One-tag Checker: Message-locked Integrity Auditing on Encrypted Cloud Deduplication Storage

Xuefeng Liu*, Wenhai Sun†, Wenjing Lou†, Qingqi Pei‡, Yuqing Zhang*
*School of Cyber Engineering, Xidian University, Xi¡¯an, Shaanxi, China.
†Virginia Polytechnic Institute and State University, Blacksburg, VA, USA.
‡School of Telecommunications Engineering, Xidian University, Xi¡¯an, Shaanxi, China.

***Abstract*—In this paper, we investigate the problem of integrity auditing for cloud deduplication storage. Specifically, in addition to the outsourced data confidentiality, we also aim to ensure the integrity of the deduplicated cloud storage. With the existing works based on Provable Data Possession (PDP)/Proof of Retrievability (PoR), we are either required to rely on a fully trusted proxy server or inevitably sacrifice the privacy and efficiency. In contrast, we present a novel *message-locked* integrity auditing scheme *without an additional proxy server*, which is applicable to both file-level and chunk-level deduplication systems. In particular, our scheme is storage efficient in the sense that apart from eliminating the ciphertext redundancy, we also enable the *integrity tag deduplication* by a message-derived signing key, which merely incurs minimal client-side computation overhead. Besides, we can still *publicly* perform the integrity check over any client's cloud storage by incorporating the proxy re-signature technique. We show that the proposed scheme will not disclose the data ownership information and is provably secure under the Computational Diffie-Hellman (CDH) assumption in the random oracle model. Finally, the performance evaluation demonstrates its effectiveness and efficiency.**

I. Introduction

The last decade has witnessed the increasing proliferation of cloud storage. It is predicted that the cloud data volume will achieve 40 trillion gigabytes in 2020. A recent survey conducted by EMC indicates that about 75% of recent digital data stored in the cloud are duplicated copies [1]. For storage efficiency, commercial cloud storage services, such as Google Drive and Dropbox, adopt deduplication technique at the file/chunk level to store one copy of each data and refer other duplicates to this stored copy.

To prevent the third-party cloud from accessing user sensitive information, it is a common practice for users to encrypt their data before uploading. In the meanwhile, we still expect to enjoy the conventional data utilization services on top of the encrypted storage [2], [3], [4], [5], [6]. For example, other than data confidentiality, users may also want to ensure that their data are stored correctly in the cloud and remain intact. However, the fact that given randomized encryption and different user signing keys, the same data produces distinct ciphertexts and integrity tags, will outright incapacitate data deduplication and impose additional overhead on the system using current PDP/PoR techniques [4], [5], [7], [8], [9], [10], [11]. We may resort to message-locked encryption (MLE) [12], [13] or its prominent instantiation, convergent encryption (CE) [14] to enable dedupe functionality. Specifically, deterministic encryption is carried out with a message-derived key, i.e., the identical data always produces the identical encryption key and thus the same ciphertext. In this case, the integrity auditing for encrypted data using PDP/PoR will be "inefficient". This is because, for the identical data, each user is required to sign it with his associated private key, thereby generating different integrity tags (or authenticators, and we use them interchangeably hereafter). On the other hand, cloud needs to store all these authenticators, even if they are intended for the same data, so as to provide integrity proof for data owners in the later auditing process.

In the literature, the existing work for integrity auditing on deduplication storage, either relies on a fully trusted proxy server [15] or sacrifices the privacy and computation efficiency [16]. In [15], an additional independent proxy server is set up to collect users' files before uploading them to the cloud. If the file is unique in the cloud storage, the server computes the integrity tags for this file using its private key. Otherwise, it only adds related metadata to the storage. Therefore, the authenticators are dedupable due to the same server signing key. However, such strong assumption of the existence of an additional independent proxy server may not be practical for most application settings. Towards a serverless solution, clients in [16] need to compute a set of authenticators with their respective private keys, regardless of the underlying file duplication. Subsequently, cloud aggregates those sets of tags for the same file into one set. In addition to the high user-side computation cost, this design inevitably introduces a side channel during the auditing phase such that it discloses the file ownership information, since the integrity check needs public key information of all the owners of this file.

Our Contributions. Motivated by CE, we use a *message-derived private key* to sign the same data into an identical set of integrity tags. Therefore, we can reap the benefits of tag deduplication, which in turn results in the reduced user-side computation (only the user who first uploads the data to the cloud storage needs to generate the authenticators) and cloud storage savings (cloud only keeps one set of integrity tags for all the corresponding data owners). However, such design may break the linkage between the data and its owners during the auditing process, since instead of using the user-associated public key, performing verification requires the universal message-related public key. In order to address this challenging issue, we employ the proxy re-signature technique

[17] to ensure that the cloud can prove the integrity of the challenged message to its owner under user-associated public key. Our main contributions can be summarized as follows:

1) We propose a novel message-locked integrity auditing scheme on encrypted cloud deduplication storage without an additional proxy server. Besides CE-enabled data deduplication, we can also achieve the *integrity tag deduplication* by innovatively using the message-derived private key while still enabling *public* integrity auditing over encrypted storage via the homomorphic authenticator and proxy re-signature techniques. Our design is applicable to the practical deduplicated storage at either granularity, file level or chunk level.
2) The proposed scheme is efficient in terms of storage savings in the cloud and computation overhead on the user side, which is validated by our experimental results.
3) We show that our scheme will not leak the data ownership information during the auditing phase. We formally define and prove the security of the scheme under the CDH assumption [18] in the random oracle model.

II. Related Work

The concept of Provable Data Possession was first introduced by Ateniese *et al.* [5], which enables a verifier to check the integrity of users' data stored in an untrusted server. In their scheme, a data owner first splits his file F into n blocks $\{m_1, m_2, ..., m_n\}$. For each block m_i and its index i, the data owner computes an RSA-based homomorphic authenticator σ_i using his private key and outsources (m_i, σ_i) to the storage server. Given file F and its authenticators, the server can convince a verifier that a linear combination of blocks $\sum_{i=1}^{n} \nu_i m_i$ (with arbitrary weights $\{\nu_i\}$) is correctly generated via a constant-size proof computed from $\{\sigma_i\}$.

By combining erasure coding and homomorphic authenticators built from BLS signature [19], H. Shacham and B. Waters [4] proposed a Proof-of-Retrievability protocol for remote storage services, which not only assures the data owner of his outsourced files but also guarantees their full recovery. Besides, the scheme supports a tradeoff between storage cost and proof size by utilizing the fragment structure technique. In particular, a data owner breaks an erasure encoded file into blocks $\{m_1, m_2, ..., m_n\}$, where each block m_i consists of s elements (also named sectors) $\{m_{i,1}, m_{i,2}, ..., m_{i,s}\}$. The authenticator on (i, m_i) is constructed based on all the elements. Such design reduces the storage cost for authenticators to $\frac{1}{s}\times$ the case without fragment structure. On the other hand, to prove the integrity of a file, the server needs to compute an aggregated block including s elements and one aggregated authenticator.

In order to support dynamic operations on outsourced data files, Ateniese *et al.* [7] proposed a dynamic PDP protocol allowing a limited number of updates and verification requests. However, block insertions are not allowed. To overcome these limitations, Erway *et al.* [8] used authenticated dictionaries based on rank information to develop a dynamic PDP solution. Wang *et al.* [9] designed an improved PoR scheme for dynamic data based on Merkle Hash Tree and homomorphic authenticators. Zhu *et al.* [10] adopted index-hash table to achieve provable updates to outsourced data. Yang *et al.* [11] proposed a dynamic PDP scheme by using an index table to defend against replay attacks launched by the malicious storage server.

