strategy. In their design, the cloud user firstly generates a convergent ciphertext based on CE scheme. Then, it re-encrypts the ciphertext to obtain identity-associated convergent ciphertext, which is outsourced to the cloud storage together with auxiliary information for further verification. Once the number of ciphertext replicas for a certain file in cloud storage reaches k, convergent ciphertext is recovered instantly to trigger cloud side deduplication.

Such design provides a potential effective means to solve the similar problem in tag deduplication. Liu et al. [2] proposed the principle of tag deduplication, which is motivated by convergent encryption in ciphertext deduplication. In their design, integrity tags for data chunks are generated based on convergent key, which can be obtained by calculating the hash value of corresponding chunks. However, just only deduplication of integrity tags is not enough since during the process of integrity verification, the linkage between users and their tags must be established. To solve this problem, they employed the proxy re-signature technique to transform the signature of tags in cloud storage before proof generation, which in turn still suffers from side channel attack during the process of public verification. As a follow-up work, Huang et al. [3] presented a threshold aggregation scheme for integrity tags. In order to achieve aggregation, they replaced convergent encryption key with the cumulative values of user-associated random polynomials in the process of tag generation. As a result, the defined tags corresponding to the same chunk could be aggregated by employing Lagrangian interpolation once its number reaches the predefined threshold value t; thus, successive deduplication is enabled. Such design is similar to threshold deduplication of ciphertext, which is able to keep the linkage of verified tags and its owners to some extent. Even though the aggregated tag is verified to be correct during the process of public verification, TPA still cannot determine ownership of the corresponding file since it has been a popular one; thus, side channel attack is successively resisted. However, the design is not suitable for open deduplication system since parameters transmission among a fixed group of users is always necessary during the process of signature key generation for each user.

3. System Description and Design Goals

3.1. System Model. We consider a cloud storage system consists of three entities: cloud user, cloud service provider (CSP), and third-party auditor (TPA). Considering the existence of requested data during the process of deduplication checking, and whether tag aggregation has been triggered by the CSP, the system model is divided into three cases, as illustrated in Figures 1(a), 1(b), and 1(c), respectively.

The cloud user is an entity who has large amounts of data to be outsourced to the cloud for storage. In order to protect the privacy, it usually encrypts data before outsourcing. Considering the communication and cloud-side storage efficiency, the user first checks existence of the target data. According to the checking result, it determines whether to perform a complete upload, as is shown in Figures 1(a) and

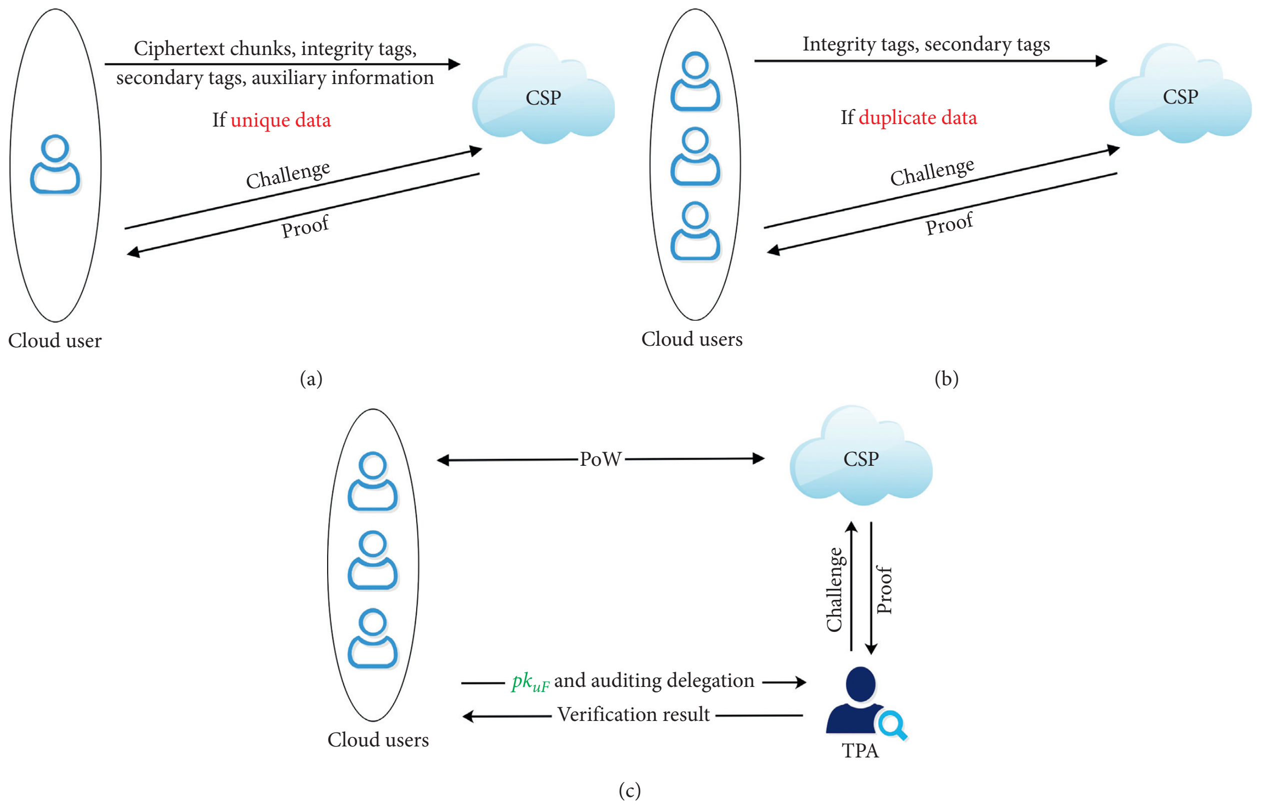


FIGURE 1: System model of cloud storage service: (a) the process of unique data outsourcing; (b) the process of duplicate data outsourcing before tag aggregation; (c) integrity auditing after tag aggregation.

1(b), respectively. Specifically, when *UID* is exposed, the adversary could then calculate the user-associated secret key, which is able to be utilized to check the user-associated proof returned from the CSP. Therefore, secondary tags are necessary to deal with such a potential side channel attack. If the uploaded data are unique, the cloud user has to upload complete data, including ciphertext chunks, auxiliary information, integrity tags, and secondary tags as well. Otherwise, once the data are already existed in cloud storage, the user only needs to outsource integrity tags for proof of ownership (POW) verification and secondary tags for private verification. Specifically, if the number of corresponding tags in cloud storage does not reach the predefined threshold value, the user has to upload integrity tags and secondary tags for each one of the ciphertext chunks. Otherwise, as is shown in Figure 1(c), once the threshold value has been reached, there is no need to upload secondary tags, and the user only needs to outsource an aggregated verification tag for the target file, which is utilized in proof of ownership and then deleted from the storage. The difference is that once the ownership is proved in this case, tags are deleted instantly to achieve tag deduplication. However, in former cases, qualified integrity tags are stored in the cloud storage to participate in subsequent tag aggregation.

CSP is an entity who manages a great amount of cloud servers to provide significant storage space and computation resources. Once it receives a deduplication request from a cloud user, it searches for the requested data in local storage and queries whether tags have been aggregated before generating a deduplication response. It is worth mentioning that the CSP maintains a tag counter for each one of the ciphertext chunks. Once the number of integrity tags of each chunk reaches the predefined threshold value, they are aggregated instantly before being deleted from the cloud storage. As is shown in Figures 1(a) and 1(b), if tag aggregation is not triggered, the cloud user has to check the integrity of his/her data by interacting with the CSP. Otherwise, it is able to delegate the auditing task to a third-party auditor, as is shown in Figure 1(c). During the checking process, the CSP is obligated to generate a proof based on the challenge received.

TPA is an entity who has expertise and capabilities to help users check the integrity of their data stored in the CSP. According to our design, integrity auditing is only appointed to the TPA once tags in the CSP have been aggregated. Together with auditing delegation, the cloud user has to outsource a corresponding content-associated public key for the specific file to the TPA simultaneously, which is utilized in subsequent integrity verification. Finally, TPA returns the verification result to the user as response.

3.2. *Threat Model.*

According to the system model introduced above, data confidentiality and integrity are most essential security requirements for the scenario of data outsourcing. Under public auditing, our primary security goal is to protect the existence privacy of data. We consider both inside and outside adversaries in the threat model. The inside one is the CSP, who is curious about the content of user's data, even though it provides storage and computation service for most of the time. More seriously, it may tamper or corrupt user's data for its own benefits. The outside one is an adversary who is able to establish a side channel to the CSP. It firstly predicts the content of predictable data in cloud storage by launching brute-force dictionary attack and then learns the ownership of predicted data by performing integrity verification with the public key of corresponding user. Specifically, the outside adversary may be a malicious user or the TPA who is curious about the existence privacy of a certain cloud user, both of which are in possession of public key of the target user.

