Fuzzy Deduplication Scheme Supporting Pre-verification of Label Consistency

Zehui Tang[1], Shengke Zeng[1], Tao Li[2,3], Shuai Cheng[1] and Haoyu Zheng[1]

[1] School of Computer and Software Engineering, Xihua University, Chengdu 610039, China
[2] State Key Laboratory of Public Big Data, College of Computer Science and Technology, Guizhou University, Guiyang 550025, China
[3] School of Computer Science, Qufu Normal University, Rizhao 276825, China.
`Corresponding:zengshengke@gmail.com`

Abstract

Efficiently and securely removing encrypted redundant data with cross-user in the cloud is challenging. Convergent Encryption (CE) is difficult to resist dictionary attacks for its deterministic tag. Server-aided mechanism is against such attacks while it may exist collusion. Focus on multimedia data, this paper proposes an efficient and secure fuzzy deduplication system without any additional servers. We also propose a notion of pre-verification of label consistency to compensate for the irreparable post-verification loss. Compared with other fuzzy deduplication schemes, our work has apparent advantages in deduplication efficiency and security based on a natural data set.

Contents

1 Introduction

People use big data to describe and define the massive data generated in the information age. This data enables applications such as video, audio, images, logs, health records, social network interactions, scientific data, etc [20]. IBM has pointed out that the 2.5EB of data created daily is also 90% of the data generated in the past two years [12]. In order to solve the problem of duplicate data redundancy and reduce expensive storage waste, deduplication technology was born, including the detection of near-duplicate images [10], the elimination of similar or nearly identical audio and video, and other methods [12, 14]. The same or similar files encrypted by different clients are uploaded as completely independent ciphertexts, so basic or naive encryption applications can severely hinder near-duplicate data deduplication. This conflicts with deduplication because comparing the ciphertexts encrypted by users with different private keys is difficult. To address this challenge, some deduplication mechanisms have been proposed, such as Convergent Encryption (CE) [4], Message Lock Encryption (MLE) [2] and its variants [18].

Li *et al.'s* [7] proposed the first secure client-side similar image deduplication scheme for cloud storage to efficiently reduce data redundancy in memory. However, this scheme relies on a trusted third party, and the key sharing among groups does not apply to public cloud storage platforms with a large number of users (such as Huawei Cloud). Chen *et al.'s* [3] also proposed a similar secure image deduplication scheme based on group key sharing, although this scheme can provide some security against external adversary attacks (such as server collusion attacks). However, at the cost of compromising the privacy of group members, using such a scheme makes it difficult to defend against social engineering attacks. For systems with multi-user and cross-domain interaction, this data deduplication mechanism is embarrassing. Jiang *et al.'s* [6] proposed a combination of FuzzyMLE and FuzzyPoW so that the cloud server can safely deduplicate encrypted multimedia data within a certain distance (i.e. Hamming distance) while reducing data overhead from client to server over the network. However, FuzzyMLE is based on the auxiliary server to avoid a guessing attack in which these two servers may collude for each other to leak data, and FuzzyPoW can indeed verify whether there is similar ciphertext between clients, still, its shortcoming is that in the case that FuzzyPoW fails to pass. The server cannot distinguish which client provided incorrect information, and cannot play a tracking role. Our scheme uses a zero-knowledge proof label consistency pre-verification method to solve the problem of FuzzyPoW.

Liu *et al.'s* [9] proposed a single server data deletion scheme for the first time. This scheme relies on the strong collision nature of the short hash to resist the dictionary attack of the storage server and calls similar users to recover the key by uploading users with the help of the cloud storage server. In addition, it is very regrettable that the secure deduplication scheme of Liu *et al.'s* is exact deduplication, which only applies to the exact same files. Takeshia*et al.'s* [13] aim to solve almost the same image deduplication problem, but their scheme cannot resist the brute force attack by repeated queries from the server and the client. The deduplication of fuzzy data based on a single server still has a long way to go.

These deduplication schemes up to date only achieve post-verification of label consistency (verify through the deduplication phase or the download phase). Although they can handle the tag-matching problem, it can not make up for the loss after the integrity compromise. Therefore, we propose a notion of pre-verification to check the label consistency in advance, which effectively solving the irreversible loss problem in post-verification.