Note that it is impossible to achieve a fully deduplicated cloud storage with all the above-mentioned works, because the integrity tags stored in the cloud are generated under different users' private keys.

In the literature, there are auditing schemes focusing on shared cloud data for a group of users. Wang *et al.* [20] exploited ring signature to design an integrity auditing scheme for shared data in a group. Due to the characteristic of ring signature, the identity of a signer on each block is also protected. Yuan *et al.* [21] used polynomial-based authentication tags and proxy-tag update techniques to support dynamic shared data in a group. These works considered data sharing among a preset group of users [22], which are not allowed in deduplication systems. Other works considered different scenarios, *e.g.*, integrity auditing in distributed clouds [23].

Recently, Li *et al.* [15] proposed an integrity auditing scheme for encrypted deduplication storage. More specifically, a user encrypts his file by using a CE-based technique and uploads the ciphertext to a fully trusted proxy auditor server, who maintains a distributed MapReduce system. The proxy auditor server receives all the users' files before uploading them to the cloud. If an encrypted file has already been stored in the cloud, the auditor only adds the user metadata to the cloud; otherwise, the auditor invokes the MapReduce system to compute integrity tags with the auditor's private key and outsources them together with the file to the cloud. Thus, integrity tag deduplication is achieved. Albeit the merits of such additional proxy server design, such as capable of slowing down the dictionary attack against CE, this strong assumption is difficult to meet in the practical commercial context and the comparable functionality can also be realized in the serverless setting [24].

In [16], similar to the PDP/PoR based schemes for non-deduplication storage, each user has to compute the integrity tags, even for the file that has been stored in the cloud, with his private key. Next, the cloud aggregates the tags on the same block into one tag. As a result, the tag storage cost can be reduced. However, user-side auditing requires an aggregated public key that is computed via the multiplication of all the file owners' associated public keys. Hence, the adversary may launch brute-force attack to recover the file ownership information by guessing the possible component keys.

III. Preliminaries

A. Convergent Encryption

CE [14] aims at providing unpredictable data confidentiality in deduplication storage by encrypting a message F using the message-derived key K. As a special case of MLE [12], [13], the key K is the message's fingerprint/hash. Formally, a convergent encryption scheme can be defined as follows:

- $\mathbf{KeyGen_{CE}}(1^\kappa, F) \rightarrow K$: A deterministic key generation algorithm takes as input a security parameter κ as well as a message F, and outputs a convergent key K.
- $\mathbf{Encrypt_{CE}}(K, F) \rightarrow C$: A deterministic symmetric encryption algorithm takes as input the convergent key K as well as message F, and outputs a ciphertext C.
- $\mathbf{Decrypt_{CE}}(K, C) \rightarrow M$: A deterministic decryption algorithm takes as input the convergent key K as well as ciphertext C, and outputs the original message F.

We assume that the cloud storage is encrypted with CE. We only focus on the integrity auditing design, which is also compatible with existing MLE/CE techniques [12], [13], [25]. In practice, CE can be instantiated with any deterministic symmetric encryption, such as AES128.

B. Proof of Ownership

Proof of Ownership (PoW)[26] is an interactive protocol running between the storage server and a user. With PoW, a user is able to convince the server that he indeed owns a claimed file, in a more efficient way than uploading the whole file. By its security definition, the probability of a successful PoW by any malicious user is negligible, with the assumption that the adversary has the ability to access a portion of the target file.

C. Proxy re-signature

Proxy re-signature [27], was introduced by Blaze *et al.*, which enables a semi-trusted proxy to act as a translator of signatures between two users, for example, Alice and Bob. More specifically, given the proxy re-signature key, the proxy can convert a signature of Alice into a signature of Bob on the same message. Meanwhile, the proxy cannot learn any private keys of the two users. In other words, the proxy is not able to sign any message on behalf of either Alice or Bob. In this paper, we exploit the proxy re-signature technique [17] to let the cloud act as the proxy and convert the homomorphic authenticators under the message-related public/private key pairs to those with user-associated public/private key pairs during the integrity auditing process. As a result, the cloud is able to prove the integrity of the challenged message to the user under his public key.

D. Bilinear Map

Let G_1 and G_2 be two multiplicative cyclic groups of the same prime order q. Let e: $G_1 \times G_1 \rightarrow G_2$ denote a bilinear map constructed with the following properties. (1) Bilinearity: for all $a, b \in Z_q^*$ and $g_1, g_2 \in G_1$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$; (2) Non-degeneracy: there exists a point g_1 such that $e(g_1, g_1) \neq 1$; (3) Computability: $e(g_1, g_2)$ for any $g_1, g_2 \in G$ can be computed efficiently.

Definition 1: The CDH Assumption [18]. Given $g, g^s, g_0 \in G_1$ for unknown $s \in Z_q^*$, no probabilistic polynomial-time algorithm can compute g_0^s with non-negligible advantage.

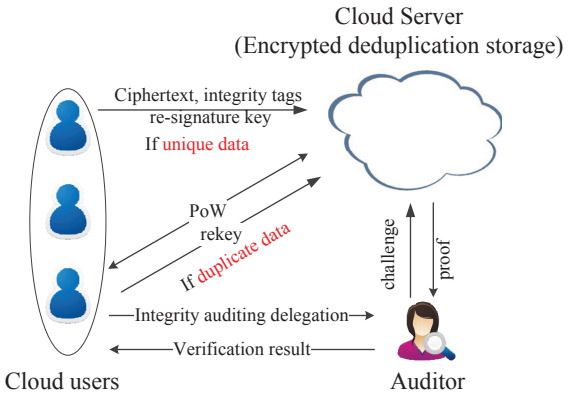


Fig. 1: Architecture

IV. Problem Formulation

A. System Model

As illustrated in Fig.1, there are three entities in our system. *Cloud user* uploads his CE-encrypted storage to the cloud while he still can be assured of the outsourced data integrity. If the uploaded data is *unique* (it has not been stored in the cloud), the user will send the integrity tags and re-signature key along with the ciphertext to the cloud. Otherwise, he will engage in a PoW protocol to convince the cloud that he is indeed the owner of the target data. After that, the user only needs to upload his re-signature key (rekey) to the cloud.

The *cloud server* provides data storage service to its users. In order to reduce the storage cost, the cloud only retains one copy of the duplicate data (including user files and the associated integrity tags). In addition, the cloud is obligated to generate a proof in the integrity auditing phase upon receiving the challenge request from a public auditor.

Since the data integrity is publicly verifiable by user public key, the role of an auditor can be played by either the user himself or a public trusted third party. Consistent with previous works [4], [9], [20], [15], [16], we assume that the auditing task is delegated to a third-party auditor in order to save the user's computation resources. On behalf of the user, the auditor invokes the integrity auditing protocol by sending the cloud a challenge. On receipt of the proof from the cloud, the auditor verifies the target data integrity by the user's public key and notifies the user of the result.

B. Design Goals

In this paper, we construct a practical and efficient auditing scheme for encrypted deduplication storage, which achieves the following design goals.

- **Functionality**. The integrity auditing task should be publicly verifiable. It realizes deduplication not only on the user data but also on the associated integrity authenticators, without an additional proxy server. Besides, the proposed scheme is applicable to both file and chunk based dedupe storage.