3.3. Design Goals. To enable secure and efficient cloud storage as well as side channel attack resistant public auditing under the aforementioned system model and threat model, the design of our protocol should achieve the following security and performance goals:

(1) Data confidentiality: to ensure the inside adversary cannot derive the content of plaintext from the data stored in cloud storage.
(2) Private and public auditability: to ensure the data owner or his/her delegated TPA to verify the correctness of his/her data by running the challenge-response protocol. During the process of verification, data download is not necessary.
(3) Side channel attack resistance: to ensure the outside adversary cannot acquire existence privacy for specific data from a certain cloud user by launching side channel attack during the process of public verification. To achieve such a security goal, the user-associated public key should be eliminated from the process of verification.
(4) Lightweight: to allow cloud users to generate verification information with minimum computation and communication overhead. And to ensure the overhead of public verification is also minimized.

4. Proposed Scheme

4.1. Preliminaries

4.1.1. Convergent Encryption. CE [18] is an encryption scheme to achieve cross-user deduplication of cloud data. It generates an encryption key based on the content of data and encrypts the plaintext with the key in the form of symmetric encryption. A standard CE algorithm contains the following three basic algorithms.

(1) $KeyGen(F, 1^{\lambda}) \longrightarrow CEK$: a deterministic key generation algorithm takes as input a file $F \in \{0,1\}^*$ and a security parameter $\lambda \in \mathbb{N}$. The algorithm outputs a convergent key CEK, which is obtained by calculating $CEK \leftarrow h(F)$, where $h(\cdot)$ is a cryptographic hash function.
(2) $CEEnc(F, CEK) \longrightarrow C$: a deterministic symmetric encryption algorithm takes as input a file F as well as the corresponding convergent key CEK and outputs a ciphertext C, which is independent of the user to perform encryption.
(3) $CEDec(C, CEK) \longrightarrow F$: a deterministic decryption algorithm takes as input ciphertext C as well as the corresponding convergent key CEK and generates the plaintext F as output.

4.1.2. Bilinear Map. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of the same prime order p and g be a generator of group $\mathbb{G}_1$. A bilinear map is defined as e: $\mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$ with the following properties:

(1) Bilinearity: for all $g_1, g_2 \in \mathbb{G}_1$ and any $a, b \in \mathbb{Z}_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$;
(2) Nondegeneracy: $e(g, g) \neq 1$;
(3) Computability: for every $g_1, g_2 \in \mathbb{G}_1$, $e(g_1, g_2)$ can be calculated efficiently.

4.1.3. Lagrangian Interpolation. The basic principle of Lagrange interpolation is to obtain a certain polynomial whose curve passes through a set of given points. Suppose there are k points $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$, each of which is different from the others. The Lagrange interpolation polynomial can be defined as $L(x) = \sum_{i=1}^{k} y_i \cdot \lambda_i(x)$, where $\lambda_i(x)$ is the interpolation coefficient, which is expressed as

$$\lambda_i(x) = \prod_{i=1, i\neq j}^{k} \frac{x - x_j}{x_i - x_j}. \quad (1)$$

4.2. Scheme Overview. The design of ATDS includes both efficiency and security considerations. Our protocol supports deduplication of ciphertext chunks and their integrity tags simultaneously and is able to resist side channel attacks launched by the outside adversary during the process of integrity auditing. Specifically, to achieve tag aggregation, the CSP has to maintain a threshold value t, together with a counter for integrity tags in storage. Once the number of integrity tags for a certain chunk reaches t, tag aggregation is triggered instantly, followed by deduplication. In our design, each user is assigned a unique index UID, which is only known to the user himself/herself and the CSP. When a cloud user attempts to outsource ciphertext chunks of file F together with the corresponding tags, he/she first sends a unique index *fid* (usually a hash value of the file) of file F as an uploading request to the CSP. If the ciphertext of file F has not been uploaded, the user needs to upload the ciphertext chunks, integrity tags, secondary tags, as well as the auxiliary information. Otherwise, it only needs to upload integrity tags, secondary tags for ciphertext chunks, or aggregated verification tag for file F depending on whether tag aggregation has been triggered in cloud storage. In a word, the CSP only keeps one copy of ciphertext chunks and auxiliary information for a certain file.

The overall process is shown in Figure 2. Consider user u_1 as the first uploader for the ciphertext of file F. It first derives

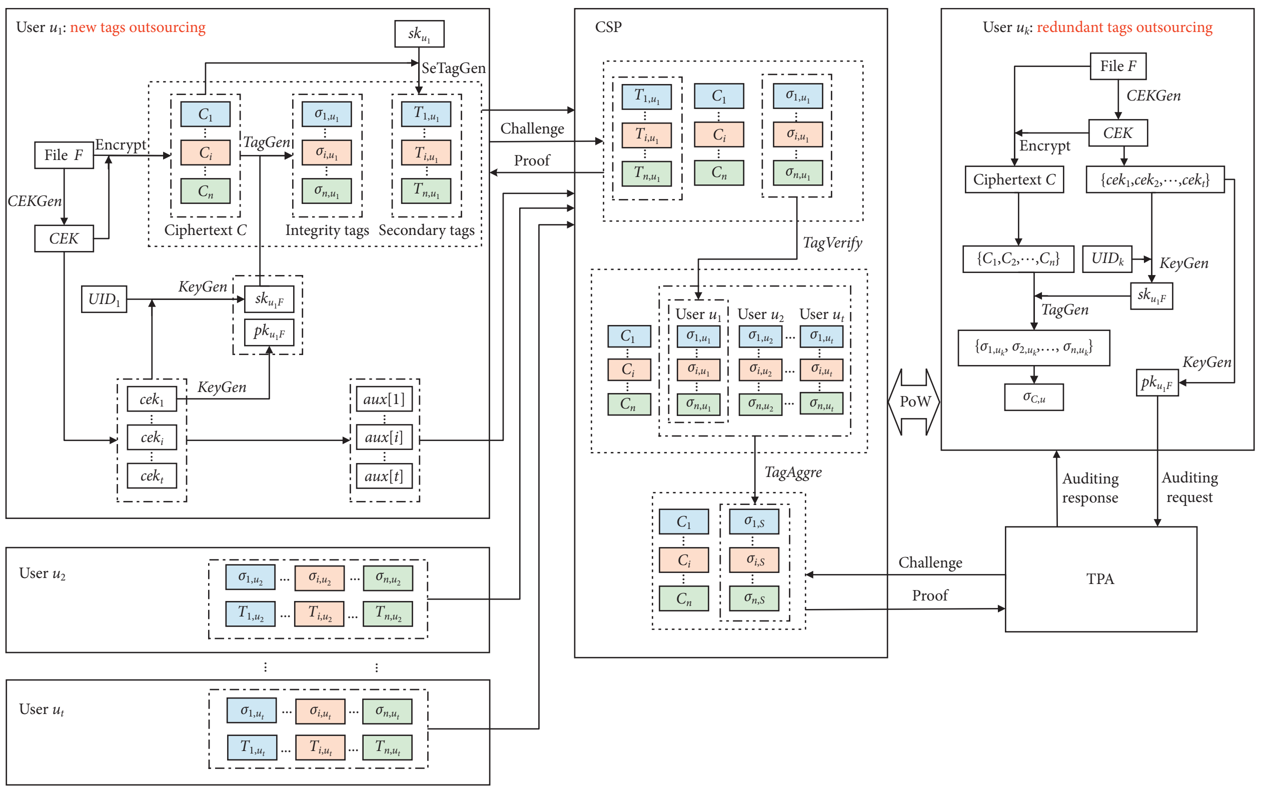


FIGURE 2: The architecture of our proposed scheme.

the content-associated key CEK from file F, which is utilized as the encryption key to generate ciphertext C. Then, the user divides C into n chunks and CEK into t pieces of the same size. Each one of the pieces is employed as coefficients of the defined polynomial to generate the user-associated private key sk_{u_1F}, which is utilized to generate tags $\{\sigma_{1,u_1}, \sigma_{2,u_1}, \ldots, \sigma_{n,u_1}\}$ for ciphertext chunks $\{C_1, C_2, \ldots, C_n\}$ of file F. Specifically, UID_1 of user u_1 is also combined in the process of key generation. Moreover, the user also generates a set of secondary tags $\{T_{1,u_1}, T_{2,u_1}, \ldots, T_{n,u_1}\}$ with the help of his/her own secret key, which are utilized in further private verification before tag aggregation is triggered. In addition, a series of auxiliary information is also generated correspondingly. Note that to generate the public content-associated key, only cek_1 is needed. Finally, user u_1 uploads his/her ciphertext chunks, integrity tags, secondary tags, and auxiliary information to CSP.

