Deduplication-friendly Watermarking for Multimedia Data in Public Clouds*

Weijing You[1], Bo Chen[2], Limin Liu[3], and Jiwu Jing[1]

[1] School of Computer Science and Technology, University of Chinese Academy of Sciences (UCAS), Beijing, China
[2] Department of Computer Science, Michigan Technological University, Michigan, United States
[3] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences (CAS), Beijing, China

Abstract. To store large volumes of cloud data, cloud storage providers (CSPs) use deduplication, by which if data from multiple owners are identical, only one unique copy will be stored. Deduplication can achieve significant storage saving, benefiting both CSPs and data owners. However, for ownership protection, data owners may choose to transform their outsourced multimedia data to "protected formats" (e.g., by watermarking) which disturbs deduplication since identical data may be transformed differently by different data owners.

In this work, we initiate research of resolving the fundamental conflict between deduplication and watermarking. We propose DEW, the first secure Deduplication-friEndly Watermarking scheme which neither requires any interaction among data owners beforehand nor requires any trusted third party. Our key idea is to introduce novel protocols which can ensure that identical data possessed by different data owners are watermarked to the same "protected format". Security analysis and experimental evaluation justify security and practicality of DEW.

Keywords: Public Clouds, Multimedia Data, Deduplication, Watermarking, Proofs of Ownership

1 Introduction

Nowadays, outsourcing data to public cloud storage providers (CSPs) like Amazon S3 [1], iCloud [2], Microsoft Azure [3], has become an economical and convenient practice for data owners. A critical issue faced by the CSPs is how to manage the ever-surge volume of data [4]. This can be mitigated by deduplication [5], which ensures that when multiple identical copies are present in the CSPs, only one unique copy will be stored, eliminating unnecessary wasting of storage. According to recent reports [5], deduplication can save up to 87% storage. For multimedia data, deduplication also achieves significant storage saving [6].

Although cloud outsourcing can bring significant benefits to data owners, it also introduces significant security concerns. One concern is losing ownership of their valuable data. To protect ownership, data owners would choose to transform their data to "protected formats" before outsourcing them. One prevalent solution is through encryption. This, however, is cumbersome because both sharing and processing of encrypted data in clouds will be difficult[4]. Fortunately, for multimedia data (noise-tolerant signals such as images/audio/videos), an alternative solution is through digital watermarking [7]. The watermarked data are directly usable by other users, facilitating data sharing in the clouds with copyright protection.

The watermarking, however, creates significant obstacles for deduplication, because: identical data are usually transformed to different "formats" by different owners using their unique secrets, which cannot be deduplicated any more. To resolve this conflict, we initiate investigation of a deduplication-friendly watermarking design. The core idea of our design is to make sure that different data owners will always transform identical data to the same "watermarked format", such that deduplication will not get stuck. The challenge

* This is a full technical report of the paper appeared in ESORICS '20.

[4] Although the homomorphic encryption can be used to process data in public clouds, they are still far from practicality.

is, the data owners usually do not know each other and will not synchronize information (e.g., sharing secrets) among them; without information synchronization, it would be difficult for them to generate the same "watermarked format" for identical data possessed by them individually.

Message-locked encryption (MLE) [8] derives encryption key from message being encrypted, such that identical messages can always be encrypted into identical ciphertexts. Motivated by MLE, an immediate solution towards addressing the aforementioned challenge is to perform watermarking using secret derived by MLE. For identical data possessed by different data owners, the secret being derived will be identical, leading to identical watermark embedding and hence deduplication will not be disturbed. However, most the existing MLE instantiations are problematic, since they either rely on impractical assumptions or suffer from various attacks. The implementation like Dupless [9] requires an additional independent key server which is fully trusted, but such a trusted third party is rarely found in practice. Some MLE implementations [10, 11] require cloud users to synchronize information initially which is also impractical. The only known MLE implementation, *convergent encryption* (CE) [12], does not require the trusted third party and the initial user synchronization, but is shown to be vulnerable to an offline brute-force attack [13], in which the attacker can try all possible content of a file since content space for a fixed-size file is limited.

In this work, for the first time, we adapt MLE to our setting by carefully considering both the nature of watermarking and deduplication. The resulting design, DEW, is the first secure and practical Deduplication-friEndly Watermarking scheme. Our key insights are threefold: 1) We separate secrecy of a watermark into two parts: the watermark location (i.e., where the watermark is embedded) and the watermark bit stream (i.e., the content forming the watermark). CE is used to determine the location only, while the watermark bit stream is picked secretly by the first data owner, and securely propagated to other data owners during deduplication. 2) We design a new Proof of Ownership (PoW) scheme specifically for our purpose. Our new PoW protocol can allow the verifier to verify proofs of PoW without having access to the original file. 3) We design a novel technique which can securely and efficiently propagate the watermark bit stream to other data owners via the untrusted cloud server.

Contributions. We summarize our contributions in the following:

– We are the first to identify as well as resolve the conflict between deduplication and watermarking. The resulting design, DEW, is the first deduplication-friendly watermarking scheme without requiring interactions among data owners or any trusted third parties.

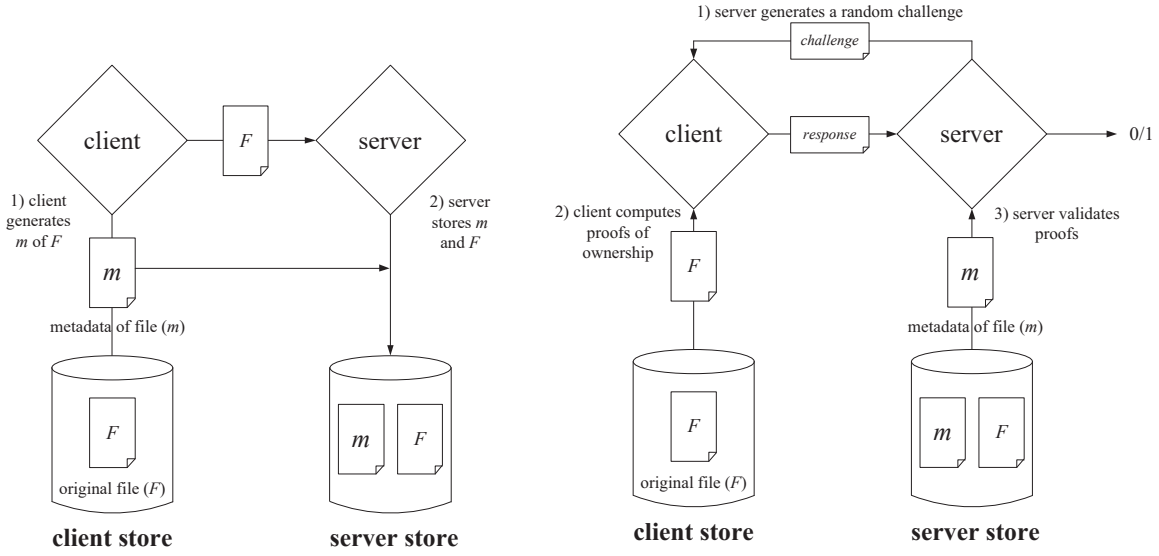
– We adapt MLE in the watermarking process to ensure that identical file data are always deduplicable after watermarking. Additionally, we propose a novel PoW scheme which can allow the cloud server to verify PoW proofs without having access to the original file.

– We analyze security of DEW. Additionally, we implement DEW and experimentally assess its practicality.

2 Background

Deduplication and proofs of ownership. Deduplication aims to eliminate unnecessary storage space by removing duplicate data among cloud users. Note that it is only interested in removing cross-user duplicates which are not used for data durability purpose [14–18]. Based on granularity of data being deduplicated, it can be categorized as *file-level* [12] (i.e., the entire file will be deduplicated) and *block-level* deduplication [19] (i.e. duplicate data will be detected by blocks or chunks). Based on where deduplication is performed, there are *server-side* deduplication, in which all files are uploaded and deduplicated by the cloud server transparently to clients, and *client-side* deduplication, where each client collaborates with the cloud server to perform deduplication. Specifically, the first client uploads the file normally; the following client checks if the file has been stored using a unique checksum of the file. If so, the file will not be uploaded again. Instead, the cloud server simply accepts the client as an owner of this file. Compared to the server-side deduplication, the client-side deduplication is more advantageous in saving both storage and bandwidth, and is thus used broadly in practice (e.g., Dropbox [20]).