- **Efficiency**. Our scheme should incur low storage cost in the cloud and low computation overhead on the user side. Further, the proof generation and verification cost are expected to be on a par with existing works.
- **Security**. 1) *Ownership information protection*: Our protocol will not disclose the sensitive file ownership information to the user during the auditing phase[1]; 2) *Correctness*: The cloud is able to generate a valid proof if the challenged blocks and their integrity tags are correctly stored. 3) *Unforgeability*: we will formally prove that our proposed scheme can achieve *existential unforgeability under adaptive chosen-message attack* (see Sect.IV-D for the security definition).

C. Algorithm Formulation

In this subsection, we provide the formal algorithm definition of our construction.

Definition 2: Our scheme consists of a tuple of algorithms as follows.

- $\mathbf{KeyGen}(1^\kappa) \rightarrow (pk_u, sk_u)$: *A probabilistic algorithm run by a user takes the security parameter κ as input, and outputs a user-associated public/private key pair (pk_u, sk_u).*
- $\mathbf{TagKey}(1^\kappa, F) \rightarrow (pk_F, k_F)$: *A deterministic algorithm run by a user takes as input the security parameter κ as well as data F, and outputs a message-related public/private key pair (pk_F, k_F).*
- $\mathbf{Rekey}(pk_F, k_F, pk_u, sk_u) \rightarrow rk_{u,F}$: *A deterministic algorithm run by a user takes as input the message-related public/private key pair (pk_F, k_F), the user-associated public/private key pair (pk_u, sk_u), and outputs a re-signature key $rk_{u,F}$.*
- $\mathbf{TagBlock}(pk_F, k_F, C_i) \rightarrow T_i$: *A (possibly) probabilistic algorithm run by a user, takes as input the message-related public/private key pair (pk_F, k_F), a block C_i, and outputs a publicly verifiable tag T_i. We assume C is the ciphertext of F by using the convergent encryption, and C_i is the i-th block of C.*
- $\mathbf{GenProof}(Chal, \Sigma, \Gamma, rk_{u,F}) \rightarrow \mathcal{P}$: *It is a deterministic algorithm run by the cloud. It takes as input a challenge $Chal$, an ordered collection Σ of blocks, an ordered collection Γ of tags as well as a re-signature key $rk_{u,F}$, and outputs a proof $\mathcal{P}$.*
- $\mathbf{CheckProof}(pk_u, Chal, \mathcal{P})$: *A deterministic algorithm is run by the auditor to check the correctness of $\mathcal{P}$. It takes as input a user-associated public key pk_u, a challenge $Chal$, as well as a proof $\mathcal{P}$, and outputs 1 (accept) or 0 (reject).*

D. Security model

We consider that cloud may delete users' data due to server hacks or failures but will hide the data corruptions for its reputation [28]. The auditor is assumed to be honest and has no incentive to collude with the cloud. A user may infer the file ownership information during the auditing phase. Furthermore, it is possible that a portion of the user data are exposed to the adversary. In this case, we should still guarantee the integrity of the remaining uncompromised storage.

Definition 3: We state the formal security definition via the following experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathbf{1}^\kappa}$, which is a variation of the standard existential unforgeability under an adaptive chosen-message attack [29]. The experiment captures that an adversary cannot successfully construct a valid proof without possessing all the blocks corresponding to a given challenge, unless that it correctly guesses all the missing blocks.

Setup: We divide users into normal users and malicious users. For the l normal users and $l^{'}$ malicious users in the system, the challenger performs algorithm $\mathbf{KenGen}$ to generate user-associated public/private key pairs $(pk_{nu}, sk_{nu})_{nu\in[1,l]}$ and $(pk_{mu}, sk_{mu})_{mu\in[1,l']}$. In the end, normal users' public keys and malicious users' public/private keys are sent to adversary $\mathcal{A}$.

Step-1 Query: Adversary $\mathcal{A}$ can adaptively query oracle $\mathbf{TagKen}$ to obtain message-related public/private key pairs $(pk_{F_w^{'}}, k_{F_w^{'}})_{w\in[1,o']}$ for $o^{'}$ files. Then, $\mathcal{A}$ queries $\mathbf{Rekey}$ and gets re-signature keys $rk_{nu,F_w^{'}}$ and $rk_{mu,F_w^{'}}$ for $(1 \leq nu \leq l, 1 \leq mu \leq l^{'}, 1 \leq w \leq o^{'})$. Finally, $\mathcal{A}$ adaptively query $\mathbf{TagBlock}$ as follows.

Adversary $\mathcal{A}$ chooses a block $m_1^{'}$ and sends it to the challenger for the tag under message-related public key $pk_{F_w^{'}}$. The challenge calls algorithm $\mathbf{TagBlock}(pk_{F_w^{'}}, k_{F_w^{'}}, m_1^{'}) \rightarrow T_{1,w}^{'}$ and sends $T_{1,w}^{'}$ back to $\mathcal{A}$. Adversary $\mathcal{A}$ continually queries the tags on blocks $m_2^{'}, ..., m_{n'}^{'}$ under $pk_{F_w^{'}}$, and the challenger responds $T_{2,w}^{'}, ..., T_{n',w}^{'}$ accordingly. Finally, $\mathcal{A}$ stores the blocks and their tags.

Step-2 Query: $\mathcal{A}$ can adaptively query oracle $\mathbf{TagKen}$ to obtain message-related public keys $(pk_{F_w})_{w\in[1,o]}$ for o files. $\mathcal{A}$ then queries $\mathbf{Rekey}$ and gets re-signature keys rk_{nu,F_w} for $(1 \leq nu \leq l, 1 \leq w \leq o)$. In the end, $\mathcal{A}$ adaptively query $\mathbf{TagBlock}$ on blocks $m_1, m_2, ..., m_n$ as the case in step-1 query.

Challenge: The challenger requests $\mathcal{A}$ to provide a proof of possession for $\{m_i\}_{i\in I\subseteq[1,n]}$ determined by a challenge $Chal$ under the normal user public key pk_{nu}.

Forge: The adversary $\mathcal{A}$ outputs a possession proof $\mathcal{P}$.

If $\mathbf{CheckProof}(pk_{nu}, Chal, \mathcal{P})$ returns 1, then the adversary $\mathcal{A}$ wins this experiment.

Definition 4: We say that a data integrity auditing scheme is secure, if for any probabilistic polynomial time adversary $\mathcal{A}$ who does not possess all of the challenged data blocks (e..g, deleting/modifying one or more blocks), the probability that $\mathcal{A}$ succeeds in the above experiment is negligible, i.e.,

$$Pr[\mathbf{Exp}_{\mathbf{A}}{}^{1^\kappa}(\mathcal{A}) = 1] \leq negl(\kappa). \tag{1}$$

V. Our construction

In this section, we first describe our integrity auditing scheme on encrypted *file-level* deduplication storage, and then

[1]With deduplication, such information is inevitably leaked to the cloud. A public auditor will also gradually recover this information if different users delegate the auditing tasks on the same file.