On receiving the uploaded information from u_1, CSP first needs to verify the correctness of integrity tags to ensure that only qualified tags would be stored in cloud storage and participate in the subsequent aggregation. For each one of these tags, its corresponding counter is also increased by 1. Similarly, integrity tags uploaded by subsequent users (u_2, $u_3, \ldots, u_t$) also need to be verified for correctness, and once the number for a specific tag indicated by the counter reaches the predefined threshold value t, tag aggregation would be triggered instantly. In this case, the subsequent user u_k only needs to upload aggregated verification tag for file F to prove his/her ownership.

According to the procedure described above, we consider two different situations to achieve integrity auditing. For the first one, the integrity tags in cloud storage have not been aggregated. So integrity of the target file is only able to be verified by the cloud user with the help of secondary tags and existence privacy would not be exposed to others. For the second one, the aggregated tag has been generated. In this case, the task of integrity auditing is delegated to the TPA. Take user u_k, for example, it needs to send the public content-associated key $pk_{u,F}$ to TPA together with integrity auditing delegation, which is utilized in the following verification to achieve side channel attack resistance.

4.3. Algorithm Formulation. In this subsection, we provide the formal definitions of basic algorithms in our construction as follows:

(i) $CEKGen\ (F) \longrightarrow (CEK,\ \{aux[i]\})$: a deterministic algorithm is run by a cloud user. It takes as input file F and outputs a content-associated key CEK as well as a set of auxiliary information $\{aux[1], aux[2], \ldots,\ aux[t]\}$, where $aux[i] \in \mathbb{G}_1$ for $i \in [1, t]$. Specifically, CEK can be further divided into $\{cek_1, cek_2, \ldots, cek_t\}$, where $cek_i \in \mathbb{Z}_p$ for $i \in [1, t]$.

(ii) $Encrypt(F, CEK) \longrightarrow \{C_i\}$: a deterministic algorithm is run by a cloud user. The algorithm takes as input a file F as well as the corresponding convergent key CEK and generates a ciphertext C, which can be divided into a set of ciphertext chunks $\{C_1, C_2, \ldots, C_n\}$ of the same length as output, where $C_i \in \mathbb{Z}_p$ for $i \in [1, n]$.

(iii) *KeyGen*($\{cek_i\}$, *UID*)$\longrightarrow$(sk_{uF}, pk_{uF}) : a deterministic algorithm is run by a cloud user. It takes as input the *CEK* set $\{cek_i\}, i \in [1, t]$ of file F and the index *UID* of user u and outputs an user-associated private key sk_{uF} and a content-associated public key pk_{uF} for file F.

(iv) *SEKGen*(1^k)$\longrightarrow$(sk_u): a probabilistic algorithm run by cloud user u takes the security parameter k as input and outputs a secret key sk_u of the user.

(v) *TagGen*($\{C_i\}$, sk_{uF}) $\longrightarrow$($\{\sigma_{i,u}\}$): a deterministic algorithm is run by a cloud user. It takes as input ciphertext chunks $\{C_i\} i \in [1, n]$, and user-associated private key sk_{uF} for file F. Then, the algorithm outputs the signature set $\{\sigma_{i,u}\}, i \in [1, n]$, which is an ordered collection of tags on $\{C_i\}, i \in [1, n]$.

(vi) *SeTagGen*($\{C_i\}$, sk_u) $\longrightarrow$($\{T_{i,u}\}$): a deterministic algorithm is run by a cloud user. It takes as input ciphertext chunks $\{C_i\} i \in [1, n]$ and secret key sk_u of cloud user u. Then, the algorithm outputs a set of secondary tags $\{T_{i,u}\}$, $i \in [1, n]$, each of which is signed by sk_u.

(vii) *TagVerify*($\{\sigma_{i,u}\}/\sigma_{C,u}$, $\{C_i\}$, *UID*, $\{aux[i]\}$)$\longrightarrow$($\{$TRUE, FALSE$\}$): a deterministic algorithm is run by the CSP to check correctness of the received integrity tags $\{\sigma_{i,u}\}, i \in [1, n]$ or their aggregated version $\sigma_{C,u}$ for the ciphertext C. The algorithm takes as input $\{\sigma_{i,u}\}$ or $\sigma_{C,u}$ generated by user u, their corresponding ciphertext chunks $\{C_i\}$, the unique index *UID* of the user and auxiliary information set $\{aux[i]\}, i \in [1, t]$, and outputs TRUE if tags $\{\sigma_{i,u}\}, i \in [1, n]$ or $\sigma_{C,u}$ are verified as correct or FALSE otherwise.

(viii) *TagAggre*($\{\sigma_{i,u_j}\}$, $\{\lambda_j\}$)$\longrightarrow$($\sigma_{i,S}$): an aggregation algorithm run by CSP takes as input t tags $\{\sigma_{i,u_j}\}, j \in [1, t]$, for a certain ciphertext chunk C_i as well as a Lagrangian interpolation coefficient set $\{\lambda_j\}, j \in [1, t]$, and outputs an aggregated signature $\sigma_{i,S}$ for C_i.

(ix) *ProofGen*(C, Φ, chal) $\longrightarrow$ (P): this algorithm is run by the CSP. It takes as input the ciphertext C for a certain target file, its signatures Φ, and a challenge chal. It outputs an integrity proof P for the chunks specified by chal.

(x) *ProofVerify*(key, chal, P)$\longrightarrow$($\{$TRUE, FALSE$\}$): a verification algorithm is run by either the cloud user or the TPA to check the correctness of proof P. It takes as input the challenge chal, the proof P returned from the CSP, and the verification key key, and outputs TRUE or FALSE according to the verification result. Specifically, once the aggregated tags for a certain file are existed in the cloud, the algorithm is performed by the TPA, and in this case, the input verification key is pk_{uF}. Otherwise, this algorithm is performed by the cloud user and the verification key is sk_u.

4.4. Construction of ATDS. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be multiplicative cyclic groups of prime order p, which is a sufficient large prime number, and g be the generator of $\mathbb{G}_1$. Let e: $\mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$ denote a bilinear map. $H(\cdot)$: $\{0,1\}^* \longrightarrow \mathbb{G}_1$ is defined as a secure map-to-point hash function and $h(\cdot)$: $\{0,1\}^* \longrightarrow \mathbb{Z}_p$ a cryptographic one, both of which are chosen and published by cloud users. The system public parameter set is $S = (p, \mathbb{G}_1, \mathbb{G}_2, e, g, H, h)$.

4.4.1. Setup Phase. Consider a cloud user u wants to outsource a file F, and it firstly chooses a random number $sk_u \in \mathbb{Z}_p$ as secret key by running *SEKGen*. Next, the user runs *CEKGen* to generate the content-associated key by calculating $CEK = h(F)$. Then, further divide *CEK* into a set $\{cek_1, cek_2, \ldots, cek_t\}$ and generate a corresponding auxiliary information $aux[i] = g^{cek_i} \in \in \mathbb{G}_1, i \in [1, t]$, for each one of the $cek_i \in \mathbb{Z}_p, i \in [1, t]$, involved in the set. As a follow-up work, the user runs *Encrypt* to generate a ciphertext C for file F, using *CEK* as the convergent key. It divides the ciphertext C into a set of ciphertext chunks $\{C_1, C_2, \ldots, C_n\}$ of the same length, where $C_i \in \mathbb{Z}_p$ for $i \in [1, n]$.