However, the client-side deduplication faces a unique attack that, an adversary which obtains the checksum of a file (rather than the file itself), can claim ownership of it. Proofs of Ownership (PoW) [21], were explored to mitigate this attack. A PoW protocol allows the cloud server (i.e., the verifier) to check whether



(a) Pre-processing and store (b) A Proofs of Ownership (PoWs)

Fig. 1. Client-side deduplication system

a client (i.e., the prover) indeed possesses the entire file. Halevi et al. [21] proposed the first PoW scheme based on Merkle-tree, where each leaf node in the tree is a hash value of a file block, and each non-leaf node is hash value of its child nodes. In their scheme (as shown in Figure 1), the verifier is assumed to be fully trusted, and keeps a small amount of metadata for verification. When a prover comes, the verifier challenges the prover, and the prover returns a PoW proof. If the proof is validated by the verifier, the prover will passes the check.

Digital watermarking. Digital watermarking is a kind of marker covertly embedded in a noise-tolerant signal (e.g. audios, videos, or images), and can be used to identify ownership of such signal (e.g., protecting authenticity or owners' copyright). Its core idea is to embed specific digital information in a carrier signal which is only perceptible under certain conditions. The watermark extraction process is the reverse operation of watermark embedding. The digital watermarking can be classified in terms of embedding domain (e.g., spatial/ frequency domain [22]), embedding method (e.g., spread-spectrum/ amplitude modulation/ quantization type), detection method (e.g, plaintext/ blind watermark), robustness (e.g., fragile/ robust watermark), information capacity (e.g., 1-bit/ n-bit watermark), and so on. In this paper, we choose n-bit watermarking in the spatial domain using amplitude modulation[5].

Message-locked encryption (MLE). Bellare et al. [8] proposed the message-lock encryption (MLE), which formalizes various cryptographic primitives that can be used to derive keys from messages being encrypted. By using MLE, different users owning an identical file are able to derive the same encryption key. A typical MLE scheme is convergent encryption (CE) [12], in which the encryption key is the hash value of the message being encrypted. Due to the finite content space, CE is vulnerable to an offline brute-force attack [13].

Discrete logarithm problem (DLP). Let $\mathbb{G}$ be a finite cyclic group of order q, and g be a generator of $\mathbb{G}$. Given an element $y \in \mathbb{G}$, we want to find an integer x such that $g^x = y$ and $0 \leq x < q - 1$. This problem is computationally intractable, i.e., no algorithm can determine x in polynomial time [23].

[5] A spread-spectrum watermark is modestly robust, but has a low information capacity; while a quantization watermark suffers from low robustness, though has a high information capacity. We thus choose amplitude modulation which can achieve a good balance between robustness and information capacity, and is particularly embedded in the spatial domain.

3 System and Adversarial Model

System model. We consider a cloud storage system consisting of three entities: the cloud server (S), the data owner (O), and the data user (U). S provides storage services and enables client-side deduplication. O outsources multimedia files to S but wants to retain their ownership, and therefore, O will watermark the files before outsourcing. Upon outsourcing a file, O will work with S to find out if this file has been stored by S. For a file which has been stored in S, O will not upload it, and S will simply add O to owner list of the file. U can use the outsourced files, e.g. viewing a watermarked video/picture. However, U is not supposed to have access to the original file data due to protection by watermark.

Adversarial model. All data owners are assumed to be fully trusted. Otherwise, they can simply disclose the original file data, and using watermark to protect the original file will become useless. This assumption is reasonable, since a rational data owner will care about his/her ownership and has no motivation to disclose the original file data. We also assume that it is an authentic data owner which initializes upload of the original file. This assumption is also reasonable, since by uploading an arbitrary file, the adversary will need to pay for the storage service, and not gain any additional advantage on obtaining original data of any other files.

Both the cloud server and the data users are not trusted and may misbehave. The cloud server is honest-but-curious [24], which will honestly store the file, correctly execute all required operations, and timely respond to data owners/users as promised in the service-level agreement (SLA), but it is curious about the exact content of the original file, and may try all means to figure it out. The data users may be malicious. Their malicious behaviors include: removing the watermark to obtain the original file data, becoming owner of the file by fooling the cloud server, etc. However, we assume the cloud server and the data users *will not collude*. Otherwise, the cloud server can simply add the data users to the owner list of a file. This assumption is consistent with our "honest-but-curious" cloud model, considering that collusion is not an honest behavior.

We do not consider a special attacker, namely, a revoked data owner which still keeps the original file after being revoked, since this type of revoked data owner can simply use the preserved original file to pass the PoW check and become a data owner again. But we do consider an attacker which is a revoked data owner and has deleted the original file locally. Since a major concern of this work is ownership protection, attacks which are not directly related to ownership compromise are not considered here, like network attacks (e.g., DDoS), attacks that break robustness of watermarking (e.g., the adversary distorts the watermarked file arbitrarily).

All the communication channels among the cloud server, the data owner, and the cloud user are assumed to be secure, e.g., being protected using SSL/TLS.

4 A Deduplication-Friendly Watermarking Scheme

4.1 Design Rationale

To enable a secure deduplication-friendly watermarking design, we separate the watermark into two independent components: the watermark location and the watermark content. We handle each component as follows:

- The watermark location is derived from a secret key, which is generated using convergent encryption (CE). In this manner, if all the data owners possess an identical file, by applying CE on this file, they will be able to generate the same secret key, which can be used to derive the same watermark location.
- The watermark content is a random bit stream, which will be picked by the data owner who initially uploads the file, and will then be propagated secretly to other data owners during the client-side deduplication. In this way, all the data owners of the file can share the same random bit stream. Note that relying on the CE to derive both the watermark location and the watermark bit stream will be vulnerable to the offline brute-force attack [13].

Especially, in a client-side deduplication system, a data owner initially outsources a watermarked multimedia file to a cloud server. All the following data owners will check with the cloud server whether the file has been stored or not, and the cloud server will perform a PoW check on each following data owner; once the PoW check is passed, the corresponding data owner will be added to owner list of the file, and can delete the file locally without uploading it again. During each successful PoW process, the random bit stream will be propagated stealthily to each data owner. Three questions need to be answered further:

- How can each following data owner securely and efficiently find out whether a given file has been stored in the cloud server?
- How can the cloud server perform the PoW check without having access to the original file?
- How can the random bit stream (i.e., watermark content) be propagated to other data owners securely and efficiently via the untrusted cloud server?

We answer the *first* question. In traditional client-side deduplication (in a benign setting where the cloud server is fully trusted), the data owner can simply send a checksum (typically a hash value) of the original file to the cloud server before uploading it, using which the cloud server can check whether there is a duplicate file. However, in an adversarial setting, the checksum should not be computed over either the original file or the watermarked one. Simply computing a checksum over the original file is not secure, since the honest-but-curious cloud server may perform the offline brute-force attack by knowing this checksum due to the limit content space of a file. In addition, simply computing a checksum over the watermarked file is infeasible, because at this point, the watermarked file has not been generated yet. Fortunately, we observed that in a watermarked file, the file data excluding the watermark (we call this part of file data "public portion") are always known by the cloud server, and the checksum can be derived over the public portion of the file. In this manner, when performing the offline brute-force attack, the adversary can at most find out the public portion of the file, rather than the entire original file. In addition, since the watermark location is derived using CE, any data owners who possess the original file can easily derive the watermark location, exclude the watermark, and generate the checksum based on the public portion. Since the public portion is only part of the original file, matching of it does not provide 100% guarantee of matching of the file. In the case that two different files have the same public portion (the probability should be low), the mismatching will be detected later when the data owner proves ownership of the file. Therefore, to support existence detection of a particular file, we need to perform the following: 1) when outsourcing a watermarked file, the data owner should compute a checksum over its public portion, and upload to the server both the watermarked file and the checksum; and 2) during client-side deduplication, a data owner who wants to check whether a local file has been stored by the cloud server or not, will compute a checksum over the public portion of file, and send the checksum to the cloud server.

We now answer the *second* question. Suppose the watermarked file has been uploaded to the cloud server by a data owner (O_1) and another data owner (we call this data owner O_l, where $1 < l$) comes, who also possesses the same original file. After having found out the data owner may have a duplicate file (refer to our answer to the first question), the cloud server will run a PoW protocol to further verify whether O_l really possesses the original file. In the traditional PoW protocol [21] the cloud server is fully trusted and possesses the original file, which is different from our setting that the cloud server is assumed to be honest-but-curious and only stores the watermarked file. Therefore, the traditional PoW protocol cannot work and a new PoW protocol is designed as: the data owner generates a "miniature" of the original file to assist the cloud server to run the PoW protocol, under the condition that, the cloud server (i.e., PoW verifier) will not learn the original file from either the "miniature" or the PoW proofs, and the client (i.e., PoW prover) should not be able to pass the PoW verification without possessing the original file.