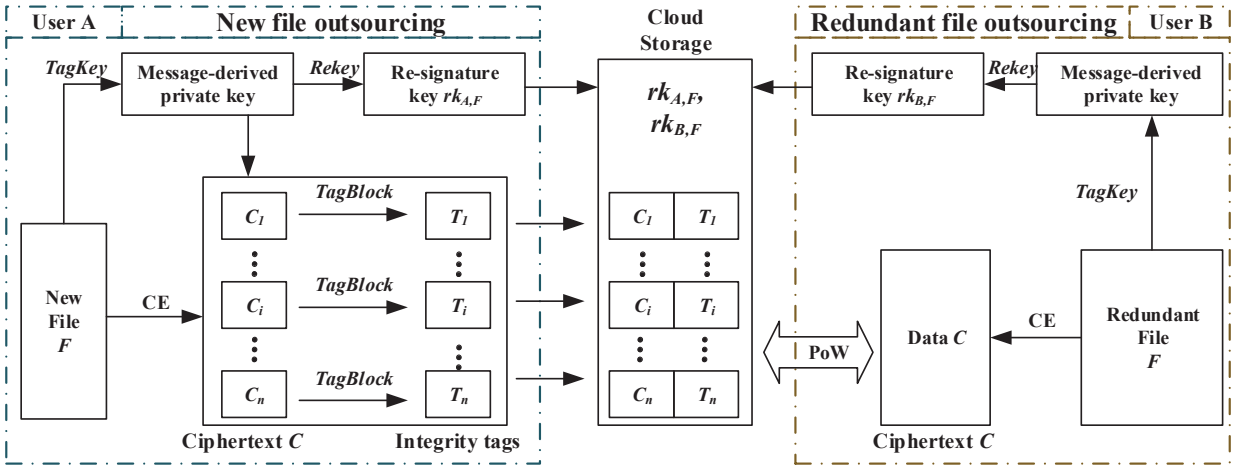


Fig. 2: Cloud storage deduplication for the encrypted user data and integrity tags.

apply it to the fine-grained *chunk level*.

A. Overview

Our scheme enables the deduplication for both file ciphertext and its authenticators in the cloud storage as shown in Fig. 2. Different from previous PDP/PoR schemes [5], [4], [7], [8], [9], [10], [11], [15], [16], we use a message-derived private key instead of the user-associated private key to compute integrity tags. Such design allows users and the cloud to reap the benefits of tag deduplication, i.e., the reduced user-side computation and the cloud storage savings. More precisely, only the user who first uploads the data to the cloud needs to generate the authenticators, and the cloud only keeps one set of integrity tags for all the corresponding data owners. However, the linkage between the file and its owners may be lost, due to the use of the universal message-derived signing key. On the other hand, the key management will be cumbersome if the user needs to maintain the potentially huge amount of message-derived key pairs especially in the case of chunk-based deduplication. Through the proxy re-signature technique [17], the cloud given a re-signature key can generate an integrity proof for the challenged blocks under user-associated public key.

Note that the cloud who knows a portion of the plaintext data in the storage may recover the user private key if we adopt the conventional re-signature technique [17]. For instance, this includes the situation that the cloud also happens to own some popular files in the storage, or that the cloud can even collude with malicious users. To address this issue, we protect each user's private key with the random masking technique during the re-signature key generation and the following auditing task can still be performed with the introduced randomness. In what follows, we provide the details of our construction for both file and chunk based deduplication storage.

B. Data Integrity Auditing for File-level Deduplication

Let G_1 and G_2 be two multiplicative cyclic groups of the same prime order q, and e: $G_1 \times G_1 \rightarrow G_2$ denote a bilinear map. $g, g_1, ..., g_s$ represent $s+1$ random elements in G_1. In addition, H: $\{0,1\}^* \rightarrow G_1$ and h: $\{0,1\}^* \rightarrow Z_q^*$ are two secure hash functions. The system public parameters are $\{e, G_1, G_2, q, g, g_1, ..., g_s, H, h\}$.

Input: File F, user-associated private key sk_u obtained from **KeyGen**.
Output: A re-signature key $rk_{u,F}$ or a tuple {ciphertext C, integrity tags $T = (T_1, ..., T_n)$, a message-related public key pk_F, a re-signature key $rk_{u,F}$}.

1: Encrypt file F using CE to ciphertext C.
2: Perform client-side deduplication protocol with the cloud via $ID_C = \mathsf{SHA1}(C)$.
3: **if** ID_C is already in the storage **then**
4: Run PoW with the cloud and invoke **TagKey** to get the message-related private key k_F.
5: Compute the re-signature $rk_{u,F}$ from **Rekey**.
6: **return** $rk_{u,F}$
7: **else**
8: Call **TagKey** to obtain the message-related public/private key pair (pk_F, k_F)
9: Generate tags $T = \{T_1, ..., T_n\}$ via **TagBlock**.
10: Compute the re-signature key $rk_{u,F}$ through **Rekey**.
11: **return** $C, T, pk_F, rk_{u,F}$
12: **end if**

Fig. 3: File uploading process

Consider that user $\mathcal{U}$ wants to outsource file F to the cloud, he proceeds as shown in Fig.3. The involved algorithms are described as follows. Note that, to be consistent with the practical deduplication storage (e.g., Dropbox), we adopt $\mathsf{SHA1}$ to compute the file identifier.

KeyGen: User $\mathcal{U}$ chooses a random number $s_u \in Z_q^*$ as his private key and computes the associated public key $pk_u = g^{s_u} \in G_1$. Besides, $\mathcal{U}$ selects another random number $x_u \in Z_q^*$ as his secret.

TagKey: This algorithm is performed by users to generate

message-related public/private keys. Specifically, user $\mathcal{U}$ computes $k_F = h(F) \in Z_q^*$. If the data is unique in the cloud, $\mathcal{U}$ also computes $pk_F = g^{k_F} \in G_1$, which enables the cloud to check the validity of the outsourced tags.

Rekey: It is performed by users to compute re-signature keys, which enables the cloud to prove the integrity of the challenged file under user-associated private/public key pair. $\mathcal{U}$ computes $d_{u,F} = s_u \cdot (k_F)^{-1} + r_{u,F}, h_{u,F} = r_{u,F} \cdot x_u$ and also sets $rk_{u,F} = (d_{u,F}, h_{u,F})$, where $r_{u,F}$ is a random number in Z_q^*.

TagBlock: This algorithm is run by users to compute data integrity tags. We use fragment data structure in tag generation, as shown in Fig.4. In particular, user $\mathcal{U}$ splits C into blocks $\{C_1, C_2, ..., C_n\}$ and further divides each block C_i into sectors $\{C_{i,1}, C_{i,2}, ..., C_{i,s}\}$. For each block C_i, $\mathcal{U}$ computes $T_i = [H(ID_C||i) \cdot \prod_{j=1}^{s} g_j^{C_{i,j}}]^{k_F} \in G_1$.

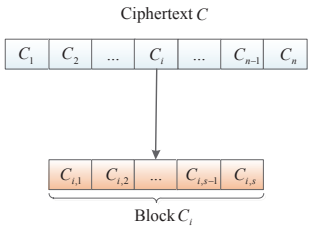


Fig. 4: Fragment structure

GenProof: The auditor chooses a random c-element subset $I \subset [1, n]$ along with c random coefficients in Z_q^*. Let $Q = \{i, v_i\}_{i \in I}$ be the set of challenge index-coefficient pairs. After receiving Q from the auditor, the cloud sends a proof $\mathcal{P} = (\mathcal{T}, \mathcal{T}_1, \rho_1, ...\rho_s)$ back to the auditor, where $\mathcal{T} = \prod_{(i,v_i) \in Q} T_i^{d_{u,F} \cdot v_i} \in G_1$, $\mathcal{T}_1 = \prod_{(i,v_i) \in Q} T_i^{h_{u,F} \cdot v_i} \in G_1$ and $\rho_j = \sum_{(i,v_i) \in Q} v_i \cdot C_{i,j} \in Z_q^*$ for $1 \le j \le s$.