The last part of setup phase is to generate a user-associated private key and a content-associated public key for the given file F, together with a random element $x_u \in \mathbb{Z}_p$ as secret key sk_u by running *KeyGen* and *SEKGen*, respectively. Based on the *CEK* set $\{cek_1, cek_2, \ldots, cek_t\}$, the cloud user generates a polynomial $f(x) = cek_1 + cek_2 x + cek_3 x^2 + \cdots + cek_t x^{t-1} \pmod p$, by defining $cek_i, i \in [1, t]$ as coefficients. It is worth mentioning that, in IPANM [3], owners of file F generate random coefficients of the polynomial independently. However, in ATDS, different owners share the same coefficients cek_i. Moreover, in order to ensure the correctness, integrity tags are verified by the CSP before aggregation instead of the user himself/herself, both of which eliminate the expensive interaction between data owners in an open deduplication system. Subsequently, a specific user u_j with user id UID_j calculates his/her user-associated secret key $sk_{u_j F}$ by

$$f(UID_j) = cek_1 + cek_2 UID_j + cek_3 UID_j^2 + \cdots + cek_t UID_j^{t-1} \pmod p. \tag{2}$$

Then, it defines the content-associated public key as

$$pk_{u_j F} = g^{cek_1}. \tag{3}$$

4.4.2. Tag Deduplication Phase. The cloud user u first runs *TagGen* to generate the tag set $\{\sigma_{i,u}\}, i \in [1, n]$, for ciphertext chunks $\{C_i\}, i \in [1, n]$, corresponding to a specific file F. The key point of this procedure is that the user-associated private key defined in the setup phase is utilized instead of the traditional private key of the cloud user. Given a certain ciphertext chunk $C_i \in \mathbb{Z}_p, i \in [1, n]$, the corresponding tag is calculated by

$$\sigma_{i,u} = \left(H(i) \cdot g^{C_i}\right)^{sk_{uF}}. \tag{4}$$

In the definition above, $H(i)$ denotes the hash value of the index for a certain ciphertext chunk C_i, which ensures the distinguishability of tags even though the corresponding content of chunks is identical. Then, the user runs *SeTagGen* to generate secondary tags for ciphertext chunks $\{C_i\}, i \in [1, n]$, as

$$T_{i,u} = \left(H(i) \cdot g^{C_i}\right)^{sk_u}. \quad (5)$$

Finally, the cloud user generates a set of integrity authenticators $\{\sigma_{i,u}\}, i \in [1, n]$, and a set of secondary tags $\{T_{i,u}\}, i \in [1, n]$, and outsources them as well as the auxiliary information to the CSP together with the deduplication checking request for the ciphertext of file F. On receiving them, the CSP first runs *TagVerify* to check the correctness of $\{\sigma_{i,u}\}, i \in [1, n]$. In order to reduce the computational overhead during verification, the CSP aggregates n integrity tags from the specific user u to generate a verification tag $\sigma_{C,u}$, which is defined as $\sigma_{C,u} \underset{=}{def} \prod_{i=1}^{n} \sigma_{i,u} = \prod_{i=1}^{n} (H(i) \cdot g^{C_i})^{sk_{uF}}$, and calculates $\mu = \sum_{i=1}^{n} C_i$. It is worth mentioning that once tag aggregation has been triggered in cloud storage, $\sigma_{C,u}$ is generated by user u himself/herself. The correctness of $\sigma_{C,u}$ and μ can be verified by checking

$$e\left(\sigma_{C,u}, g\right) = e\left(\prod_{i=1}^{n} H(i) \cdot g^{\mu}, aux[1] \cdot aux[2]^{UID} \cdot aux[3]^{UID^2} \ldots aux[t]^{UID^{(t-1)}}\right). \quad (6)$$

If so, the outsourced tags $\{\sigma_{i,u}\}, i \in [1, n]$ are considered to be correct. Then, the user u is deemed as a qualified one. Denote the group of qualified users by the set SG. Once at least t different users in SG outsourced correct tags for the same target file, chunk-level tag aggregation would be triggered by running *TagAggre*. Specifically, for a certain ciphertext chunk C_i, once t correct tags from different users in SG are collected, CSP generates the aggregated tag by calculating

$$\sigma_{i,S} = \prod_{u_j \in SG, j \neq i} \sigma_{i,u_j}^{\lambda_j} = \prod_{j=1}^{t} \sigma_{i,u_j}^{\lambda_j} = \prod_{j=1}^{t} \left(H(i) \cdot g^{C_i}\right)^{sk_{u_jF}\lambda_j}, \quad (7)$$

where $\lambda_j = \prod_{u_j \in SG, j \neq i} (-UID_j)/(UID_i - UID_j)$ is a Lagrangian interpolation coefficient and UID_j is the unique user ID of the participated signer u_j. Finally, the aggregated tags $\{\sigma_{1,S}, \sigma_{2,S}, \ldots, \sigma_{n,S}\}$ for each one of the ciphertext chunks $C_i, i \in [1, n]$ are stored in the CSP, instead of more than t pieces of original tags. We present the formal description of ATDS algorithm in Algorithm 1.

4.4.3. Integrity Auditing Phase. When a cloud user attempts to check the integrity of his/her outsourced ciphertext C corresponding to file F, he/she first needs to send a query request to the CSP to check whether the corresponding tags in the CSP have been aggregated or not. If yes, in order to reduce the computational overhead of the client side, the user delegates the integrity auditing task to the TPA. Otherwise, the user has to perform integrity verification by themselves to protect the existence privacy of his/her files in cloud storage.

For the first case, suppose user u tries to check the integrity of ciphertext C, he/she first sends an integrity auditing delegation together with his/her content-associated public key pk_{uF} to the TPA. On receiving the request, the TPA picks a random c-element subset $I = \{s_1, s_2, \cdots, s_c\}$ and chooses a random value v_i for each element $i \in I$. Then, it sends the challenge chal= $\{(i, v_i)\}_{i \in I}$ to the CSP. Upon receiving the challenge, the CSP runs *ProofGen* to generate a corresponding proof $P = \{\sigma, \rho\}$ and returns to the TPA, where $\sigma = \prod_{i=s_1}^{s_c} \sigma_{i,S}^{v_i} \in \mathbb{G}_1$ and $\rho = \sum_{i=s_1}^{s_c} v_i C_i \in \mathbb{Z}_p$.

The proof P can be verified by running *ProofVerify* in the way as follows:

$$e(\sigma, g) = e\left(\prod_{i=s_1}^{s_c} H(i)^{v_i} \cdot g^{\rho}, pk_{uF}\right). \quad (8)$$

For the second case, the cloud user generates the challenge chal $= \{(i, v_i)\}_{i \in I}$ in the same way as which is mentioned above and sends it to the CSP. Similarly, the CSP generates a proof $P = \{T_{u_j}, \rho\}$ and returns to the user, where $T_{u_j} = \prod_{i=s_1}^{s_c} T_{i,u_j}^{v_i} \in \mathbb{G}_1$ and $\rho = \sum_{i=s_1}^{s_c} v_i C_i \in \mathbb{Z}_p$. The user runs *ProofVerify* to check the correctness of the proof as follows:

$$e\left(T_{u_j}, g\right) = e\left(\prod_{i=s_1}^{s_c} H(i)^{v_i} \cdot g^{\rho}, g^{sk_{u_j}}\right). \quad (9)$$

We present the formal description of aggregation of ATDS, as shown in Algorithm 1.

5. Security Analysis

In this section, we analyze the security of ATDS against both inside and outside adversaries under the threat model defined in Section 3. Specifically, our analysis starts from the effectiveness of integrity auditing and then focuses on security against side channel attack during the process of public auditing.

Theorem 1. *After receiving all integrity tags of target file F outsourced by user u, CSP is able to generate a verification tag $\sigma_{C,u}$ that passes the verification if all the integrity tags are correct.*

Proof. For a target file F, user u generates his/her CE ciphertext $C = \{C_1, C_2, \ldots, C_n\}$ and then uploads them together with corresponding integrity tags $\{\sigma_{1,u}, \sigma_{2,u}, \ldots, \sigma_{n,u}\}$ to the cloud storage. Specifically, a certain integrity tag $\sigma_{i,u}$ for chunk C_i is defined as $(H(i) \cdot g^{C_i})^{sk_{uF}}$, and the user-

```
     Input: tags $\sigma_{i,u_j} = (H(i) \cdot g^{C_i})^{sk_{u_jF}}$ $(i \in [1,n], j \in [1,m])$.
     Output: aggregated tags $\sigma_{i,S}$;
(1)  $SG = \varnothing$;
(2)  $\mu = \sum_{i=1}^{n} C_i$;
(3)  for $j = 1$ to $m$ do
(4)      $\sigma_{c,u_j} = \sum_{i=1}^{n} \sigma_{i,u_j}$;
(5)      if $e(\sigma_{c,u_j}, g) == e(\prod_{i=1}^{n} H(i) \cdot g^{\mu}, aux[1] \cdot aux[2]^{UID}, \ldots, aux[t]^{UID^{(t-1)}})$ then
(6)          store integrity tags $\sigma_{i,u_j}$;
(7)          store secondary tags $T_{i,u_j}$
(8)          $SG = SG \cup \{u_j\}$;
(9)      else
(10)         return error
(11)     endif
(12) endfor
(13) if $|SG| > = t$ then
(14)     for $i = 1$ to $n$ do
(15)         $\sigma_{i,S} = \prod_{u_j \in SG, j=1, j \neq i}^{t} \sigma_{i,u_j}^{\lambda_j}$;
(16)     endfor
(17) endif
```