Specifically for security, the "miniature" should satisfy two properties: I) The "miniature" can be used to correctly verify a PoW proof, i.e., the "miniature" should be unique for the original file, and it is computationally infeasible to find two different files such that their miniatures are identical. II) The cloud server should not be able to derive anything else beyond the watermarked file from the "miniature". To create such a "miniature", an immediate solution is to use PoR [25]/PDP [26] tags which support public verifiability. This, however, turns out to be very expensive in tag generation [25, 26]. An improvement of efficiency could be using the PoR tags supporting private verifiability [25] (note that a PDP tag supporting private

verifiability is also expensive to be computed [26]). The privately verifiable PoR tag [25] is constructed as $\sum_{j=1}^{s} \alpha_j m_{ij} + F_\kappa(i)$, where s is the number of symbols in a file block m_i (with block index i, and $1 \leq i \leq n$ for a file with n blocks), $F_\kappa()$ is a pseudo-random function (PRF) with a key κ. Note that α_j (for $1 \leq j \leq s$) and κ are random numbers which should be kept private by the verifier. However, the privately verifiable PoR tag cannot immediately work here, since the verifier (i.e., the untrusted cloud sever here) needs to know α_j (for $1 \leq j \leq s$) and κ. By knowing α_j (for $1 \leq j \leq s$) and κ, the cloud server may be able to derive the original file content. To address this security issue, the design is tuned as follows: 1) The tag of each file block is computed as: $\sum_{j=1}^{s} \alpha_j m_{ij} + H(i||I_{wm})$, where H is a cryptographic hash function, and I_{wm} is the watermark bit stream of the file. 2) We disclose g^{α_j} (for $1 \leq j \leq s$) rather than α_j (for $1 \leq j \leq s$) to the untrusted cloud server, such that the cloud server can rely on g^{α_j} to check the PoW proofs without knowing α_j. 3) During the PoW process, we require the cloud server to stealthily distribute the watermark bit stream to the prover. Here "stealthily" means the cloud server is unable to figure out the watermark bit stream. The prover, if possesses the original file, will be able to extract the watermark bit stream I_{wm}, and to compute assisting information based on the I_{wm} together with its PoW proofs. This makes it possible that even if the verifier (i.e., the cloud server) does not know I_{wm}, it can still be able to check the PoW proofs returned by provers.

In addition, during each PoW process, the cloud server checks a random subset of file blocks from the original file (for a small file, the cloud server can simply check all the file blocks); the prover will return PoW proofs derived from file blocks being checked. A possible security issue here is that, the cloud server may learn sensitive information about the original file from the PoW proofs. Our solution is that, each time before returning a PoW proof, the prover will mask the proof with randomness; also, to facilitate the verification of the PoW proofs, assisting information relating to the randomness will be returned (can be combined together with the assisting information for I_{wm} mentioned above).

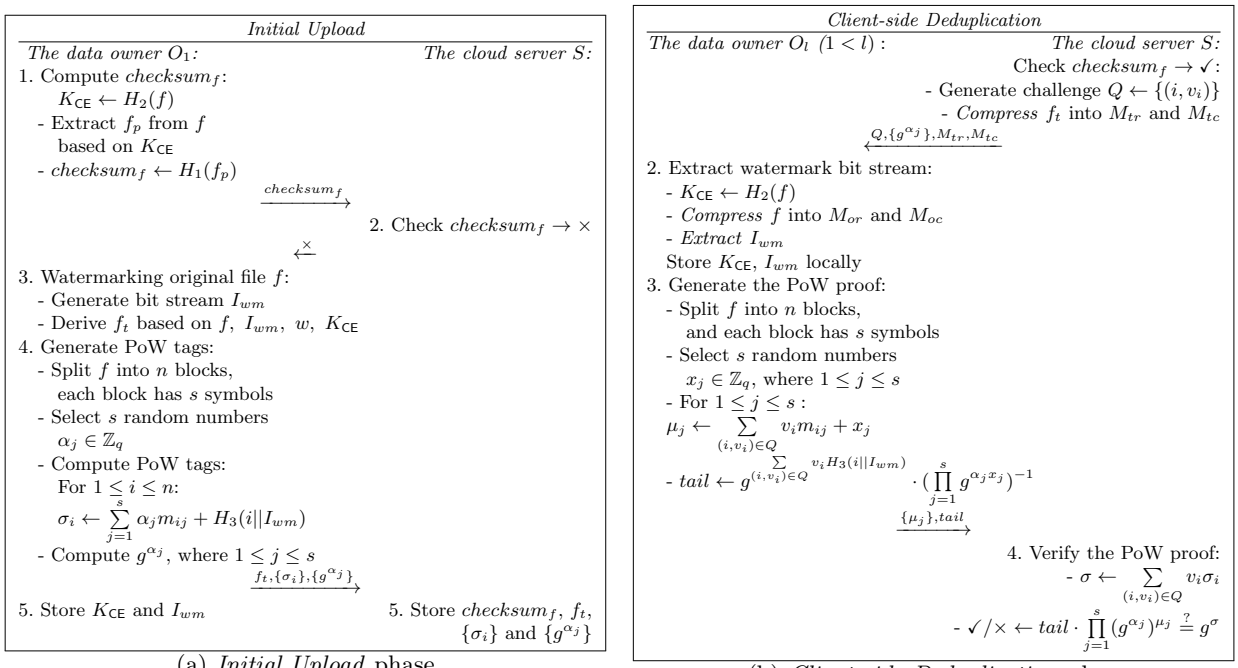
Here comes to the *third* question. Having observed that the watermark can be extracted through comparing the original file with the watermarked file, a straightforward solution is, the cloud server directly sends the watermarked file to the data owner during each PoW process. This solution suffers from a large communication overhead and, is insecure, since it discloses too much effective information of the original file to the prover before PoW checking. Having observed that, the watermark bit stream usually distributes sparsely in the watermarked file, we propose a novel approach to "compress" the watermarked file such that: 1) the size of the compressed watermarked file can be reduced significantly from $O(n)$ to $O(\sqrt{n})$, where n is the file size; and 2) without possessing the original file, one cannot extract the watermark bit stream from the compressed watermarked file (i.e., only a party which possesses the original file can extract the watermark bit stream). As a grayscale image[6] can be processed as a 2-dimension $N \times N$ pixel matrix, the key idea is to compress the matrix to a $d \times N$ row matrix and an $N \times d$ column matrix, where d is a compression parameter and $d \ll N$. The compression is performed in such a way that, compared to the original pixel matrix, the compressed matrices lose effective information significantly and are useless when used independently. Note that this special "compression" is completely different from the traditional compression which requires decompression, and therefore, it is also effective for image files in compressed format (e.g. jpeg).

4.2 Design Details

Our Deduplication-friEndly Watermarking scheme (DEW) consists of 5 phases: *Setup*, *Initial Upload*, *Client-side Deduplication*, *Retrieval*, and *Restoration*. For simple presentation, we assume there is only one file in the cloud server, which can be easily extended to a regular cloud storage system with multiple files. In addition, we focus on grayscale images which are commonly stored with 8 bits per sampled pixel.

Setup: Let β be a security parameter. The system selects a finite field $\mathbb{Z}_q$ of a prime order q, and initializes a multiplicative cyclic group $\mathbb{G}$ of the same order, and let $g \in \mathbb{G}$ be a generator. The system also selects three cryptographic hash functions, defined as H_1, H_2, and H_3 respectively: $\{0,1\}^* \rightarrow \{0,1\}^\beta$. There is also a

[6] For simplicity, we focus on grayscale images in this paper.



(a) *Initial Upload* phase

(b) *Client-side Deduplication* phase

Fig. 2. Detailed steps in the *Initial Upload* and *Client-side Deduplication* phase

pseudo-random permutation π: $\{0,1\}^{\beta} \times \{0,1\}^{log_2^n} \rightarrow \{0,1\}^{log_2^n}$, where n is file size in bytes. The compression parameter d for determining the size of compressed matrix is set to be a positive integer significantly less than n, i.e., $d \ll n$, and the watermark length is set as w, which usually is much smaller than n.

Initial Upload: In this phase, the data owner (O_1) processes the original file (f) into a deduplicable watermarked file (f_t), generates PoW tags for f, and stores PoW tags together with f_t to the cloud server (S) (Figure 2(a)):

– Step 1: O_1 runs $K_{CE} \leftarrow H_2(f)$ to derive a CE key from f. K_{CE} is used to derive the public portion, denoted as f_p. O_1 then computes a checksum (i.e. $checksum_f$) for f by running $checksum_f \leftarrow H_1(f_p)$. Lastly, O_1 sends a request with $checksum_f$ to S indicating that he/she wants to upload f.