CheckProof: The auditor sends $\mathcal{T}_1$ to $\mathcal{U}$ and obtains $\mathcal{T}_1' = \mathcal{T}_1^{x_u^{-1}}$. The auditor then accepts the proof if the following equation holds.

$$e(\frac{\mathcal{T}}{\mathcal{T}_1'}, g) = e(\prod_{(i,v_i) \in Q} H(ID_C||i)^{v_i} \cdot \prod_{j=1}^{s} g_j^{\rho_j}, pk_u) \quad (2)$$

C. Data Integrity Auditing for Chunk-level Deduplication

Here we apply our scheme to the fine-grained chunk based deduplication storage. In this case, file F is split into fixed-sized or variable-sized chunks $\{F^{(1)}, F^{(2)}, ..., F^{(n)}\}$, and the cloud executes the redundancy elimination at the granularity of chunk level. For convenience, we assume that the chunk size is constant, and that $C^{(k)}$ is the CE ciphertext of $F^{(k)}$. Each chunk $C^{(k)} (k \in [1, n])$ can be regarded as a subfile, so that the auditing protocol can be executed similarly to the file-level dedupe case. Suppose that the message-derived public/private key pair is $(pk_{F^{(k)}} = g^{k_{F^{(k)}}}, k_{F^{(k)}} = h(F^{(k)}))$, and the re-signature key is $rk_{u,F}^{(k)} = (d_{u,F}^{(k)}, h_{u,F}^{(k)}) = (sk_u \cdot (k_{F^{(k)}})^{-1} + r_{u,F}^{(k)}, r_{u,F}^{(k)} \cdot x_u)$, where $r_{u,F}^{(k)}$ is a random number chosen from Z_q^*.

1) If there is no copy of $C^{(k)}$ in the cloud, $\mathcal{U}$ computes integrity tags $T^{(k)} = \{T_1^{(k)}, T_2^{(k)}, ..., T_t^{(k)}\}$ via **TagBlock** by using the message-derived private key $k_{F^{(k)}}$. Finally, $\mathcal{U}$ outsources $\mathcal{C}^{(k)}$, integrity tags $T^{(k)}$, the re-signature key $rk_{u,F}^{(k)} = (d_{u,F}^{(k)}, h_{u,F}^{(k)})$ and the message-related public key $pk_F^{(k)}$ to the cloud.
2) Otherwise, $\mathcal{U}$ performs the PoW protocol with the cloud and sends the re-signature key $rk_{u,F}^{(k)}$ to the cloud.

TagBlock: For each block $C_i^{(k)} (1 \le i \le t)$, $\mathcal{U}$ sets the chunk identifier to be $ID_{C^{(k)}} = \mathsf{SHA1}(C^{(k)})$ and computes $T_i^{(k)} = [H(ID_{C^{(k)}}||i) \prod_{j=1}^{s} g_j^{C_{i,j}^{(k)}}]^{k_{F^{(k)}}}$.

$$\mathcal{C}^{(k)} = \begin{bmatrix} \mathcal{C}_1^{(k)} \\ \mathcal{C}_2^{(k)} \\ ... \\ \mathcal{C}_t^{(k)} \end{bmatrix} = \begin{bmatrix} \mathcal{C}_{1,1}^{(k)} & \mathcal{C}_{1,2}^{(k)} & ... & \mathcal{C}_{1,s}^{(k)} \\ \mathcal{C}_{2,1}^{(k)} & \mathcal{C}_{2,2}^{(k)} & ... & \mathcal{C}_{2,s}^{(k)} \\ ... & ... & ... & \\ \mathcal{C}_{t,1}^{(k)} & \mathcal{C}_{t,2}^{(k)} & ... & \mathcal{C}_{t,s}^{(k)} \end{bmatrix} \quad (3)$$

GenProof: The auditor chooses a random challenge set I of c indices along with c random coefficients in Z_q^*. We use $(k, i) \in I$ to indicate the index of the challenged block $\mathcal{C}_i^{(k)}$ and $v_i^{(k)}$ is the corresponding challenge coefficient. Let $Q = \{(k, i), v_i^{(k)}\}_{(k,i) \in I}$ be the set of challenge index-coefficient pair. Upon receiving Q, the cloud proceeds as follows and returns $\mathcal{P}$ to the auditor.

1) Compute $\mathcal{T} = \prod_{((k,i),v_i^{(k)}) \in Q} (T_i^{(k)})^{d_{u,F}^{(k)} \cdot v_i^{(k)}} \in G_1$.
2) Compute $\mathcal{T}_1 = \prod_{((k,i),v_i^{(k)}) \in Q} (T_i^{(k)})^{h_{u,F}^{(k)} \cdot v_i^{(k)}} \in G_1$ and $\rho_j = \sum_{((k,i),v_i^{(k)}) \in Q} v_i^{(k)} \cdot C_{i,j}^{(k)}$ for $1 \le j \le s$.
3) Set proof $\mathcal{P} = \{\mathcal{T}, \mathcal{T}_1, \rho_1, .., , \rho_s\}$.

CheckProof: The auditor sends $\mathcal{T}_1$ to $\mathcal{U}$ and obtains $\mathcal{T}_1' = \mathcal{T}_1^{x_u^{-1}}$. If Equation (4) holds, the auditor accepts the proof; otherwise, the proof will be rejected.

$$e(\frac{\mathcal{T}}{\mathcal{T}_1'}, g) = e(\prod_{((k,i),v_i^{(k)}) \in Q} H(ID_{\mathcal{C}^{(k)}}||i)^{v_i^{(k)}} \prod_{j=1}^{s} g_j^{\rho_j}, pk_u) \quad (4)$$

VI. Security Analysis

Our scheme achieves the confidentiality for unpredictable data by using the convergent encryption. In addition, data ownership information is not disclosed during the tag verification phase since only the user's public key pk_u is involved. Thus, we focus on the *correctness* and *unforgeability* of our integrity auditing scheme.

Theorem 1: The cloud is able to generate a proof that passes the verification if all the challenged blocks and their integrity tags are correctly stored.

Proof: Proving the *correctness* of our integrity auditing scheme for file and chunk based deduplication storage is equivalent to proving that Equation (2) and (4) hold. Due to page limitation, we only show the correctness of Equation (2) and that of Equation (4) can be proved similarly. Based on the

properties of bilinear maps, the correctness can be deduced from the followings.

$$\begin{aligned} & e(\frac{\mathcal{T}}{\mathcal{T}^{'}}, g) \\ = \ & e(\prod_{(i,v_i)\in Q} T_i^{d_{u,F}v_i} \cdot (\prod_{(i,v_i)\in Q} T_i^{r_{u,F}v_i})^{-1}, g) \\ = \ & e(\prod_{(i,v_i)\in Q} T_i^{sk_u\cdot(k_F)^{-1}v_i}, g) \\ = \ & e(\prod_{(i,v_i)\in Q} [H(ID_C||i)\cdot \prod_{j=1}^{s} g_j^{C_{i,j}}]^{s_u\cdot v_i}, g) \\ = \ & e(\prod_{(i,v_i)\in Q} [H(ID_C||i)^{v_i}\cdot \prod_{j=1}^{s} g_j^{v_i C_{i,j}}], pk_u) \\ = \ & e(\prod_{(i,v_i)\in Q} [H(ID_C||i)^{v_i}\cdot \prod_{j=1}^{s} g_j^{\rho_j}], pk_u) \end{aligned} \tag{5}$$

■

Theorem 2: Under the CDH assumption, our proposed scheme is secure against an adaptive chosen-message attack in the random oracle model.