Algorithm 1: Aggregation algorithm of ATDS.

associated secret key sk_{uF} is denoted as $f(UID)$. The CSP first generates a verification tag $\sigma_{C,u} = \prod_{i=1}^{n} \sigma_{i,u} = \prod_{i=1}^{n} (H(i) \cdot g^{C_i})^{sk_{uF}}$ and then verifies its correctness by checking the bilinear equation (6). A special case is that once tag aggregation has been triggered by the CSP, $\sigma_{C,u}$ is generated by user u himself/herself. Based on the properties of bilinear maps, the correctness can be deduced from the following:

$$\begin{aligned} e(\sigma_{C,u}, g) &= e\left(\prod_{i=1}^{n} \sigma_{i,u}, g\right) = e\left(\prod_{i=1}^{n} \left(H(i) \cdot g^{C_i}\right)^{sk_{uF}}, g\right), \\ &= e\left(\prod_{i=1}^{n} \left(H(i) \cdot g^{C_i}\right)^{cek_1 + cek_2 UID + cek_3 UID^2 + \cdots + cek_t UID^{t-1}}, g\right), \\ &= e\left(\prod_{i=1}^{n} H(i) \cdot g^{\sum_{i=1}^{n} C_i}, g^{cek_1 + cek_2 UID + cek_3 UID^2 + \cdots + cek_t UID^{t-1}}\right), \\ &= e\left(\prod_{i=1}^{n} H(i) \cdot g^{\mu}, aux[1] \cdot aux[2]^{UID} \cdot aux[3]^{UID^2}, \ldots, aux[t]^{UID^{(t-1)}}\right). \end{aligned} \tag{10}$$

Theorem 2. *For a target file F, if the number of integrity tags for each chunk does not reach the threshold value t, each valid user is able to correctly check the integrity of file F by launching a challenge-response protocol.*

Proof. According to Theorem 1, the CSP only keeps correct tags in storage, the number of which is identical for each chunk involved in a certain file. For a certain target file, if the tag counter does not reach the predefined threshold value t, the cloud user has to verify the proof generated by CSP. Specifically, cloud user sends a random challenge chal $= \{(i, v_i)\}_{i \in I}$ to CSP and receives a proof $P = \{T_{u_j}, \rho\}$, which can be verified by checking the bilinear equation (9). Similar to the previous verification, the correctness can be deduced from the following:

$$\begin{aligned} e(T_{u_j}, g) &= e\left(\prod_{i=s_1}^{s_c} T_{i,u_j}^{v_i}, g\right), \\ &= e\left(\prod_{i=s_1}^{s_c} \left(H(i) \cdot g^{C_i}\right)^{sk_{u_j} v_i}, g\right) \\ &= e\left(\prod_{i=s_1}^{s_c} H(i)^{v_i} \cdot g^{\sum_{i=s_1}^{s_c} v_i C_i}, g^{sk_{u_j}}\right) \\ &= e\left(\prod_{i=s_1}^{s_c} H(i)^{v_i} \cdot g^{\rho}, g^{sk_{u_j}}\right). \end{aligned} \tag{11}$$

Theorem 3. *TPA is able to verify integrity of the target file F by checking proof P returned from the CSP if the aggregated tags have been generated.*

Proof. Once the number of the tags for a certain chunk reaches the threshold value t, tag aggregation would be triggered instantly and the aggregated tag is obtained based on equation (6). It is worth mentioning that, in this case, aggregated tags for each chunk of the target file are generated simultaneously. Therefore, the cloud user is able to delegate the auditing task to TPA. During the verification, TPA picks a random c-element subset $I = \{s_1, s_2, \ldots, s_c\}$ and chooses a random value v_i for each element $i \in I$ to generate a challenge $\text{chal} = \{(i, v_i)\}_{i\in I}$ before sending to CSP. Then, TPA receives proof $P = \{\sigma, \rho\}$, where $\sigma = \prod_{i=s_1}^{s_c} \sigma_{i,S}^{v_i}$ and $\rho = \prod_{i=s_1}^{s_c} v_i C_i$. The proof P generated based on aggregated tags can be verified by checking the bilinear equation (8). According to the principle of Lagrangian interpolation, the aggregated tag in equation (8) can be denoted as $\sigma_{i,S} = (H(i) \cdot g^{C_i})^{cek_1}$. Therefore, the correctness of the bilinear equation can be deduced as follows:

$$\begin{aligned} e(\sigma, g) &= e\left(\prod_{i=s_1}^{s_c} \sigma_{i,S}^{v_i}, g\right), \\ &= e\left(\prod_{i=s_1}^{s_c} \left(H(i) \cdot g^{C_i}\right)^{cek_1 v_i}, g\right) \\ &= e\left(\prod_{i=s_1}^{s_c} H(i)^{v_i} \cdot g^{\sum_{i=s_1}^{s_c} C_i v_i}, g^{cek_1}\right) \\ &= e\left(\prod_{i=s_1}^{s_c} H(i)^{v_i} \cdot g^{\rho}, pk_{uF}\right). \end{aligned} \qquad (12)$$

Theorem 4. *The TPA cannot steal the existence privacy of a specific cloud user by launching side channel attack during the process of public auditing.*

Proof. Based on the threat model defined in Section 3, once the TPA is interested in the existence privacy of a specific cloud user, it first predicts the content of data by launching brute-force dictionary attack. It is worth mentioning that such kind of attack is always feasible for predictable data in cloud storage. For a predicted file F, the TPA encrypts it by convergent encryption and then divides the ciphertext into n chunks $\{C_1, C_2, \ldots, C_n\}$ of the same length, where $C_i \in \mathbb{Z}_p$ for $i \in [1, n]$. To determine existence of the ciphertext in cloud storage, the TPA has to generate a challenge $\text{chal} = \{(s_i, v_i)\}_{i\in I}$ and send to the CSP. According to our proposed scheme, the CSP generates a proof $P = \{\sigma, \rho\}$ based on the challenge and returns to the TPA, where $\sigma = \prod_{i=s_1}^{s_c} \sigma_{i,S}^{v_i} \in \mathbb{G}_1$ and $\rho = \sum_{i=s_1}^{s_c} v_i C_i \in \mathbb{Z}_p$. Based on equation (7), each one of $\sigma_{i,S}$ in σ corresponds to t correct tags from different users. Moreover, the public key pk_{uF} utilized in bilinear equation (8) is a content-associated key, rather than a user-associated one. Therefore, once the correctness of the proof is verified by checking the bilinear equation (8), the TPA is only able to know that the ciphertext is in possession of at least t users but cannot determine which specific user owns the target file. By setting the threshold value elaborately, once existence is known to the TPA, the target file has been a popular one. Thus, there is no risk of privacy leakage.

Theorem 5. *The existence privacy for a certain target user is secure even though his/her UID is exposed to an outside adversary.*

Proof. Suppose the *UID* of an ordinary user is exposed to an outside adversary, who generates a correct version of file F owned by this user by launching dictionary attack. The adversary is able to calculate the user-associated secret key sk_{uF} according to formula (2). Then, the adversary attempts to verify the linkage between F and the user through challenge-response protocol with the help of the key.

Firstly, the outside adversary needs to check whether integrity tags for each chunk of F have been aggregated. If yes, the existence privacy of the user would not be compromised according to Theorem 4. Otherwise, if secondary tags are not employed, the adversary has to generate a random challenge $\text{chal} = \{(i, v_i)\}_{i\in I}$ and verify the returned proof $P = \{\sigma_{u_j}, \rho\}$ by checking whether $e(\sigma_{u_j}, g) = e(\prod_{i=s_1}^{s_c} H(i)^{v_i} \cdot g^{\rho}, g^{sk_{uF}})$, where $\sigma_{u_j} = \prod_{i=s_1}^{s_c} \sigma_{i,u_j}^{v_i} \in \mathbb{G}_1$ and $\rho = \sum_{i=s_1}^{s_c} v_i C_i \in \mathbb{Z}_p$. Once the verification is passed, the existence privacy is exposed instantly. To deal with this problem, we introduce the secondary tag to ATDS, which is able to achieve integrity verification before tag aggregation is triggered according to Theorem 2. As a result, the adversary cannot obtain the existence privacy of the certain target user during the process of verification any more.