– Step 2: S checks existence of f using $checksum_f$. If f has been stored, the *Initial Upload* phase will end, and *Client-side Deduplication* phase will be be invoked. Otherwise (denoted as $\times$), proceed to next Step 3.

– Step 3: O_1 generates a random bit stream (I_{wm}) with length w and uses Algorithm 1 (input: f, I_{wm}, w, K_{CE}) to generate a watermarked file f_t.

– Step 4: O_1 splits f into n blocks and each block contains s symbols in $\mathbb{Z}_q$. Each symbol is denoted as m_{ij}, where $1 \le i \le n$ and $1 \le j \le s$. Then O_1 selects s random numbers $\alpha_j \in \mathbb{Z}_q$ and computes g^{α_j}, for $1 \le j \le s$. The PoW tag for each file block i is computed as $\sigma_i = \sum_{j=1}^{s} \alpha_j m_{ij} + H_3(i||I_{wm})$, for $1 \le i \le n$.

– Step 5: f_t is outsourced to S along with $\{g^{\alpha_j} | 1 \le j \le s\}$ and $\{\sigma_i | 1 \le i \le n\}$. Then O_1 deletes f and stores K_{CE} and I_{wm} locally.

Client-side Deduplication: If f has been stored by the cloud server, when a following data owner (O_l, where $1 < l$) sends the checksum (i.e., $checksum_f$), the cloud server will find a match. Then, the cloud server will ask O_l to prove ownership of f. The detailed steps (Figure 2(b)) are elaborated as follows:

– Step 1: S generates a PoW challenge $Q = \{(i, v_i) | 1 \le i \le c\}$, where i denotes a randomly selected block index, v_i denotes a random number picked from $\mathbb{Z}_q$, and c is the number of file blocks being challenged. S further compresses f_t using Algorithm 2 (input: f_t and d), generating a compressed row matrix M_{tr} and a compressed column matrix M_{tc}. S sends Q, M_{tr}, M_{tc}, $\{g^{\alpha_j} | 1 \le j \le s\}$ to O_l.

– Step 2: O_l extracts I_{wm}: 1) O_l computes K_{CE} from f by running $K_{CE} \leftarrow H_2(f)$. 2) O_l compresses the original file f following the Algorithm 2 (input: f and d), obtaining a compressed row matrix M_{or} and a

Algorithm 1: Watermarking an image file (for grayscale images)

Input: An image file f, I_{wm}, watermark length w, K_{CE}
Output: A watermarked image file f_t
View the file f as an $N \times N$ pixel matrix M
for $i = 1 : w$ **do**
  $t = \pi_{K_{\mathsf{CE}}}(i)$;// Derive watermark locations
  $L[i] = ((t-1)/N + 1, (t-1) \bmod N + 1)$;// Derive watermark index
k=1;
for $i = 1 : N$ **do**
  for $j = 1 : N$ **do**
    if $L[k] == (i, j)$ **then**
      $M[i, j] = (M[i, j] + I_{wm}[k])(\text{mod } 256)$;// embed a watermark bit
      ++k;
The new matrix M will be output as a watermarked image file f_t.

compressed column matrix M_{oc}. 3) O_l runs Algorithm 3 (input: f, M_{or} and M_{oc}, M_{tr} and M_{tc}, d, w, K_{CE}) to obtain I_{wm}. K_{CE} and I_{wm} will be stored locally.

– Step 3: O_l generates the PoW proof according to Q: 1) O_l divides the file f into n blocks, and each block contains s symbols. Each symbol is m_{ij} where $1 \le i \le n$ and $1 \le j \le s$. 2) O_l selects s random numbers x_j from $\mathbb{Z}_q$, where $1 \le j \le s$ and computes: $\mu_j = \sum_{(i,v_i)\in Q} v_i m_{ij} + x_j$, for $1 \le j \le s$. 3) O_l computes $tail = g^{\sum_{(i,v_i)\in Q} v_i H_3(i||I_{wm})} (\prod_{j=1}^{s} g^{\alpha_j x_j})^{-1}$. O_l replies to S with the PoW proof: $\{\mu_j | 1 \le j \le s\}$, $tail$.

– Step 4: After having received the PoW proof, S validates its correctness: 1) S computes $\sigma = \sum_{(i,v_i)\in Q} v_i \sigma_i$, and then computes g^{σ}. 2) S checks whether or not equation $g^{\sigma} = tail \cdot \prod_{j=1}^{s} (g^{\alpha_j})^{\mu_j}$ holds.

Algorithm 2: Compressing an image file (for grayscale images; $M_r[i, :]$ denotes all the N elements in row i of matrix M_r; $M_c[:, i]$ denotes all the N elements in column i of matrix M_c)

Input: An image file f, compression parameter d
Output: A compressed row matrix M_r, and a compressed column matrix M_c
View the file f as an $N \times N$ pixel matrix M
for $i = 1 : d$ **do**
  $M_r[i, :] = 0$;
  $M_c[:, i] = 0$;
  for $j = 0 : \frac{N}{d} - 1$ **do**
    $M_r[i, :] = (M[i + d \times j, :] + M_r[i, :])(\text{mod } 256)$;// aggregated by row
    $M_c[:, i] = (M[:, i + d \times j] + M_c[:, i])(\text{mod } 256)$;// aggregated by column

Retrieval: All valid data owners and data users are able to retrieve the f_t from the cloud server. Upon receiving a retrieval request, the cloud server authenticates the request (not the focus of this paper), and sends back f_t.

Restoration: Only data owners (O_l, where $1 \le l$) are able to restore f from f_t, which is an inverse operation of the watermark embedding process in the *Initial Upload* phase. Specifically, O_l downloads f_t from the cloud server and removes I_{wm} from watermark locations determined by K_{CE} to restore f.

Algorithm 3: Extracting watermark bit stream I_{wm}

Input: The original file f, M_{or}, M_{oc}, M_{tr}, M_{tc}, d, w, K_{CE}
Output: The extracted watermark bit stream I_{wm}
Initiate an $N \times N$ matrices M_1, and $L \leftarrow \emptyset$;
for $i = 1 : w$ **do**
  $t = \pi_{K_{\mathsf{CE}}}(i)$;
  $L = L \cup \{(t-1)/N + 1, (t-1) \bmod N + 1)\}$;
for $i = 1 : N$ **do**
  for $j = 1 : N$ **do**
    if $(i, j) \in L$ **then** $M_1[i, j] = 2$;// set 2 for watermark locations
    else
      $M_1[i, j] = 3$;// set 3 for other locations
for $i = 1 : d$ **do**
  for $j = 1 : N$ **do**
    $M_r[i, j] = (M_{tr}[i, j] - M_{or}[i, j])(\text{mod } 256)$;
    $M_c[j, i] = (M_{tc}[j, i] - M_{oc}[j, i])(\text{mod } 256)$;
for $i = 1 : d$ **do**
  for $j = 1 : N$ **do**
    $t = 0$;
    for $k = 0 : \frac{N}{d} - 1$ **do**
      if$(M_1[k \times d + i, j] == 2)$ **then** $++t$;
    if $t == 0$ **then continue**;
    if $M_r[i, j] == 0$ **then**
      for $k = 0 : \frac{N}{d} - 1$ **do**
        if $M_1[k \times d + i, j] == 2$ **then** $M_1[k \times d + i, j] = 0$;
    else if $M_r[i, j] == t$ **then**
      for $k = 0 : \frac{N}{d} - 1$ **do**
        if $M_1[k \times d + i, j] == 2$ **then**
          $M_1[k \times d + i, j] = 1$;
          $M_c[k \times d + i, (j-1) \bmod d + 1] -= 1$;
for $i = 1 : N$ **do**
  for $j = 1 : d$ **do**
    $t = 0$;
    for $k = 0 : \frac{N}{d} - 1$ **do**
      if $(M_1[i, k \times d + j] == 2)$ **then** $++t$;
    if $t == 0$ **then continue**;
    if $M_c[i, j] == 0$ **then**
      for $k = 0 : \frac{N}{d} - 1$ **do**
        if $M_1[i, k \times d + j] == 2$ **then** $M_1[i, k \times d + j] = 0$;
    else if $M_c[i, j] == t$ **then**
      for $k = 0 : \frac{N}{d} - 1$ **do**
        if $M_1[i, k \times d + j] == 2$ **then** $M_1[i, k \times d + j] = 1$;
For remaining elements with value 2 in M_1, retrieve corresponding elements from f_t, determine them by comparing with f. Sequentially output elements with value 0 and 1 from matrix M_1 as I_{wm}.