In general, we construct a fuzzy deduplication for the similar multimedia data with single server to be against various attacks. Our contribution mainly includes the following 4 items, and the comparison of the related deduplication schemes is listed in Table 1.

- We propose a fuzzy deduplication strategy for approximate data which fits multimedia data. Our scheme does not depend on any additional servers to resist the brute-force attacks. Therefore, it is not necessary to assume the server is honest.
- We propose a notion of pre-verification to avoid the difficulty of tracking malicious users for schemes which are post-verification. As a result, our work compensates for the irreparable loss of post-verification.
- We consider the underlying collusion attacks between the clients and the cloud server and make use of a variable-length short hash technology to handle it.
- Our experimental results show that our scheme achieves high deduplication of fuzzy data by comparing the similarity of tags. With a threshold of 1, our deduplication rate is 20.8% higher than that of Jiang *et al.'s* [6]

Test Group	SS	FD	CR	BFAR	RAR	TC	CG
Jiang *et al.'s* [6]	$\times$	$\surd$	$\times$	$\surd$	$\surd$	$\times$	$\surd$
chen *et al.'s* [3]	$\surd$	$\times$	$\times$	$\surd$	$\times$	$\times$	$\times$
Takeshita *et al.'s* [13]	$\surd$	$\surd$	$\surd$	$\times$	$\surd$	$\times$	$\surd$
Our'scheme	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

Table 1: Comparison of related programss

$'\surd' =$ Satisfied $'\times' =$ Unsatisfied SS = Single Server
FD = Fuzzy Deduplication CR = Collusion Resistance BFAR = Brute-Force Attacks Resistance
RAR = Replay Attacks Resistance TC = Tag Consistency CG = Cross Group

2 Preliminaries

In this section, we summarize the Hamming distance, perceptual hash [8], hash collision [9], and zero-knowledge proof, which constitute the primary supporting knowledge of our system.

2.1 Hamming Distance and Threshold

Hamming Distance. Hamming distance is a common distance measurement method, which is usually used to compare the distance between two characters. Let $\mathtt{X} = (\mathtt{x_1}, \mathtt{x_2}, ..., \mathtt{x_n})$ and $\mathtt{Y} = (\mathtt{y_1}, \mathtt{y_2}, ..., \mathtt{y_n})$, then the Hamming distance between $\mathtt{X}$ and $\mathtt{Y}$ is as shown in the equation (1):

$$\mathtt{H}(\mathtt{X}, \mathtt{Y}) = \sum_{\mathtt{i}=0}^{\mathtt{n}} \mathtt{d}(\mathtt{x_i}, \mathtt{y_i}) \tag{1}$$

Here, $\mathtt{d}(\mathtt{x_i}, \mathtt{y_i})$ means that when the characters in each corresponding position of `X` and `Y` are the same, they are equal to 0, and vice versa, they are equal to 1. For binary coded numbers, the Hamming distance can be calculated by `XOR` operation, as shown in equation (2):

$$\mathtt{H}(\mathtt{X}, \mathtt{Y}) = \sum_{\mathtt{i}=0}^{\mathtt{n}} \mathtt{d}(\mathtt{x_i} \oplus \mathtt{y_i}) \tag{2}$$

Threshold. The threshold has a direct impact on the deduplication rate of the deduplication scheme. We set `D` to represent the threshold between Hamming distances to judge the similarity of data, as shown below:

1)if $\mathtt{d}(\mathtt{X},\mathtt{Y}) \leq \mathtt{D} \Rightarrow$ `X and Y are recognized as similar strings.`
2)if $\mathtt{d}(\mathtt{X},\mathtt{Y}) = 0 \Rightarrow$ `X and Y are recognized as identical strings.`
3)if $\mathtt{d}(\mathtt{X},\mathtt{Y}) > \mathtt{D} \Rightarrow$ `X and Y are recognized as dissimilar strings.`

2.2 Perceptual Hashing

Perceptual Hashing. The Perceptual Hash function can be used to determine whether the original data is similar. Perceptual Hash describes a type of hash value that can be compared, which is a digital signature calculated based on multimedia content and features. Standard Perceptual Hashing algorithms [17] are mainly divided into **ahash** , **phash** and **dhash**. Zauner *et al.'s* pointed out that although the speed of **phash** is slightly slower than that of **ashah** and **dhash**, the recognition effect is the best. Therefore, in order to achieve more effective image feature matching, many deduplication systems select **phash** as the feature vector of massive image data. A common **phash** processes images as follows:

1. Size reduction: To avoid the impact of image size on deduplication, we uniformly scale the image to $\mathtt{N} \times \mathtt{N}$ (high frequency gives details, low frequency gives structure).