6. Performance Analysis and Evaluation

In order to evaluate the performance, we take experiments in this section to compare the storage overhead, computation overhead, and communication overhead of proposed ATDS with the other two state-of-the-art schemes: one-tag checker [2] and IPANM [3]. Specifically, we employ Amazon Elastic Computing Cloud (EC2) instances to implement the process of CSP, which is dedicated to provide storage and tag deduplication service to clients. Moreover, we implement both the client and TPA side processes using pairing-based cryptography (PBC) library version 0.5.14 on workstations with Intel Core i7-6700 CPU@3.40 GHz, 4 GB RAM, and a 7200 RPM 1 TB hard drive in Ubuntu 15.5 operating system. The security level is chosen to be 160 bits, and all results are on the average of 20 tries.

6.1. Storage Overhead. In this part, we compare ATDS with one-tag checker [2] in cloud side storage overhead in the scenario of tag deduplication. Specifically, the overhead is evaluated in the form of deduplication rate. For simplicity, we assume convergent ciphertext chunks of the target file have been stored in the cloud, and the number of which is denoted as n. Suppose the predefined deduplication threshold value for integrity tags is t. In our evaluation, we denote the length of an element in multiplicative cyclic group $\mathbb{G}_1$ with prime order p as $|\mathbb{G}_1|$ and that in $\mathbb{Z}_p$ as $|\mathbb{Z}_p|$.

For a certain target file, in our proposed ATDS, the first uploader of integrity tags needs to outsource secondary tags and auxiliary information for n chunks, as well as the storage overhead of which is $2n|\mathbb{G}_1| + t|\mathbb{Z}_p|$ in total. For successive uploaders of integrity tags, they outsource integrity tags, secondary tags, or just only integrity tags depending on the situation. It is worth mentioning that if the number of tags for a certain chunk does not reach the threshold value, the integrity tags are used for correctness verification before storing in cloud storage. At the same time, secondary tags are needed for further private verification. Otherwise, once the threshold value is reached, only aggregated verification tag for the target file is needed, which is utilized in proof of ownership and then deleted from the storage. Meanwhile, tag deduplication is triggered so that the storage overhead drops down from $(2n \cdot t)|\mathbb{G}_1| + t|\mathbb{Z}_p|$ to $n|\mathbb{G}_1| + t|\mathbb{Z}_p|$. In this case, the number of uploaded tags just reaches the threshold value t. Even though the following users continue to upload tags for the same chunk, the storage overhead remains constant since they would not be stored at all. Thus, a deduplication rate of $1 - ((n|\mathbb{G}_1| + t|\mathbb{Z}_p|)/(2nt|\mathbb{G}_1| + t|\mathbb{Z}_p|))$ is obtained.

As for one-tag checker [2], the first uploader of a certain target file needs to outsource n integrity tags as well as a rekey, each of which with a storage overhead of $n|\mathbb{G}_1|$ and $2|\mathbb{Z}_p|$, respectively. Even though the concept of threshold value is absent in one-tag checker, we define the number of uploaders to be t in order to keep the comparison in the same level. For t uploaders, the overhead can be denoted as $nt|\mathbb{G}_1| + 2t|\mathbb{Z}_p|$. Thanks to the deduplication scheme employed, subsequent uploaders only need to upload the rekey. Thus, the storage cost drops to $n|\mathbb{G}_1| + 2t|\mathbb{Z}_p|$, and a deduplication rate of $1 - ((n|\mathbb{G}_1| + 2t|\mathbb{Z}_p|)/(nt|\mathbb{G}_1| + 2t|\mathbb{Z}_p|))$ is achieved.

In order to verify the above analysis, we compare both schemes in deduplication rate. Specifically, we set the threshold value t to be 20 and 40 and divide the target file into equal-sized chunks with number from 300 to 1100. The results are presented in Figures 3(a) and 3(b), respectively. Take Figure 3(a), for example, the threshold value t is set to be 20. When the number of ciphertext chunks increases from 300 to 1100, the deduplication rate of our proposed scheme is always higher than 0.97, which is obviously better than that of one-tag checker since the cost of rekey is not needed in our scheme. The similar result is shown in Figure 3(b).

Moreover, we present the storage overhead of our proposed scheme in Figure 4, in which Y axis is drawn as the logarithmic coordinate. In this figure, the target convergent ciphertext is divided into 300 and 500 chunks, and the threshold value t is set to be 20 and 40, respectively. Consider the case of $n = 300$ and $t = 20$, when the number of users increases from 0 to 20, the overhead of our proposed scheme increases from 3200 to 1,923,200, which drops rapidly to 51,200 once the number continues to increase since tag aggregation has been triggered. To achieve side channel attack resistance in an open deduplication system, we introduce secondary tags to ATDS, which bring about 48,000 bits extra storage overhead for each user before the number of users reaches the threshold value t. Similar results are obtained for other parameter combinations.

6.2. Computational Overhead.

6.2. Computational Overhead. We consider the computational overhead during the process of public verification in this section and still compare our ATDS with one-tag checker [2]. Specifically, the running time during both processes of proof generation and proof verification is evaluated. For simplicity, we use $\mathrm{Exp}_{\mathbb{Z}_p}$, $\mathrm{Mul}_{\mathbb{Z}_p}$, $\mathrm{Exp}_{\mathbb{G}_1}$, and $\mathrm{Mul}_{\mathbb{G}_1}$ to denote exponential operation and multiplication operation in $\mathbb{Z}_p$ and $\mathbb{G}_1$, respectively. Similarly, $\mathrm{Add}_{\mathbb{Z}_p}$ is employed to represent additional operation in $\mathbb{Z}_p$. In addition, Pair denotes bilinear pairing operation on $e\colon \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$, and $\mathrm{Hash}_{\mathbb{G}_1}$ represents hash operation $H(\cdot)\colon \{0,1\}^* \longrightarrow \mathbb{G}_1$.

In the process of proof generation, on receiving the challenge from the TPA, the CSP generates a proof $P = \{\sigma, \rho\}$ accordingly. Denote the number of challenged chunks to be c. As is shown in Table 1, for the proposed scheme, the computational overhead of proof generation is $c\mathrm{Exp}_{\mathbb{G}_1} + (c-1)\mathrm{Mul}_{\mathbb{G}_1} + c\mathrm{Mul}_{\mathbb{Z}_p} + (c-1)\mathrm{Add}_{\mathbb{Z}_p}$, which is obviously less than $2c\mathrm{Exp}_{\mathbb{G}_1} + 2(c-1)\mathrm{Mul}_{\mathbb{G}_1} + (2c+s)\mathrm{Mul}_{\mathbb{Z}_p} + s(c-1)\mathrm{Add}_{\mathbb{Z}_p}$ in one-tag checker. The reason is that, in Liu et al.'s scheme, the CSP has to execute the procedure of proxy re-signature to maintain the linkage between integrity tags and their owners during the process of public auditing. While for computational overhead of proof verification, our proposed scheme still has a better performance since expensive interaction with the data owner is also eliminated.

Moreover, we implement both schemes based on a certain target file and compare the computational overhead in terms of running time. It is worth mentioning that if the file stored in the CSP is maliciously tampered or deleted, TPA is able to detect it with a probability greater than 95% or 99% by setting the number c to be 300 or 460 [5]. As is shown in Table 2, when the value of c is 300, the running time of our scheme is 1.262 s, which is about 55.7% lower than that of one-tag checker. Similarly, when the number c is set to be 460, our scheme still shows better performance.