5 Analysis and Discussion

5.1 Security Analysis

DEW aims to protect ownership of the outsourced multimedia files in a client-side deduplication-enabled cloud storage system, which implies: 1) any untrusted party, both the honest-but-curious cloud server and the malicious data users, should not obtain the original file; and 2) any party which does not possess the original file should not be able to pass the PoW check. Security of DEW is captured in Theorem 1, 2 and 3.

Theorem 1. *The cloud server cannot obtain the original file.*

Proof. (sketch) The could server can have access to: 1) the watermarked file (f_t), and 2) the checksum of the file ($checksum_f$), and 3) the PoW tags, g^{α_j}, as well as the PoW proofs, and 4) some of the watermark locations during the process of extracting watermark bit stream in Algorithm 3. In the following, we analyze the cloud server could not obtain the original file based on the aforementioned information:
1. By having access to f_t, the cloud server cannot obtain the original file by removing the watermark directly. Without any additional knowledge, the only strategy for the cloud server to obtain the original file is either brute-force guessing the original file directly or conducting a removal attack in which brute-force guessing the watermark location (denoted as event X) and content (denoted as event Y) simultaneously. However, possessing nothing other than f_t the cloud server cannot determine the correctness of brute-force guessing, which means restoring the original file from the watermarked file directly is infeasible.
2. By additionally having the $checksum_f$, which implies the knowledge of watermark location, the cloud server could not obtain the original file. Recall that, the watermark is determined by two independent factors, i.e., the watermark location and the watermark content. Only when exact watermark content is removed from exact location can the watermarked file be restored to the original file correctly. Based on $checksum_f$ and f_t, the strategy for the cloud server to restore the original file is still brute-force guessing, but $checksum_f$ can help the cloud server to determine if the guessed watermark location is correct. Let event A_2 denotes one successful removal attack when given $checksum_f$, the probability of which is multiplication of probability of X and Y as the two events are independent. Suppose c_0, l_0 be total number of elements in the image matrix and the length of watermark bit stream ($l_0 \leq c_0$), respectively. The probability of identifying watermark location then is $P(X) = \frac{1}{\binom{c_0}{l_0}}$, and the probability of event Y is: $P(Y) = \frac{1}{2^{l_0}}$. Therefore, the probability of A_2 is:

$$P(A_2) = P(X \bigcap Y) = P(X) \times P(Y) = \frac{1}{2^{l_0}} \times \frac{1}{\binom{c_0}{l_0}} < \frac{1}{2^{l_0}}.$$

It is clearly that when l_0 is large enough, the probability of the removal attack is negligible even when $checksum_f$ is disclosed to the cloud server.
3. We show that by having access to the PoW tags and g^{α_j} only, the cloud server cannot obtain the original file. Assume the file has n blocks, and n PoW tags. Each tag is computed as: $\sigma_i = \sum_{j=1}^{s} \alpha_j m_{ij} + H_3(i||I_{wm})$, for $1 \leq i \leq n$. The goal of the cloud server is to identify the value of those symbols (i.e., m_{ij}) which are covered by watermark bit stream. Let σ_k be a tag of a file block which contains at least one symbol covered by the watermark bit stream. Although the cloud server can have access to σ_k (computed as $\sum_{j=1}^{s} \alpha_j m_{kj} + H_3(k||I_{wm})$), it cannot have access to $H_3(k||I_{wm})$, i.e., $\sum_{j=1}^{s} \alpha_j m_{kj}$ will remain unknown by the cloud server. Also, knowing other tags except σ_k will not help, since m_{kj} will not appear in other blocks. Therefore, to identify m_{kj}, the cloud server can only perform brute-force guessing, with a successful probability at most $\frac{1}{q}$, which is negligibly small since q is a large prime number (e.g., for 160-bit security, $\frac{1}{q}$ will be approximately 2^{-160}).

We next show that, by accumulating the PoW proofs, the cloud server still cannot obtain the original file. Note that: 1) The cloud server is assumed to be honest but curious (Sec. 3), and will strictly follow the

protocol during the PoW, e.g., issuing a challenge which selects a random subset of file blocks with a new set of random coefficients during each PoW execution. 2) During each PoW execution, the prover (note that the prover will not collude with the verifier) will mask each μ_j (where $1 \leq j \leq s$) with a newly generated random number. 3) Only when the PoW protocol is successful, the accumulated PoW proof may help (i.e., the data owner which gets involved into the PoW actually possesses the original file). After each successful PoW execution, the cloud server will accumulate a new PoW proof μ_1, μ_2, ..., μ_s as well as $tail$. Note that if the blocks being challenged during the aforementioned PoW execution contains any symbols which are covered by the watermark bit stream, the corresponding proof may help. But we show in the following that accumulating such a proof gives the cloud server negligible advantages. The detailed proof obtained upon a successful PoW execution is:

$$\mu_1 \leftarrow \sum_{(i,v_i)\in Q} v_i m_{i1} + x_1$$
$$\mu_2 \leftarrow \sum_{(i,v_i)\in Q} v_i m_{i2} + x_2$$
$$...$$
$$\mu_s \leftarrow \sum_{(i,v_i)\in Q} v_i m_{is} + x_s$$
$$tail \leftarrow g^{\sum_{(i,v_i)\in Q} v_i H_3(i||I_{wm})} \cdot (\prod_{j=1}^{s} g^{\alpha_j x_j})^{-1}$$

Each μ_j $(1 \leq j \leq s)$ is masked by an independent unknown random number x_j, therefore, the set of $\{\mu_j | 1 \leq j \leq s\}$ alone does not bring any advantage for computing m_{ij} where $(i, v_i) \in Q$ and $1 \leq j \leq s$. The only hope is to take advantage of $tail$. But due to hardness of DLP (Sec. 2), the only option for the cloud server is to guess $\{x_j | 1 \leq j \leq s\}$, and to rely on the $tail$ value to verify correctness of each guessing. Clearly, the probability for a successful guessing will be approximately $\frac{1}{q^s}$, which is negligibly small for a large prime number q. In other words, the extra advantage brought by the aforementioned PoW proof is an additional guessing with a negligible successful rate (informally).

4. By obtaining the watermark locations (or some of the watermark locations) during the watermark bit stream extraction (Algorithm 3), the cloud server is not able to obtain the original file since the watermark bit stream still remains unknown (see aforementioned proof for point 2).

Theorem 2. *The malicious data user cannot obtain the original file.*

Proof. (sketch) We do not consider a special data user which is a revoked data owner but still keeps the original file after being revoked (Sec. 3). Therefore, there are two types of malicious data users: 1) The malicious data user obtains the watermarked file f_t, and tries to learn the original file. This has been shown infeasible in the proof of Theorem 1. 2) The malicious data user gets involved into the PoW process by successfully guessing the $checksum_f$ even though he/she does not possess the original file. This type of malicious data users will not be able to obtain more information than the cloud server. Having M_{tr} and M_{tc} cannot be even equivalent to having the watermarked file (f_t), and all the other information the malicious data user can have access to is also accessible to the cloud server. According to Theorem 1, the cloud server cannot obtain the original file, and definitely, the malicious data user cannot obtain the original file either.

Theorem 3. *Any parties which do not possess the original file cannot pass the PoW check.*

Proof. (sketch) In the following, we first prove the correctness of the verification process. We then show that using spot checking (i.e., checking a random subset of the entire file rather than the entire file) can verify whether the prover actually possesses the entire file with a high probability. Lastly, we show that a successful PoW verification ensures that the prover possesses the blocks being challenged.