Proof: We assume that there exists an adversary $\mathcal{A}$ succeeding in the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathbf{1}^\kappa}$ with non-negligible probability. Then we show how to construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to solve the CDH problem. That is, given a CDH tuple (g, g^a, g_0), the adversary $\mathcal{B}$ is able to compute g_0^a with non-negligible probability. $\mathcal{B}$ simulates the proposed scheme for $\mathcal{A}$ as follows.

Setup: The normal user-associated public keys are set to be $pk_{nu} = g^{as_{nu}}$ for $nu \in [1, l]$, where s_{nu} are randomly chosen from Z_q^*. In addition, $\mathcal{B}$ sets $g_j = g_0^{y_j}$ for $1 \le j \le s$. Besides, $\mathcal{B}$ chooses x_{nu} randomly from Z_q^*. For malicious users, $\mathcal{B}$ selects random numbers $x_{mu}, s_{mu} (mu \in [1, l^{'}])$ and computes the public keys $pk_{mu} = g^{s_{mu}}$. Finally, the system parameters, the normal user public keys $pk_{nu} (nu \in [1, l])$, the malicious public and private key pairs $(pk_{mu}, s_{mu}, x_{mu}) (mu \in [1, l^{'}])$ are given to the adversary $\mathcal{A}$.

Step-1 Query: There are four types of queries that $\mathcal{A}$ can request: oracle $\mathbf{TagKey}$, oracle $\mathbf{Rekey}$, oracle $\mathbf{TagBlock}$ and the hash function H.

1) Oracle $\mathbf{TagKey}(1^\kappa, F_w^{'})$: if $F_w^{'}$ has not been queried before, $\mathcal{B}$ returns a random number $x_w^{'} \in Z_q^*$ to $\mathcal{A}$ and records it in list TagKey. Otherwise, $\mathcal{B}$ obtains $x_w^{'}$ from list TagKey and responds it to $\mathcal{A}$.
2) Oracle $\mathbf{ReKey}(pk_{F_w^{'}}, pk_{nu}/pk_{mu})$: for malicious user public key pk_{mu}, $\mathcal{B}$ returns $x_w^{'-1} \cdot s_{mu} + r_{mu,w^{'}} \cdot x_{mu}$ to $\mathcal{A}$, where $r_{mu,w^{'}}$ is a random in Z_q^*; for normal user, $\mathcal{B}$ returns two random numbers to $\mathcal{A}$.
3) Oracle $\mathbf{TagBlock}(pk_{F_w^{'}}, m_i^{'}, ID_{m_i^{'}}||i)$: if $ID_{m_i^{'}}||i (i \in [1, n^{'}])$ has not been queried before, $\mathcal{B}$ chooses a random element from G_1 as the value of $H(ID_{m_i})||i)$ and then computes $T_i^{'} = ((H(ID_{m_i})||i)\prod_{j=1}^{s} g_j^{m_i})^{x_w^{'}}$ for the query. In the end, $\mathcal{B}$ records $T_i^{'}$ in list and returns $H(ID_{m_i}||i)$ for the corresponding hash query. Otherwise, $\mathcal{B}$ returns $T_i^{'}$ from list $T_{F_w^{'}}$ to $\mathcal{A}$.

Step-2 Query: There are three types of queries that $\mathcal{A}$ can request: oracle $\mathbf{Rekey}$, oracle $\mathbf{TagBlock}$ and the hash function H. For the message-related public key, $\mathcal{B}$ sets $pk_{F_w} = g^{ax_w}$, where x_w is a random number in Z_q^*.

1) Oracle $\mathbf{ReKey}(pk_{F_w}, pk_{nu})$: $\mathcal{B}$ returns $(x_w^{-1} \cdot s_{nu} + r_{nu,w}, x_{nu} r_{nu,w})$ to $\mathcal{A}$, where $r_{nu,w}$ is a random number in Z_q^*.
2) Oracle $\mathbf{TagBlock}(pk_{F_w}, m_i, ID_{m_i}||i)$: if $ID_{m_i}||i (i \in [1, n])$ has not been queried before, $\mathcal{B}$ computes $T_i = g^{ax_w r_i}$ and records it in list T_{F_w}. In the end, $\mathcal{B}$ returns T_i and $H(ID_{m_i}||i) = \frac{g^{r_i}}{\prod_{j=1}^{s} g_j^{m_{i,j}}}$ for the corresponding hash query. It is easily observed that T_i is a valid tag under the public key pk_{F_w}. If $\{m_i, ID_{m_i}||i\}$ is in list T_{F_w}, $\mathcal{B}$ obtains T_i and returns it to $\mathcal{A}$.

Challenge: For simplicity, $\mathcal{B}$ requests the adversary $\mathcal{A}$ to prove the integrity of all blocks $m_1, ..., m_n$ by sending coefficients $a_1, ..., a_n$ under the public key pk_u.

Forge: We assume that $\mathcal{A}$ has deleted or modified one or more blocks. Let $\rho_j^{'} = \sum_{i=1}^{n} a_i m_{i,j}$ be the real result. $\mathcal{A}$ returns a proof $\mathcal{P} = (\mathcal{T}, \mathcal{T}_1, \rho_1, ..., \rho_s)$ satisfying $e(\frac{\mathcal{T}}{\mathcal{T}_1^{x_{nu}^{-1}}}, g) = e(\prod_{i=1}^{c} H(ID_{\mathcal{C}}||i)^{a_i} \cdot \prod_{j=1}^{s} g_j^{\rho_j}, pk_u)$, but there exists at least one value $\rho_j \neq \rho_j^{'}$. Since $\mathcal{P}$ is a valid proof under public key pk_u, we have

$$\begin{aligned} \frac{\mathcal{T}}{\mathcal{T}_1^{x_{nu}^{-1}}} &= [\prod_{i=1}^{n} H(ID_{m_i}||i)^{a_i} \cdot \prod_{j=1}^{s} g_j^{\rho_j}]^{as_u} \\ &= (\prod_{i=1}^{n} (\frac{g^{r_i}}{\prod_{j=1}^{s} g_j^{m_{i,j}}})^{a_i} \cdot \prod_{j=1}^{s} g_j^{\rho_j})^{as_u} \\ &= (\prod_{i=1}^{n} g^{r_i a_i} \prod_{j=1}^{s} g_j^{\rho_j - \rho_j^{'}})^{as_u} \\ &= g^{as_u \sum_{i=1}^{n} r_i a_i} (g_0^{s_u \sum_{j=1}^{s} y_j(\rho_j - \rho_j^{'})})^a \end{aligned} \tag{6}$$

From the equation (7), $\mathcal{B}$ can easily compute $g_0^a = (\frac{\mathcal{T}}{\mathcal{T}_1^{x_{nu}^{-1}} g^{as_u \sum_{i=1}^{n} r_i a_i}})^{[s_u \sum_{j=1}^{s} y_j(\rho_j - \rho_j^{'})]^{-1}}$.