6.3. Communication Overhead. In the evaluation of communication overhead, we compare ATDS with IPANM [3], in which the similar aggregation strategy is also employed even though it is not suitable for open deduplication system. The communication overhead consists of two parts: the overhead of tag generation and that of tag aggregation. In ATDS, tag generation is executed by the cloud user himself/herself, and thus, no communication is needed. However, for IPANM, the process entails communication of key

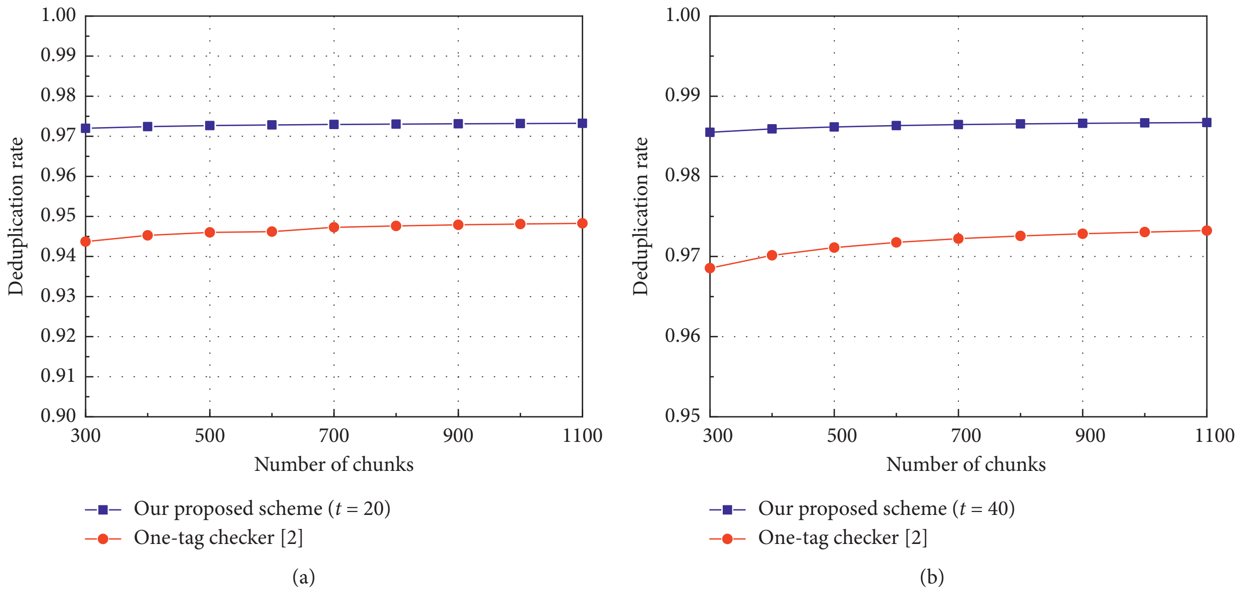


FIGURE 3: The relationship between deduplication rate and the number of chunks: (a) $t = 20$; (b) $t = 40$.

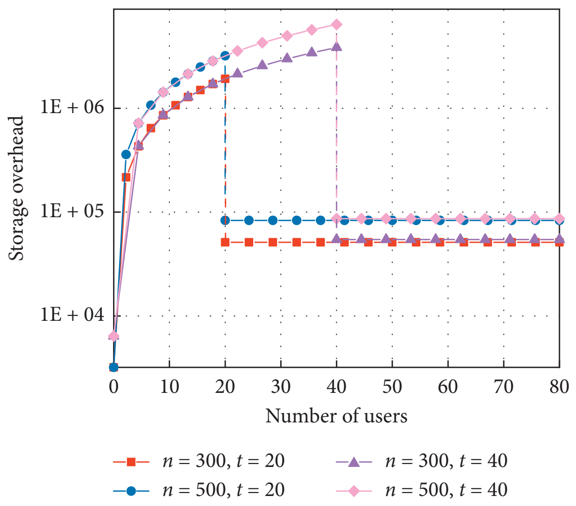


FIGURE 4: Storage overhead of our proposed scheme.

segments as well as verification parameters. Consider a system contains k users, IPANM first needs to select d qualified users based on validity verification. In this process, the communication overhead is $k(k-1)|\mathbb{Z}_p| + kt|\mathbb{G}_1|$. Moreover, in order to verify the correctness of generated integrity tags, each user in the group needs to receive verification parameters from others in the qualified user group to produce his/her public key sharing. The communication cost in this case is $td|\mathbb{G}_1|$. On the contrary, it has to send the public key sharing to each one of the users in the group with communication overhead of $td|\mathbb{G}_1|$.

As for the process of tag aggregation, in our proposed scheme, the first uploader needs to outsource auxiliary information, integrity tags, as well as secondary tags, the communication overhead of which is $t|\mathbb{G}_1| + 2nt|\mathbb{G}_1|$. If tag aggregation is not triggered, only integrity tags and secondary tags are needed for successive uploaders, and the cost is $2nt|\mathbb{G}_1|$ in this case. Otherwise, the user only needs to upload an aggregated verification tag, with communication overhead of $|\mathbb{G}_1|$. Thus, the total cost for k $(k > t)$ users is $t|\mathbb{G}_1| + 2nt|\mathbb{G}_1| + (k-t)|\mathbb{G}_1|$. In IPANM, the cost is $nt|\mathbb{G}_1|$ since t selected users need to send integrity tags to an aggregator, who carries out the process of aggregation. The results are shown in Table 3.

To verify the correctness of the above analysis, we also consider two target files and compare the total communication overhead of ATDS with IPANM [3]. Similar as the comparison in storage overhead, we also divide each one of the target files into 300 and 500 chunks of the same size and set the threshold value t to be 20 and 40, respectively. The communication overhead for both schemes is presented in Figures 5(a)–5(d). In these figures, the cost of our scheme

Table 1: Comparison of computational overhead.

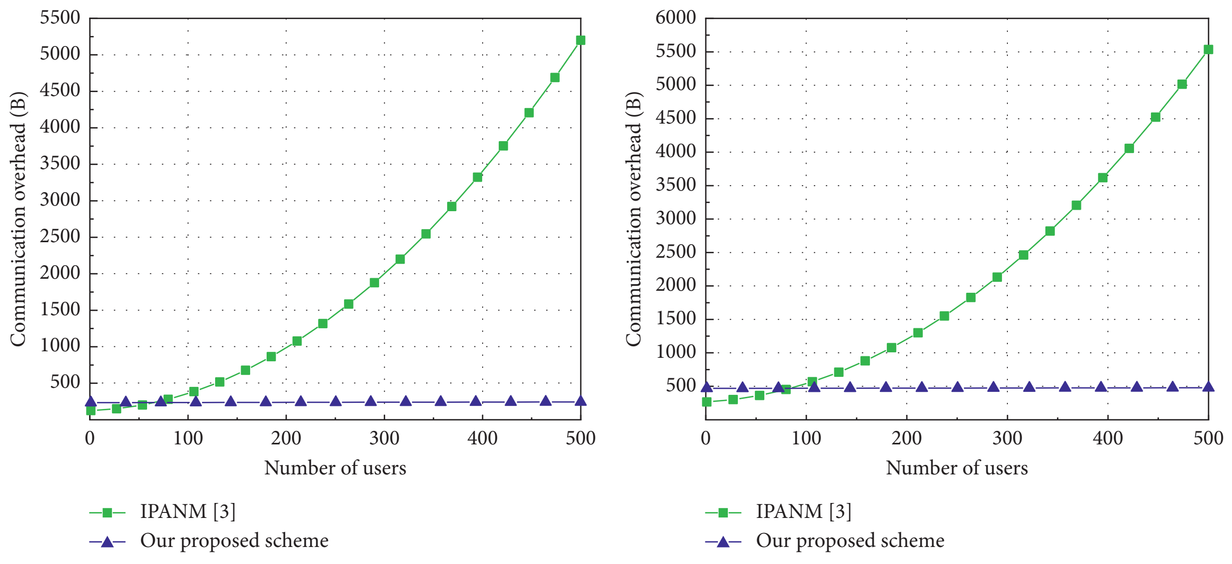
	Computational overhead	
	ATDS	One-tag checker [2]
Proof generation	$c\mathrm{Exp}_{\mathbb{G}_1} + (c-1)\mathrm{Mul}_{\mathbb{G}_1} + c\mathrm{Mul}_{\mathbb{Z}_p} + (c-1)\mathrm{Add}_{\mathbb{Z}_p}$	$2c\mathrm{Exp}_{\mathbb{G}_1} + 2(c-1)\mathrm{Mul}_{\mathbb{G}_1} + (2c+s)\mathrm{Mul}_{\mathbb{Z}_p} + s(c-1)\mathrm{Add}_{\mathbb{Z}_p}$
Proof verification	$2\mathrm{Pair} + c\mathrm{Hash}_{\mathbb{G}_1} + (c-1)\mathrm{Mul}_{\mathbb{G}_1} + (c+1)\mathrm{Exp}_{\mathbb{G}_1}$	$2\mathrm{Pair} + c\mathrm{Hash}_{\mathbb{G}_1} + (c+s)\mathrm{Mul}_{\mathbb{G}_1} + (c+s)\mathrm{Exp}_{\mathbb{G}_1}$

Table 2: Comparison of time required to perform public verification between ATDS and one-tag checker.