2. Adjust colour: Simplify the colour of the image and convert it to grayscale using the following equation (3). Red (R), Green (G), Blue (B).

$$\mathtt{Gray} = \mathtt{R} \times 0.299 + \mathtt{G} \times 0.587 + \mathtt{B} \times 0.114 \tag{3}$$

3. Discrete cosine transform (`DCT`): `DCT` is a special Fourier transform that transforms a picture from the pixel domain to the frequency domain.

4. Calculate `DCT` average value: calculate the average value of 64 reserved low frequencies.

5. **Phash** calculation: compare each `DCT` value with the average value. equation (4) e.g.,

$$\begin{aligned} &\mathtt{if}\ (\mathtt{DCT'value} \geq \mathtt{Avg}) \longmapsto \mathtt{Output}\quad 1 \\ &\mathtt{if}\ (\mathtt{DCT'value} < \mathtt{Avg}) \longmapsto \mathtt{Output}\quad 0 \end{aligned} \tag{4}$$

6. Hamming distance: Perform the XOR operation on the **phash** of the file to determine the number of different characters in the corresponding position of the string.

2.3 Hash Collisions

The hash function is an irreversible mapping from message space to image space and compresses a any input length into a fixed output length. The hash function accepts a string $\mathtt{X} \in \{0,1\}^*$ as input and outputs a string $\mathtt{H(x)} \in \{0,1\}^{\mathtt{n}}$, where n is the length. Hash functions with longer outputs are less collision-resistant, while hash functions with shorter outputs are more collision-resistant.

We use a short hash to avoid guessing attacks and improve efficiency. We obtain `phash` from the original data and select the fingerprints with odd (or even) serial numbers in `phash` to reorganize into a new short `phash`. The assembly process is shown in Figure 1.

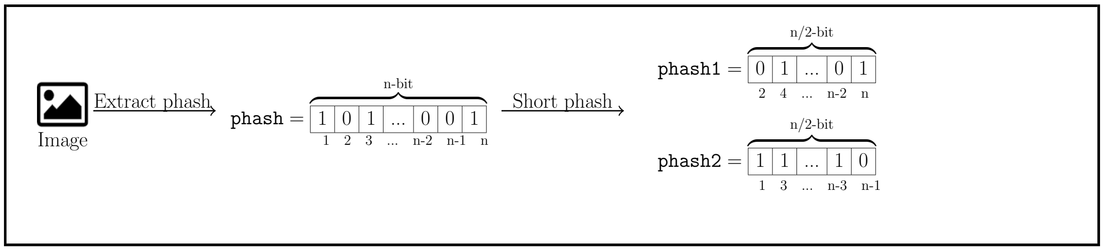


Figure 1: A hash of length n constructs a short hash

2.4 Zero-knowledge Proof

Zero-knowledge system is a cryptographic protocol. One side of the protocol is called the prover (PR), and the other side of the protocol is the verifier (VE). In our paper, we use the zero-knowledge proof of graph isomorphism to propose a verification method for the cloud server to check whether the client's tags are consistent or not, which realizes the pre-verification and avoids the defect of irreparable losses caused by post-verification. In the proof of label consistency, we need to prove that the image matches with the corresponding `phash`, therefore we only need to prove that image (I) and `phash` (P) satisfy the relationship of $\mathtt{I} = \varphi\mathtt{P}$, where φ is the mapping relationship. The proof process is shown in Figure 2.