1. The correctness of the verification process is proved below:

$$
\begin{aligned}
g^{\sigma} &= g^{\sum\limits_{(i,v_i \in Q)} v_i \sigma_i} \\
&= g^{\sum\limits_{(i,v_i \in Q)} v_i (\sum\limits_{j=1}^{s} \alpha_j m_{ij} + H_3(i||I_{wm}))} \\
&= g^{\sum\limits_{j=1}^{s} \sum\limits_{(i,v_i \in Q)} \alpha_j (v_i m_{ij}) + \sum\limits_{(i,v_i \in Q)} v_i H_3(i||I_{wm})} \\
&= g^{\sum\limits_{j=1}^{s} \alpha_j (\mu_j - x_j) + \sum\limits_{(i,v_i \in Q)} v_i H_3(i||I_{wm})} \\
&= g^{\sum\limits_{(i,v_i) \in Q} v_i H_3(i||I_{wm})} (g^{\sum\limits_{j=1}^{s} \alpha_j x_j})^{-1} \cdot g^{\sum\limits_{j=1}^{s} \alpha_j \mu_j} \\
&= g^{\sum\limits_{(i,v_i) \in Q} v_i H_3(i||I_{wm})} (\prod_{j=1}^{s} g^{\alpha_j x_j})^{-1} \cdot \prod_{j=1}^{s} g^{\alpha_j \mu_j} \\
&= tail \cdot \prod_{j=1}^{s} (g^{\alpha_j})^{\mu_j}
\end{aligned}
$$

2. For small files, the verifier simply checks all the file blocks, and can definitely find out if the original file is not possessed by the prover. For large files, for efficiency consideration, the verifier only checks a random subset of file blocks among the entire file. We show in the following that, in this way, the verifier can still detect it if the prover does not possess the entire original file, with a high probability. The special case that the prover possesses the watermarked file is equivalent to the case the a portion of the original file is missing. According to PDP [26], by randomly checking c out of n file blocks, if a malicious prover is missing t blocks, the verifier can detect it with a probability at least $1 - (\frac{n-t}{n})^c$. For example, if $t = 0.01n$, when $c = 460$, the probability is 99%. For a large enough c, $(\frac{n-t}{n})^c \to 0$, and $P \to 1$.
3. During the PoW verification, σ remains unknown to the prover, and hence g^{σ} is also unknown. If the prover is missing one block being challenged, it needs to guess this block when computing the PoW proof μ_1, μ_2, ..., μ_s, $tail$. Without knowing g^{σ}, the prover can only brute-force this missing block, and compute the PoW proof based on this guessed block. The probability that this PoW proof can pass the PoW check successfully is $\frac{1}{q}$, which is negligibly small for a large prime q. In other words, without having possessed all the challenged blocks, the prover is not able to pass the PoW check.

5.2 Discussion

Regarding watermark extraction. During the Client-side Deduplication phase, to ensure watermark bit stream can be extracted from the compressed row and column matrix (i.e., M_{tr} and M_{tc}) sent by the cloud server, O_l may need to retrieve additional elements from the watermarked file f_t, if some of the elements in the watermarked locations cannot be determined (Algorithm 3). The number of additional elements needed to be retrieved is usually much smaller than the size of watermark bit stream according to our experimental evaluation (value r in Table 1), which only introduces a small extra communication overhead. In addition, this may disclose some of the watermark locations to the cloud server, but it will not help the cloud server to learn the original file, because only a small portion of watermark locations may be disclosed and most importantly, the watermark bit stream remains unknown to the cloud server. Note that to ensure Algorithm 3 can work correctly, $\frac{N}{d}$ should not reach 256 (i.e., to ensure that after aggregating all the potential watermark bits in a row/column, it will not be greater than or equal to 256, causing overflow).
Revoking data owners. A data owner may be revoked over time, e.g., after the owner deletes a multimedia file, he/she will be revoked. To prevent a revoked owner from restoring the file in the future by simply keeping the watermark (small in size), we should re-watermark the file so that after re-watermarking process is done, the revoked data owner will not be able to obtain the original file again by retrieving a watermarked file and subtracting the kept watermark. The re-watermarking process is usually triggered upon an owner is

revoked. In the case that a file is shared by many owners which are revoked frequently, we may delay the re-watermarking process so that each such process will handle multiple revoked owners. After the re-watermarking process, the watermark bit stream may have been changed, and should be re-propagated to valid owners[7].

Side-channel attack resistance. Traditional client-side deduplication is susceptible to a side-channel attack [27], where the checksum can be used to identify existence of a given file, and hence to learn file content. DEW can mitigate this attack since the checksum is computed from "public portion" of a file, i.e. the adversary cannot identify a file exactly, and at most learn "public portion" of the file.

Applications of DEW in practice. DEW can be applied to a lot of real-world scenarios. For example, crowd-funding projects become popular nowadays, in which different users may collaborate for one project and produce multimedia files, which are later outsourced to clouds by individual users using watermarking for ownership protection. Due to COVID-19, online teaching becomes more and more prevalent, and individual teachers may store multimedia files (purchased from the same publisher) to clouds, using watermarking for ownership protection.

Limitations of DEW. The design of DEW has a few limitations: First, due to complication of multimedia data, we currently initiate DEW using an n-bit additive modulation watermarking scheme in a spatial domain with modest robustness, which is sufficient regarding ownership protection. However, other watermarking schemes, e.g., watermarking in frequency domain and quantization-based watermarking, may be supported by DEW, by slightly tuning the design to accommodate different types of watermarking process. This will be further investigated in our future work. Second, the current design of DEW uses the grayscale images. However, DEW can be extended to other multimedia data. For example, 1) for a colored image (i.e. RGB image), it can be decomposed to three single-channel images, each can be processed like a grayscale image; 2) for a video file, it can be viewed as a collection of still images (either colored images or grayscale images).

6 Implementation and Evaluation

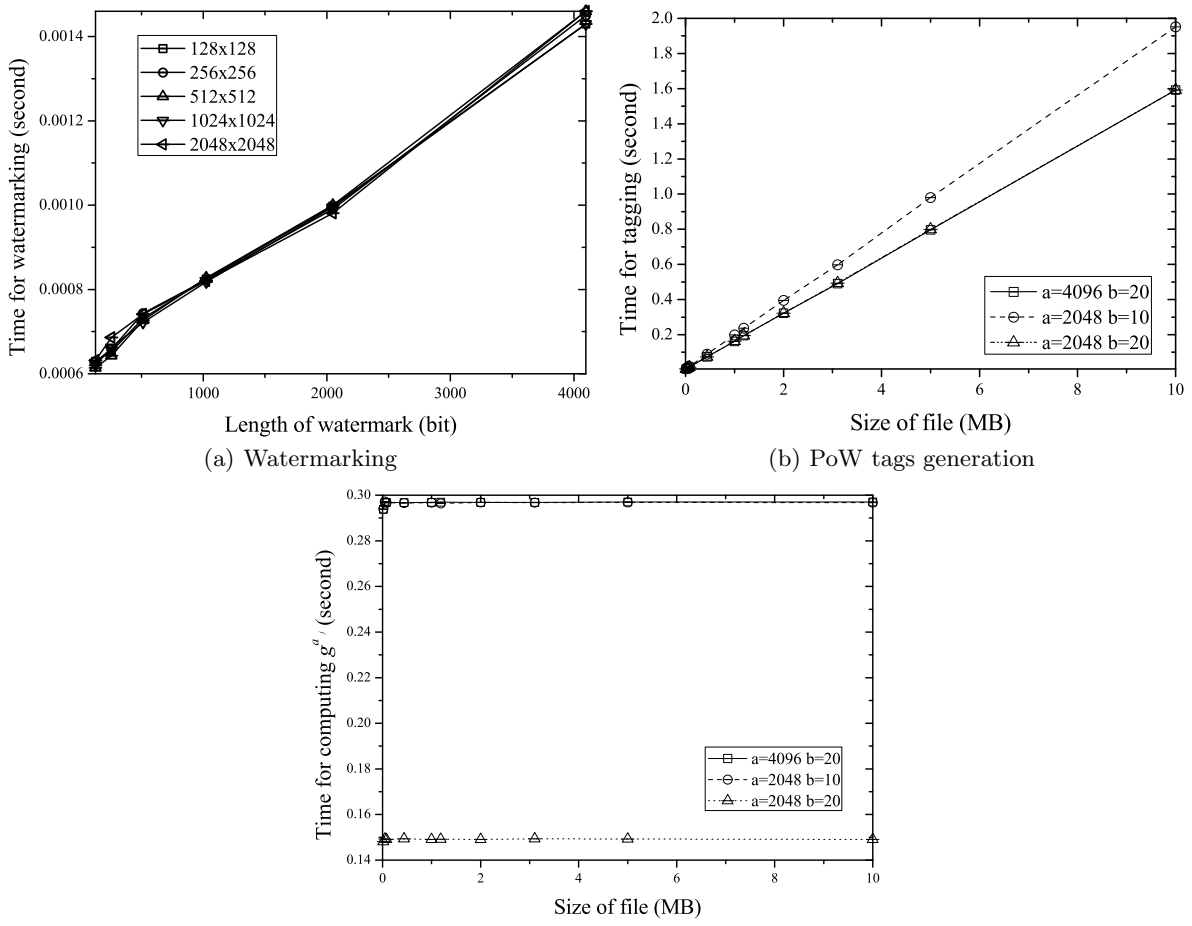
Both the cloud server and the client were run on a local machine (Intel Xeon E3-1225 v5 CPU 3.30GHz, 8.0 GB RAM, and Ubuntu 16.04 with kernel version 4.15.0-50-generic). The security parameter β was selected as 160. We worked on the type A pairings which are constructed on the curve $y^2 = x^3 + x$ [28] over the finite field $\mathbb{Z}_q$. The order (i.e., q) of the finite field $\mathbb{Z}_q$ and the cyclic group $\mathbb{G}$ is a 160-bit prime[8]. We used the pairing-based cryptographic library PBC-0.5.14 [28]. The hash function H_1, H_2 and H_3 were instantiated using SHA1, and the pseudo-random permutation π was implemented using *randomperm*() in MATLAB-R2016a. d was selected as 8, and N were 128, 256, 512, 1024, and 2048, respectively. Note that $N = 2048$ is a corner case for satisfying the condition that $\frac{N}{d} < 256$ (Sec. 5.2), and we ensured that no overflow happened during the experiments. When the number of blocks in a file is less than or equal to 460, the cloud server will run the PoW protocol by checking the entire file; otherwise, the number of blocks being checked is 460, which can ensure that if 1% of the file has been corrupted, the server will detect the corruption with 99% probability [26]. We used 1000+ images for testing, the sizes of which range from 16KB to 10MB, under three test cases with different sizes of each file block and single symbol in a file block. Images were processed using MATLAB scripting language.