Number of challenged chunks	Computational overhead (s)	
	ATDS	One-tag checker [2]
$c = 300$	1.262	2.850
$c = 460$	1.955	4.032

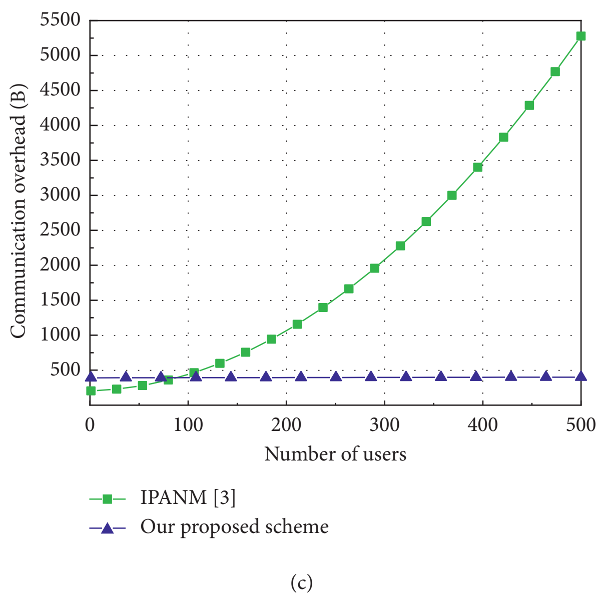
Table 3: Comparison of communication overhead.

	Communication overhead	
	ATDS	IPANM [3]
Tag generation phase	0	$k(k-1)\lvert\mathbb{Z}_p\rvert + kt\lvert\mathbb{G}_1\rvert + td\lvert\mathbb{G}_1\rvert + k\lvert\mathbb{G}_1\rvert$
Tag aggregation phase	$t\lvert\mathbb{G}_1\rvert + 2nt\lvert\mathbb{G}_1\rvert + (k-t)\lvert\mathbb{G}_1\rvert$	$nt\lvert\mathbb{G}_1\rvert$
Total communication overhead	$t\lvert\mathbb{G}_1\rvert + 2nt\lvert\mathbb{G}_1\rvert + (k-t)\lvert\mathbb{G}_1\rvert$	$k(k-1)\lvert\mathbb{Z}_p\rvert + (kt + td + k + nt)\lvert\mathbb{G}_1\rvert$



(a) (b)

Figure 5: Continued.



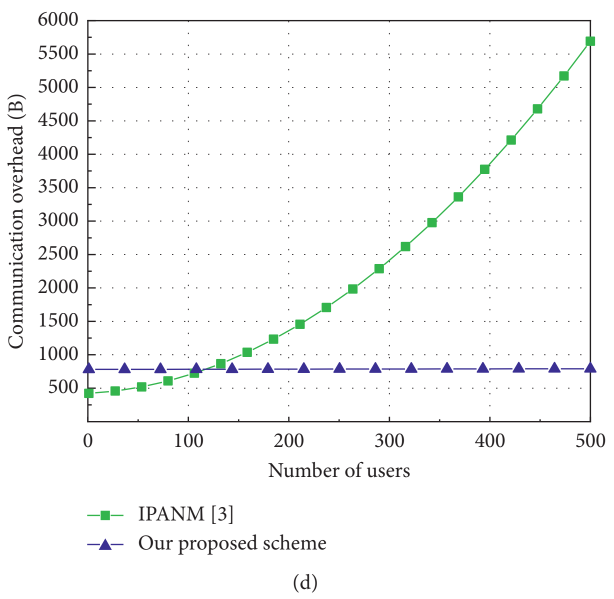


FIGURE 5: Comparison of communication overhead: (a) $n = 300$ and $t = 20$; (b) $n = 300$ and $t = 40$; (c) $n = 500$ and $t = 20$; (d) $n = 500$ and $t = 40$.

rises slightly with the number of users, while that of IPANM grows much faster. As shown in Figure 5(a), when the number of users increases from 0 to 66, the communication overhead of our proposed scheme is slightly larger than IPANM. The reason lies in that, before tag aggregation is triggered, users in our scheme need to upload both integrity tags and secondary tags. However, subsequent users only need to upload an aggregated verification tag, and the expensive interaction between the group of users in IPANM is completely eliminated in our scheme. Thus, the overall increment of communication overhead is just only 9.75 B for our proposed scheme, obviously lower than 5073.01 B for IPANM. The similar results are shown in Figures 5(b)–5(d).

7. Conclusion

This paper proposes a lightweight yet secure tag deduplication scheme called ATDS to address the potential side channel attack launched by a malicious TPA, which is able to steal the existence privacy of a certain target user during the process of public verification. With the help of the defined user-associated private key, the integrity tags for a certain ciphertext chunk could be aggregated via Lagrangian interpolation, which in turn achieves aggregation-based tag deduplication. With the help of this design, even though public verification is passed, the TPA is only able to judge that the verified data are correctly corresponding to at least a group of users in cloud storage, rather than determining specific owners. Thus, the potential side channel attack is well resisted. The results obtained from the experiment demonstrate that the cost of ATDS is controllable with security guaranteed.

Data Availability

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was specially supported by Fundamental Research Funds for the Central Universities, University of International Relations (3262021T13), Research Funds for NSD Construction, University of International Relations (2019GA36), and National Natural Science Foundation of China (U1536207).

References

[1] J. Douceur, A. Adya, W. Bolosky, D. Simon, and M. Theimer, “Reclaiming space from duplicate files in a serve-less distributed file system,” in *Proceedings of the IEEE DCS*, pp. 617–624, Shanghai, China, 2002.
[2] X. F. Liu, W. H. Sun, W. J. Lou, and Q. Q. Pei, “One-tag checker: message-locked integrity auditing on encrypted cloud deduplication storage,” in *Proceedings of the IEEE INFOCOM*, pp. 1–9, Atlanta, GA, USA, 2017.
[3] L. X. Huang, J. L. Zhou, G. X. Zhang et al., “IPANM: incentive public auditing scheme for non-manager groups in clouds,” *IEEE Transactions on Dependable and Secure Computing*, 2020.
[4] J. P. Berrut and L. N. Trefethen, “Barycentric Lagrange interpolation,” *SIAM Review*, vol. 46, no. 3, pp. 501–517, 2004.
[5] G. Ateniese, R. Burns, R. Curtmola et al., “Provable data possession at untrusted stores,” in *Proceedings of the ACM CCS*, pp. 598–609, Alexandria, VA, USA, 2007.
[6] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for secure cloud storage,” *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, 2013.

[7] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of Cryptology*, vol. 17, no. 4, pp. 297–319, 2004.

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

[9] H. Tian, Y. Chen, C. C. Chang et al., "Dynamic-hash-table based public auditing for secure cloud storage," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 701–714, 2017.

[10] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717–1726, 2013.

[11] Y. Zhu, H. X. Hu, G. J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multi-cloud storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 12, pp. 2231–2233, 2014.

[12] W. Shen, J. Qin, J. Yu, R. Hao, and J. Hu, "Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 331–346, 2019.

[13] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proceedings of the ACM CCS*, pp. 874–885, Denver, CO, USA, 2015.

[14] H. Kwon, C. Hahn, D. Y. Koo, and J. Hur, "Scalable and reliable key management for secure deduplication in cloud storage," in *Proceedings of the IEEE CLOUD*, pp. 391–398, Honolulu, HI, USA, 2017.

[15] Y. Zhang, Y. L. Mao, M. Z. Xu, F. Y. Xu, and S. Zhong, "Towards thwarting template side-channel attacks in secure cloud deduplication," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[16] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: deduplication in cloud storage," *IEEE Security & Privacy Magazine*, vol. 8, no. 6, pp. 40–47, 2010.

[17] J. Stanek and L. Kencl, "Enhanced secure thresholded data deduplication scheme for cloud storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 694–707, 2018.

[18] M. Bellare, S. Keelveedhi, and T. Ristenpart, "DepLESS: server-aided encryption for deduplicated storage," in *Proceedings of the USENIX Security Symposium*, pp. 179–194, Washington, DC, 2013.

[19] C. M. Yu, "Poster: Efficient cross-user chunk-level client-side data deduplication with symmetrically encrypted two-party interactions," in *Proceedings of the ACM CCS*, pp. 1763–1765, Vienna, Austria, 2016.

[20] X. Tang, L. N. Zhou, Y. F. Huang, and C. C. Chang, "Efficient cross-user deduplication of encrypted data through re-encryption," in *Proceedings of the IEEE Trustcom*, pp. 897–904, New York, NY, USA, 2018.