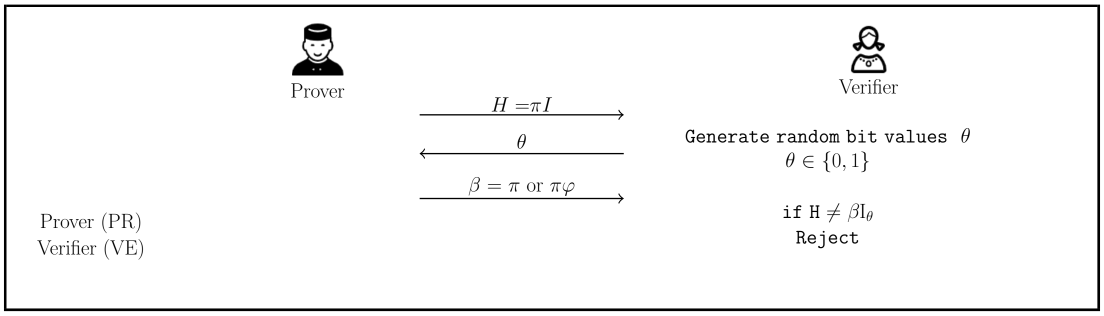

Prover
Verifier
$H = \pi I$
θ
$\beta = \pi$ or $\pi\varphi$
Generate random bit values θ
$\theta \in \{0,1\}$
if $H \neq \beta I_\theta$
Reject
Prover (PR)
Verifier (VE)

Figure 2: Zero-knowledge Proofs for Label Consistency

1. PR randomly generates a permutation π, calculates the graph $\mathtt{H} = \pi\mathtt{I}$, and then sends H to VE.

2. VE generates a bit value $\theta \in \{0,1\}$ and sends θ to PR.

3. PR reacts differently according to θ. As shown in the equation (5)

$$\begin{aligned} \texttt{if } (\theta = 1) &\Rightarrow \texttt{PR} \quad \texttt{sends} \quad \beta = \pi \\ \texttt{if } (\theta = 0) &\Rightarrow \texttt{PR} \quad \texttt{sends} \quad \beta = \pi\varphi \end{aligned} \tag{5}$$

4. VE calculation. As shown in the equation (6)

$$\begin{aligned} \texttt{if } (\theta = 0) &\Rightarrow \beta \texttt{I}_\theta = \pi\varphi\texttt{P} = \pi\texttt{I} = \texttt{H} \\ \texttt{if } (\theta = 1) &\Rightarrow \beta \texttt{I}_\theta = \pi\texttt{I} = \texttt{H} \end{aligned} \tag{6}$$

By using the above method, it is realized that the cloud server verifies whether the image provided by the client matches with phash.

3 Models and Design Goals

3.1 System Model

Unlike the server-aided schemes [6,13], our scheme does not rely on any additional servers (single server only) to solve the brute-force guessing attacks. We adopt a client-side deduplication strategy, in which the files are unnecessary to be uploaded if the duplicates are checked. Therefore, our scheme comprises three kinds of roles: an uploading client, a cloud server, and parallel client $\texttt{PC}_\texttt{i} = \{\texttt{PC}_1, \texttt{PC}_2, ..., \texttt{PC}_\texttt{n}\}$, where n is the number of clients that have stored data similar to the uploading file. The system framework is shown in Figure 3, and each entity is described as follows:

- Uploading Client (C): The uploading client is a user who needs to outsource its data to a cloud server. The stored files should be encrypted for the data privacy. In order to save storage and bandwidth, C would check the existence of the uploading file on the server(by sending the file's fingerprint). Duplicate files are not uploaded.
- Cloud Server (CS): The cloud server stores the client's private data. In order to save storage costs, it needs to deduplicate multimedia data such as pictures and videos. The cloud server would strictly enforce instructions but may also try to guess client privacy (honest but curious).
- Parallel Client ($\texttt{PC}_\texttt{i}$): The parallel clients have stored data on CS. When C may hold similar or identical images, $\texttt{PC}_\texttt{i}$ assists CS in completing the data ownership verification of C. C is defined as a new $\texttt{PC}_\texttt{i}$ member if it passes, and a key for decrypting data is given.

3.2 Threat Model

In our scheme, none of the three entities is absolutely trustworthy, and we analyze the hazards of each entity according to the characteristics of each entity, as shown below:

A malicious uploading client C attempts to launch a data ownership spoofing attack using the file's fingerprint. In addition, it may carry out a forgery attack, where C uses the correct label to repeatedly check but uploads the forged ciphertext (inconsistent with the label) to damage the integrity of other client files.