Evaluating the Initial Upload phase. In this phase, the data owner will generate the watermarked file, and compute PoW tags (from the original file) as well as $\{g^{\alpha_j} | 1 \leq j \leq s\}$ for PoW verification.

Watermarking. The watermarking computation is shown in Figure 3(a). We can observe that the watermarking computation is mainly determined by the watermark length rather than the file size. This is because, the major computation during watermarking is adding watermark bits to specific elements selected by the K_{CE}, which depends on the length of the watermark bit stream.

[7] Using the old watermarked file as well as the old watermark, a valid owner can restore the original file, and by retrieving a compressed version of the new watermarked file, the owner is able to extract the new watermark bit stream. We will investigate a more efficient design in the future work.

[8] To ensure that each symbol is valid (i.e., not larger than the finite field), only 159 bits from a file were read each time.



(a) Watermarking

(b) PoW tags generation

(c) Computing $\{g^{\alpha_j}\}$

Fig. 3. Computational cost of various components in Initial Upload phase

Preparing for the PoW. The computational cost for generating PoW tags and computing $\{g^{\alpha_j}|1 \leq j \leq s\}$ is shown in Figure 3(b) and 3(c), respectively. Let a denotes the block size (in bytes) and b denotes the symbol size (in bytes). We can observe that: 1) for fixed block/symbol size, the computational cost for generating PoW tags increases linearly with the file size and decreases with the symbol size, while is independent to size of each file block. This is because, this cost is mainly determined by total numbers of symbol-wise operations, the computational complexity of which approximates $O(\frac{|F|}{b})$, i.e., determined mainly by the file size $|F|$ and the symbol size b; 2) for fixed block/symbol size, the time for computing $\{g^{\alpha_j}|1 \leq j \leq s\}$ is independent to the file size. This is because, it is mainly determined by $s = \frac{a}{b}$ rather than the file size. This also explains that, for fixed symbol size, if the block size is doubled, the computation time is also doubled; in addition, for fixed block size, if the symbol size is doubled, the computation time is approximately reduced by 50%.

Evaluating the Client-side Deduplication phase. In this phase, the data owner will compute the PoW proof, and the cloud server will validate correctness of the PoW proof. Besides, the watermark bit stream will be distributed to the data owner during the PoW process, which requires the cloud server to compress the watermarked file and the data owner to extract the watermark bit stream.

PoW proof generation. During the PoW process, the data owner will compute the PoW proof based on the cloud server's challenge and the compressed watermarked file. The computational cost for PoW proof generation is shown in Figure 4(a). We have following observations: 1) The computational cost for PoW proof generation increases linearly with the file size before reaching a certain threshold. This is because, the

cloud server will check all the file blocks when the file size is small (i.e., when the total number of file blocks is smaller than 460), and the number of file blocks being checked depends on the file size for fixed block size; if the file size exceeds the threshold, it will always check 460 file blocks, and therefore, the computational cost turns constant. 2) After reaching the threshold, for a fixed symbol size, the computational cost for PoW proof generation will grow if the block size grows; for a fixed block size, this computational cost will grow if the symbol size decreases. The reason is, when the number of blocks being challenged is fixed, the computation for proof generation is mainly associated to $s = \frac{a}{b}$, where a is the block size and b is the symbol size.

PoW proof verification. After having received the PoW proof, the cloud server will verify its correctness and the computational cost is shown in Figure 4(b). We can observe that: 1) for fixed block size and symbol size, the computational cost of verifying the proof is constant regardless of the file size; 2) for fixed symbol size, this cost will increase if the block size increases; 3) for fixed block size, this cost will increase if the symbol size decreases. The reason is similar to the aforementioned proof generation.

Distributing the watermark bit stream stealthily during PoW. In the PoW process, the cloud server will send two compressed matrices of the watermarked file to the data owner, and the data owner will extract the random watermark bit stream. The computational cost of compression and extracting watermark bit stream are shown in Figure 4(c) and 4(d), respectively. We have following observations: 1) The compression time mainly depends on the file size, rather than the watermark length. This is because, the compression will iteratively add each d rows/columns of the image matrix together, which is determined by the size of the matrix, hence the size of the file. 2) When the file size is fixed, the computational cost of extracting watermark approximately grows linearly with the watermark length, which is the total amount of extracting operations; in addition, for fixed the watermark length, the computational cost grows with the file size, since the extracting process requires traversing the entire file.

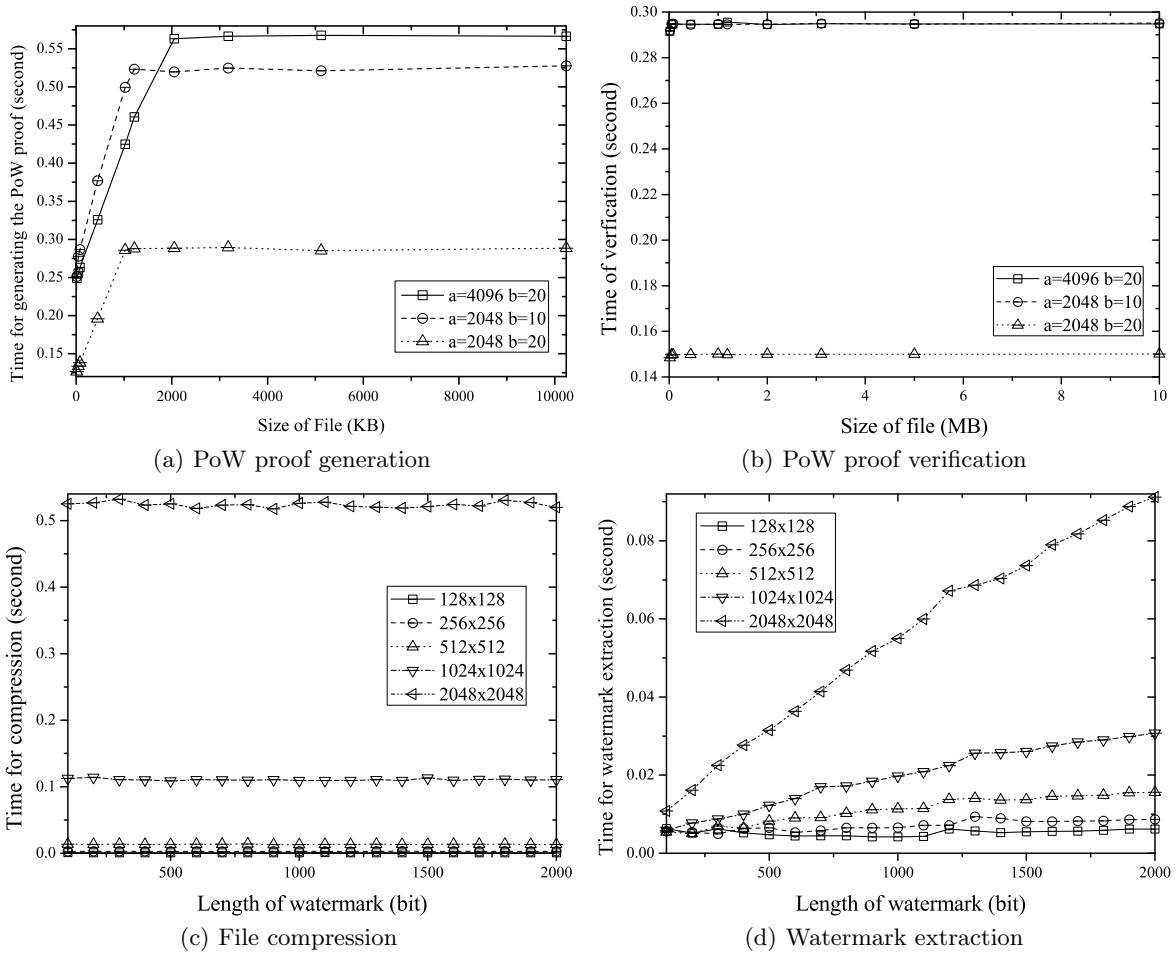
We also assess effectiveness of our compression approach by calculating the compression rate $\rho = \frac{N^2-(2\times d\times N+r)}{N^2} \times 100\%$, where N^2 is the total number of elements in the original matrix, $d \times N$ is the number of elements in a compressed row/column matrix, and r is the number of elements which are sent additionally when uncertainty happen. The experiment was performed over files with different sizes (ranging from 128×128 to 2048×2048 in pixel) and different watermark lengths (ranging from 100 bits to 2,000 bits). The results are shown in Table 1, each is averaged over 100 runs. We can observe that, ρ is at least 84.42%, which justifies effectiveness of our compression approach. Additionally, ρ increases with file size but decreases with watermark length. This is because: 1) for fixed watermark length, the larger file means more space to be embedded with watermark bit stream, leading to the more sparsely distributed watermark bits, hence less uncertainties occur during watermark extraction, i.e., r will be smaller; 2) for fixed file size, a shorter watermark bit stream leads to the more sparsely distributed watermark bits, and hence less uncertainties happen during watermark extraction, resulting in smaller r.