Now, we analyze the probability that $\mathcal{A}$ successfully forges the values satisfying $\rho_j = \rho_j^{'}$ for $(1 \le j \le s)$ if $\mathcal{A}$ does not possess all the sectors $m_{i,j} (1 \le i \le n, 1 \le j \le s)$. Consider the following multivariate polynomial in finite field Z_q^*:

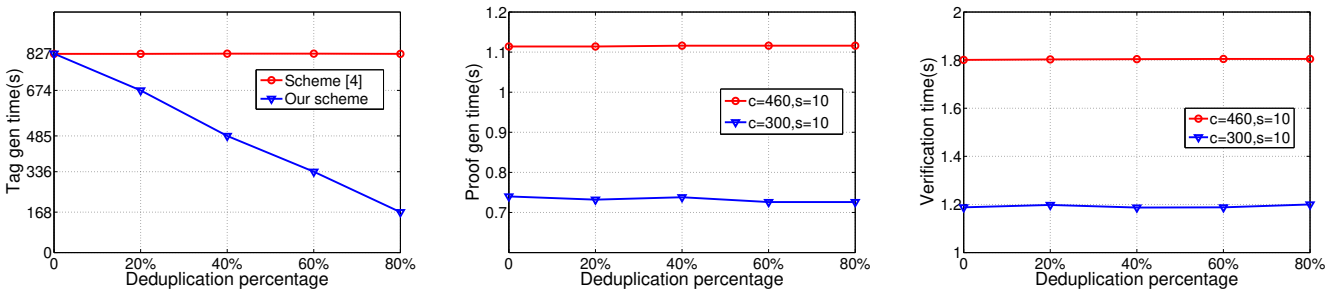
$$P(m_{1,j}, ..., m_{n,j}) = a_1 \cdot m_{1,j} + ... + a_n \cdot m_{n,j} - \rho_j^{'}. \tag{7}$$

Note that adversary $\mathcal{A}$ forging ρ_j correctly is equivalent to find ρ such that $P = 0$. However, due to Lemma 1 in [30], for any (non-zero) multivariate polynomial P in Z_q^* of degree d (in our case $d = 1$) and randomly chosen $m_{1,j}, ..., m_{n,j}$ with unknown $\rho_j^{'}$, the probability that $P = 0$ is $\frac{d}{q} = \frac{1}{q}$. Thus, we conclude that the probability that adversary $\mathcal{A}$ forges a valid value $\rho_j = \rho_j^{'}$ is negligible.

The interactions between $\mathcal{A}$ and $\mathcal{B}$ are indistinguishable from that between $\mathcal{A}$ and an honest challenger in the experiment, as $\mathcal{B}$ chooses all parameters according to our scheme for file-level deduplication storage. The proof of the security for the chunk-level case is similar, and we omit it here due to the limited space. Therefore, our scheme is secure against an adaptive chosen-message attack in the random oracle model under the CDH assumption. ■

VII. Performance evaluation

In this section, we evaluate the performance of our protocol in terms of storage cost, communication overhead, and computation efficiency. Note that we focus on the proposed integrity auditing scheme and do not intend to measure the standard CE and PoW cost here. In addition, since our scheme



(a) Tag generation cost (b) Proof generation cost (c) Proof verification cost

Fig. 5: Computation efficiency. Experiment on a 10 MB file with chunk size of 32 KB and 10 sectors in a block.

shares the similar design with [4] that was proposed in the non-deduplicaton setting, we draw the comparison here to demonstrate the efficiency of our scheme[2].

We analyze the protocol performance by both numerical evaluation and simulation. For the file-level deduplication storage, we assume that m users store the same file in the cloud, and the ciphertext of the file includes n_F blocks. In the case of chunk-level dedupe, we consider that a user file contains n_C chunks of which k chunks are redundant. We define the deduplication percentage dp as the ratio of the number of chunks that have duplicate copies in the cloud storage to the total number of chunks in the file, i.e., $dp = \frac{k}{n_C}$. Each chunk is further split into t blocks. Moreover, we conduct the experiment on a Linux server with Intel Core i5 3.3 GHz Processor and 4 GB Memory. For simplicity, we choose type-A (symmetric) pairing with 80-bit security level using PBC library[3]. Note that our scheme is also compatible with asymmetric pairings.

A. Storage cost

We measure the cloud-side storage cost for integrity tags and re-signature keys. As per our assumption, there are n_F integrity tags and m re-signature keys, which accounts for $n_F \cdot |G_1| + 2m \cdot |Z_q^*|$ storage cost. $|G_1|(|Z_q^*|)$ denotes the length of an element in $G_1(Z_q^*)$. In contrast, the corresponding storage overhead with previous works [4] is $m \cdot n_F \cdot |G_1|$.

For the chunk-level case, cloud needs to store $t \cdot (n_C - k)$ tags and n_C re-signature keys, which impose a cost of $t \cdot (n_C - k) \cdot |G_1| + 2n_C \cdot |Z_q^*|$. However, the cloud in [4] has to store $t \cdot n_C$ tags with the size of $t \cdot n_C \cdot |G_1|$. For ease of illustration, we set the system security level as 80 bits, which indicates $|G_1| = 64$ bytes and $|Z_q^*| = 20$ bytes. Thus, the storage cost ratio of our scheme to [4] is $p = \frac{t \cdot (n_C - k) \cdot |G_1| + 2n_C \cdot |Z_q^*|}{t \cdot n_C \cdot |G_1|} = 1 - \frac{k}{n_C} + \frac{40}{64t}$. Consider an example with 32 KB chunk size and s=10. We have $t = \lceil \frac{32*1024}{20*10} \rceil = 164$ and $p = 1.003 - \frac{k}{n_C}$, which indicates that if more than 3‰ chunks in a file are deduplicated, our scheme will outperform [4] in terms of cloud storage savings. For reference, the typical deduplication percentage ranges from 75% to 99.8% [31]. The more duplicate chunks, the more storage efficiency we achieve with our scheme.

[2]We adapt [4] to the encrypted deduplication scenario to facilitate the comparison.

[3]https://crypto.stanford.edu/pbc/.

B. Communication overhead

We evaluate the communication overhead incurred by one challenge-proof interaction between the cloud and auditor. Specifically, the challenge is composed of c index-coefficient pairs $\{i, v_i\}$, where i denotes the index of a block. Hence, one interaction needs $c \cdot \log n_F + c \cdot |Z_q^*|$ bandwidth for the challenge, and $2|G_1| + s \cdot |Z_q^*|$ for the proof response $\mathcal{P} = (\mathcal{T}, \mathcal{T}_1, \rho_1, ...\rho_s)$ in the file-based deduplication. At the chunk level, challenge consumes $c \cdot (\log n_C + \log t) + c \cdot |Z_q^*|$ bandwidth. The cloud needs to send a proof consisting of two elements in G_1 and s elements in Z_q^*. The communication of our scheme requires one more element in G_1 compared to [4].

C. Computation Efficiency

We evaluate the related computation efficiency of our protocol design. In particular, we measure the time cost for the integrity tag generation, proof computation and verification. Let $\mathbf{Exp}_{G_1}$ denote one exponentiation in G_1, **Pair** be one pairing operation on $e : G_1 \times G_1 \rightarrow G_2$, and $\mathbf{Hash}_{G_1}$ represent one hash operation from $\{0,1\}^* \rightarrow G_1$. In addition, $\mathbf{Mul}_{G_1}$ and $\mathbf{Add/Mul}_{Z_q^*}$ denote one multiplication in G_1 and one modular addition/multiplication in Z_q^*, respectively.