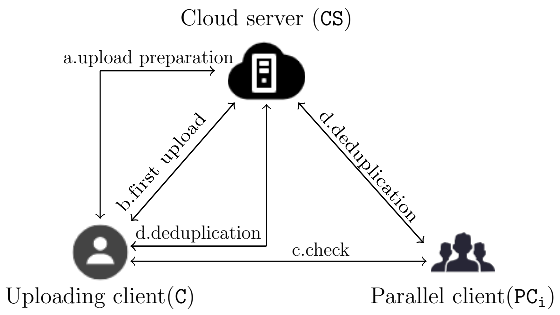


Figure 3: System model

An honest-but-curious cloud service CS faithfully executes storage instructions and deduplication instructions, however it may also try to steal and understand the underlying content of stored ciphertexts. We also consider that CS may collude with C or PC_i to guess the storage content, thereby stealing user data information.

A malicious parallel client PC_i falsely reports information in the hash compared with C resulting in duplicate data not being deduplicated, resulting in a waste of CS storage space. PC_i also may conspire with CS to steal C's private information.

The probability of pairwise conspiracy among the three entities is low, but it does not mean that it will not happen [15, 16], and it is difficult for ordinary users to detect whether their privacy has been leaked (such as CS can guess the stored information without making it public). The security of the above system needs to be improved.

3.3 Design Goals

We design the following security goals by analyzing the threat model in Section 3.2.

- Data privacy: Even if the data is transmitted in an unsafe channel and stolen by the adversary, the adversary will never know the plaintext of the data without the decryption key.

- Resistance to brute-force attacks: Even if the adversary obtains all the data plaintext, it cannot guess the encrypted data stored on CS through the specific tag data.

- Anti-collusion attack: Even if two entities cooperate in exchanging communication parameters and other information, other valid private data information cannot be obtained.

- Tag consistency: Tag inconsistency will have a serious negative impact on subsequent deduplication and download work. Therefore, we need to strictly check whether the C uploaded data matches the tag.

- Track the adversary: Data security cannot be absolutely guaranteed in the security system, but there must be a way to track the adversary after being attacked.

- Efficiency: The main goal of our scheme is to efficiently detect and delete duplicate data while ensuring user privacy and security. We strive to achieve maximum efficiency in deduplication while maintaining low overhead.

4 Proposed Scheme

We present our scheme in this section. It includes 4 main stages in total: Upload Preparation, First Upload, Deduplication and Download. We illustrate the description of the required symbols in table 2 below.

Notation	Description
$\mathtt{ID}$	the identity of $\mathtt{C}$
$\mathtt{pk_c}$	the public key of $\mathtt{C}$
$\mathtt{sk_c}$	the private key of $\mathtt{C}$
$\mathtt{k_c}$	Symmetric encryption key
$\mathtt{ID_i}$	the identity of $\mathtt{PC_i}$
$\mathtt{pk_{pci}}$	the public key of $\mathtt{PC_i}$
$\mathtt{sk_{pci}}$	the private key of $\mathtt{PC_i}$
$\mathtt{pk_{ci}}$	the public key of $\mathtt{CS}$
$\mathtt{sk_{ci}}$	the private key of $\mathtt{CS}$
$\mathtt{ph_c}$	the image fingerprint from $\mathtt{C}$
$\mathtt{ph_{ci}}$	the image fingerprint from $\mathtt{PC_i}$
$\mathtt{d}(\cdot)$	the Hamming distance calculation operation
$\mathtt{D}$	the threshold in deduplication
$\mathtt{url}$	the storage address of C
$\mathtt{OutPut}(\cdot)$	short hash constructor
$\mathtt{Enc}(\cdot)$	public-key encryption algorithm
$\mathtt{E}(\cdot)$	symmetric encryption algorithm

Table 2: Notations used in the proposed scheme

4.1 Upload Preparation

As shown in Figure 4, the client must follow the steps below to determine whether its files need to be deduplicated.

Client:

1. $\mathtt{CS}$ needs to check whether the $\mathtt{I}$ and $\mathtt{P}$ match, as shown in Figure 2. The verification is passed, and the subsequent process is carried out.
2. $\mathtt{C}$ needs to extract $\mathtt{phash}$ from the image and convert the $\mathtt{phash}$ into $\mathtt{phash_i}$, as shown in Figure 1 (For the convenience of $\mathtt{CS}$ to calculate similar fingerprints, $\mathtt{C}$ should select $\mathtt{phash_1}$ or $\mathtt{phash_2}$).
3. $\mathtt{C}$ generates the public key $\mathtt{pk_c}$, secret key $\mathtt{sk_c}$ and $\mathtt{id}$, obtains the public key $\mathtt{pk_{ci}}$ of the cloud storage server and uses $\mathtt{pk_{ci}}$ to encrypt $\mathtt{phash_i}$.
4. $\mathtt{C}$ sends $\mathtt{M_1}$ to $\mathtt{CS}$: $\mathtt{M_1} = \mathtt{Enc}_{\mathtt{pk_{ci}}}(\mathtt{id}, \mathtt{phash_1})$.