Table 1. Data compression rate. Each table cell shows ρ together with r in bracket.

File size (in pixel)	The length of watermark (bits)			
	100	500	1000	2000
128× 128	87.5%(0.1)	87.36%(23.3)	86.73%(125.7)	84.42%(505)
256× 256	93.75%(0.1)	93.73%(8.3)	93.67%(52.4)	93.34%(268.2)
512× 512	96.88%(0.1)	96.87%(2.7)	96.86%(17.8)	96.83%(104.7)
1024× 1024	98.43%(0)	98.43%(0.9)	98.42%(7.9)	98.41%(32.4)
2048× 2048	99.22%(0)	99.22%(0.2)	99.21%(2.5)	99.21%(9.5)

7 Related Work

Deduplication on encrypted data. To ensure the encrypted data remains deduplicable, a viable solution would be allowing each data owner generates the same encryption key for identical data. Douceur et al. [12]



(a) PoW proof generation (b) PoW proof verification

(c) File compression (d) Watermark extraction

Fig. 4. Computational cost of various components in Client-side Deduplication

proposed convergent encryption (CE), in which the data owner derives the encryption key from the hash value of the message being encrypted. CE, however, suffers from the offline brute-force attack. To mitigate this attack, Bellare et al. proposed DupLESS [9], in which they introduced a trusted key server to further sign the CE key using its private key to enhance randomness of the CE key, and limited the number of requests for keys in a fixed time interval. Bellare et al. [8] formalized encryption schemes to a new cryptographic primitive, called Message-Locked Encryption (MLE) and presented complete security analysis. Liu et al. [11] removed the the trusted key server by utilizing password-based authenticated key exchange protocol. Li et al. [29] and Lei et al. [24] further addressed the re-encryption problem for secure server-side deduplication and client-side deduplication, respectively. Once the encryption key is leaked, the encrypted data in a deduplication-based storage system will need to re-encrypted, but a challenge is how to re-encrypt the data efficiently. Using various techniques like all-or-nothing (AONT) transform [29] and delegated re-encryption [24], the re-encryption efficiency can be improved.

Proofs of ownership (PoWs) and cloud storage security. A PoW [21] protocol is a vital component for a secure client-side deduplication system, which can prevent an adversary from claiming ownership over a file without really possessing it. The first PoW scheme is based on Merkle-tree [21]. Pietro et al. [30] introduced a more efficient and secure PoW design, by asking the cloud server to pre-compute responses for a number of pre-computed PoW challenges. The two schemes assume the cloud server is fully trusted, but the client is untrusted. In parallel with PoWs, another direction of cloud storage security is to allow a client to verify whether the cloud server honestly stores the outsourced data, in which the client is the verifier while the cloud server is the prover. Typical protocols includes Provable Data Possession (PDP) [26, 31, 32]

and Proofs of Retrievability (PoR) [33]. In this line of research, the client is assumed to be trusted while the server is untrusted, and new challenges include, the client is usually equipped with limited computation and storage resources, and hence needs to remain lightweight (e.g., the protocol should only require constant client storage, constant client computation).

8 Conclusion

In this work, we identify a fundamental conflict in deduplication-based cloud storage systems that, CSPs want to remove duplicate data (i.e., deduplication) to save storage space, while data owners want to protect ownership of their outsourced multimedia data by embedding watermarks which may hinder deduplication. We design a deduplication-friendly watermarking (DEW) scheme that can resolve the aforementioned conflict. A salient feature of our design is that, it does not require any interaction among data owners as well as any trusted third party. Security analysis and experimental evaluation show that DEW is secure at the cost of a modest additional overhead.

References

1. "Amazon simple storage service," 2019. `http://aws.amazon.com/cn/s3/`.
2. "Icloud," 2019. `https://www.icloud.com/`.
3. "Microsoft azure," 2019. `http://www.windowsazure.cn/?fb=002`.
4. "Cisco global cloud index: Forecast and methodology, 2016 2021 white paper," 2018. `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html`.
5. D. T. Meyer and W. J. Bolosky, "A study of practical deduplication.," *ACM Transactions on Storage*, vol. 7, no. 4, pp. 1–1, 2012.
6. F. Rashid and A. Miri, "Deduplication practices for multimedia data in the cloud," in *Guide to Big Data Applications*, pp. 245–271, Springer, 2018.
7. A. Z. Tirkel, G. Rankin, R. Van Schyndel, W. Ho, N. Mee, and C. F. Osborne, "Electronic watermark," *Digital Image Computing, Technology and Applications (DICTA'93)*, pp. 666–673, 1993.
8. M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 296–312, Springer, 2013.
9. M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: Server-aided encryption for deduplicated storage," in *USENIX Conference on Security*, pp. 179–194, 2013.
10. A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
11. J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 874–885, 2015.
12. J. R. Douceur, A. Adya, W. J. Bolosky, S. Dan, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *International Conference on Distributed Computing Systems*, pp. 617–624, 2002.
13. "Known attacks towards convergent encryption," 2013. `https://tahoe\-lafs.org/hacktahoelafs/drew\_perttula.html`.
14. B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pp. 31–42, ACM, 2010.
15. B. Chen and R. Curtmola, "Towards self-repairing replication-based storage systems using untrusted clouds," in *Proceedings of the third ACM conference on Data and application security and privacy*, pp. 377–388, ACM, 2013.
16. B. Chen, A. K. Ammula, and R. Curtmola, "Towards server-side repair for erasure coding-based distributed storage systems," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 281–288, ACM, 2015.
17. B. Chen and R. Curtmola, "Remote data integrity checking with server-side repair," *Journal of Computer Security*, vol. 25, no. 6, pp. 537–584, 2017.
18. B. Chen, "New directions for remote data integrity checking of cloud storage," 2014.
19. S. Quinlan and S. Dorward, "Venti: A new approach to archival storage.," in *FAST*, vol. 2, pp. 89–101, 2002.

20. "Dropbox," 2019. https://www.dropbox.com/.
21. S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems.," in *ACM Conference on Computer and Communications Security*, pp. 491–500, ACM, 2011.
22. I. J. Cox, M. L. Miller, J. A. Bloom, and C. Honsinger, *Digital watermarking*, vol. 53. Springer, 2002.
23. D. Gordon, *Discrete Logarithm Problem*. Boston, MA: Springer US, 2011.
24. L. Lei, Q. Cai, B. Chen, and J. Lin, "Towards efficient re-encryption for secure client-side deduplication in public clouds," in *International Conference on Information and Communications Security*, pp. 71–84, Springer, 2016.
25. H. Shacham and B. Waters, "Compact proofs of retrievability," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 90–107, Springer, 2008.
26. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 598–609, Acm, 2007.
27. D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.
28. "Pairing based cryptographic library," 2019. https://crypto.stanford.edu/pbc/.
29. J. Li, C. Qin, P. P. C. Lee, and J. Li, "Rekeying for encrypted deduplication storage," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016.
30. R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pp. 81–82, ACM, 2012.
31. C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 4, p. 15, 2015.
32. B. Chen and R. Curtmola, "Robust dynamic provable data possession," in *2012 32nd International Conference on Distributed Computing Systems Workshops*, pp. 515–525, IEEE, 2012.
33. A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 584–597, Acm, 2007.