1) Tags generation. As shown in Sect.V, m users in file-based deduplication should compute n_F tags and m re-signature keys, and the computation cost is $n_F \cdot [\mathbf{Hash}_{G_1} + (s+1)\mathbf{Exp}_{G_1} + s\mathbf{Mul}_{G_1}] + 3m \cdot \mathbf{Add/Mul}_{Z_q^*}$. With [4], it takes $n_F \cdot l \cdot [\mathbf{Hash}_{G_1} + (s+1)\mathbf{Exp}_{G_1} + s \cdot \mathbf{Mul}_{G_1}]$ to compute integrity tags for the same file.

In the chunk-level deduplication scheme, a user computes $t \cdot (n_C - k)$ tags and n_C re-signature keys, i.e., $t \cdot (n_C - k)[\mathbf{Hash}_{G_1} + (s+1) \cdot \mathbf{Exp}_{G_1}) + s \cdot \mathbf{Mul}_{G_1}] + 3n_C \cdot \mathbf{Add/Mul}_{Z_q^*}$. Fig.5(a) shows that the time cost for tag generation with our scheme is inversely proportional to dp, whereas [4] is constant in any circumstances and user needs to compute authenticators for all the chunks in the file.

2) Proof generation. The proof generation cost is determined by the number c of challenge index-coefficient pairs and the

number s of sectors in a block . For both file-level and chunk-level deduplication storages, the cloud spends $2c \cdot \mathbf{Exp}_{G_1} + 2(c-1) \cdot \mathbf{Mul}_{G_1} + [c + s(2c-1)] \cdot \mathbf{Add/Mul}_{Z_q^*}$ on computing a proof $\mathcal{P} = (\mathcal{T}, \mathcal{T}_1, \rho_{j \in [1,s]})$. In [4], the tag generation cost is $c \cdot \mathbf{Exp}_{G_1} + (c-1) \cdot \mathbf{Mul}_{G_1} + s(2c-1) \cdot \mathbf{Add/Mul}_{Z_q^*}$.

The authors in [5] proved that a verifier can detect cloud misbehavior with a high probability by challenging a small number of random blocks. For example, if 1% of the file is deleted or modified, a verifier can detect it with a probability greater than 95% or 99% by setting the number c of challenged blocks to be 300 or 460. From Fig.5(b), we deduce that the deduplication does not bring any extra computation overhead to the proof generation.

3) Proof verification. The proof verification cost is comparable with [4]. The auditor computes two pairings, $c + s$ exponentiations and $c + s - 1$ multiplications in G_1 to verify the proof. As shown in Fig.5(c), given the fixed system parameters, the verification cost only depends on the challenge request, irrelevant to the duplication level of the storage.

In contrast to the existing works, our scheme exhibits better storage and computation efficiency. Specifically, the enabled integrity tag deduplication offers us low cloud storage cost and also low computation overhead for the tag generation on the user side, with minimal computation cost during the proof generation/verification phase.

VIII. Conclusion

In this paper, we propose a novel serverless message-locked integrity auditing scheme for encrypted deduplication storage. Our design is very suitable for the outsourcing model, from which each participant can reap the benefit. For cloud users, we offer the data confidentiality and integrity guarantees at the same time while incurring minimal computation overhead. On the other hand, the cloud can still leverage the deduplication technique to reduce its operating cost. The security analysis shows that our scheme is provably secure under the CDH assumption in the random oracle model. Experimental results demonstrate its efficiency, effectiveness, and practicality.

Acknowledgment

The work of Liu, Pei and Zhang was supported by NSFC grants 61402352, U1636209, U1401251, National Key R&D Program of China grant 2016YFB0800601, and China 111 Project B16037. The work of Sun and Lou was supported in part by the US NSF grants CNS-1446478, CNS-1405747, CNS-1217889.

References

[1] J. Gantz and D. Reinsel, "The digital universe decade–are you ready (2010)," *http://www.emc.com/collateral/analyst-reports/idc-digital-universe-are-you-ready.pdf*, 2012.

[2] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *Proc. of IEEE INFOCOM*, 2015, pp. 2110–2118.

[3] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proc. of ACM ASIACCS*, 2013, pp. 71–82.

[4] H. Shacham and B. Waters, "Compact proofs of retrievability," in *Proc. of ASIACRYPT*, 2008, pp. 90–107.

[5] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proc. of ACM CCS*, 2007, pp. 598–609.

[6] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," in *Proc. of IEEE INFOCOM*, 2014, pp. 226–234.

[7] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. of ACM SecureComm*, 2008, pp. 1–10.

[8] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM TISS*, vol. 17, no. 4, p. 15, 2015.

[9] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *Proc. of ESORICS*, 2009, pp. 355–370.

[10] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Dynamic audit services for integrity verification of outsourced storages in clouds," in *Proc. of ACM SAC*, 2011, pp. 1550–1557.

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

[12] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proc. of EUROCRYPT*, 2013, pp. 296–312.

[13] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, "Message-locked encryption for lock-dependent messages," in *Proc. of CRYPTO*, 2013, pp. 374–391.

[14] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proc. of IEEE ICDCS*, 2002, pp. 617–624.

[15] J. Li, D. Xie, and Z. Cai, "Secure auditing and deduplicating data in cloud," *IEEE TC*, vol. 65, no. 8, pp. 2386–2396, 2016.

[16] J. Yuan and S. Yu, "Secure and constant cost public cloud storage auditing with deduplication," in *Proc. of IEEE CNS*, 2013, pp. 145–153.

[17] G. Ateniese and S. Hohenberger, "Proxy re-signatures: new definitions, algorithms, and applications," in *Proc. of ACM CCS*, 2005, pp. 310–319.

[18] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE TIT*, vol. 22, no. 6, pp. 644–654, 1976.

[19] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proc. of ASIACRYPT*, 2001, pp. 514–532.

[20] B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," *IEEE TCC*, vol. 2, no. 1, pp. 43–56, 2014.

[21] J. Yuan and S. Yu, "Efficient public integrity checking for cloud data sharing with multi-user modification," in *Proc. of IEEE INFOCOM*, 2014, pp. 2121–2129.

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

[23] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," *IEEE TPDS*, vol. 23, no. 12, pp. 2231–2244, 2012.

[24] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proc. of ACM CCS*, 2015, pp. 874–885.

[25] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Proc. of USENIX Security*, 2013, pp. 179–194.

[26] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proc. of ACM CCS*, 2011, pp. 491–500.

[27] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. of EUROCRYPT*, 1998, pp. 127–144.

[28] X. Liu, W. Sun, H. Quan, W. Lou, Y. Zhang, and H. Li, "Publicly verifiable inner product evaluation over outsourced data streams under multiple keys," *IEEE TSC, PrePrint*.

[29] J. Katz and Y. Lindell, *Introduction to modern cryptography: principles and protocols*. CRC Press, 2007.

[30] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *Proc. of EUROCRYPT*, 1997, pp. 256–266.

[31] J. M. Wendt, "Getting real about deduplication ratios," *http://www.dcig.com/2011/02/getting-real-about-deduplication.html*, 2011.