Cloud Service:

1. Generate a public key $\mathtt{pk_{ci}}$ and a secret key $\mathtt{sk_{ci}}$.
2. Accept $\mathtt{M_1}$ and decrypt $\mathtt{M_1}$ with $\mathtt{sk_{ci}}$.

3. Perform a deduplication check. Confirm whether the uploaded image is an approximate duplicate file, and determine whether to perform further deduplication operations.
   - if $\mathtt{d} > \mathtt{D}$ Execute First Upload .
   - if $\mathtt{d} \leq \mathtt{D}$ Execute Deduplication.

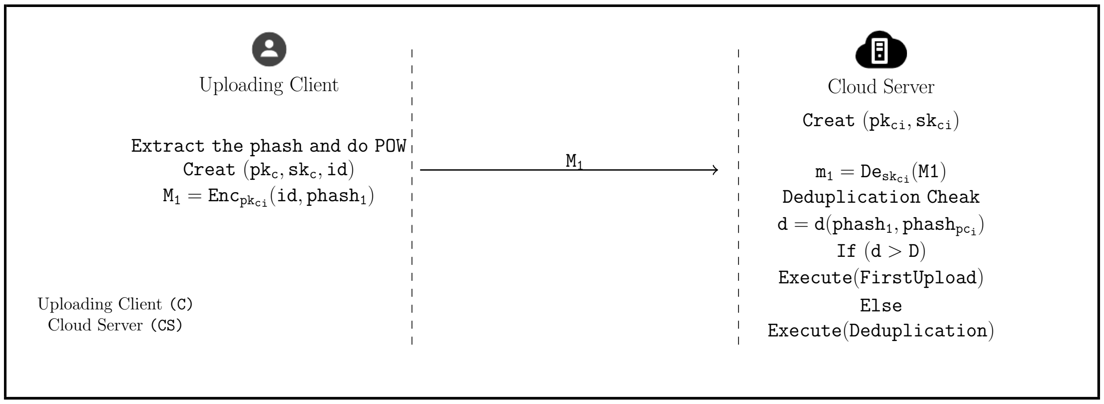

Uploading Client
Cloud Server
Creat $(\mathtt{pk_{ci}}, \mathtt{sk_{ci}})$
Extract the phash and do POW
Creat $(\mathtt{pk_c}, \mathtt{sk_c}, \mathtt{id})$
$\mathtt{M_1} = \mathtt{Enc_{pk_{ci}}}(\mathtt{id}, \mathtt{phash_1})$
$\mathtt{M_1}$
$\mathtt{m_1} = \mathtt{De_{sk_{ci}}}(\mathtt{M1})$
Deduplication Cheak
$\mathtt{d} = \mathtt{d}(\mathtt{phash_1}, \mathtt{phash_{pc_i}})$
If $(\mathtt{d} > \mathtt{D})$
Execute(FirstUpload)
Else
Execute(Deduplication)
Uploading Client (C)
Cloud Server (CS)

Figure 4: Upload Preparation

4.2 First Upload

If there are no duplicate files in CS, then C is asked to upload the image after the duplicate data deletion check, as shown in Figure 5. The specific steps are as follows:

1. CS creates a public resource locator URL to store $\mathtt{C}'$s encrypted file as $\mathtt{URL} = \mathtt{Enc_{pk_c}}(\mathtt{url})$.

2. C accepts URL and decrypts URL with $\mathtt{sk_c}$.

3. C encrypts image $(\mathtt{img})$ with $\mathtt{k_c}$ and sends $\mathtt{I_c}$ to CS as $\mathtt{I_c} = \mathtt{E_{k_c}}\ (\mathtt{img})$.

4. CS accepts $\mathtt{I_c}$ and records $\{\mathtt{I_c}, \mathtt{phash_i}, \mathtt{id}\}$.

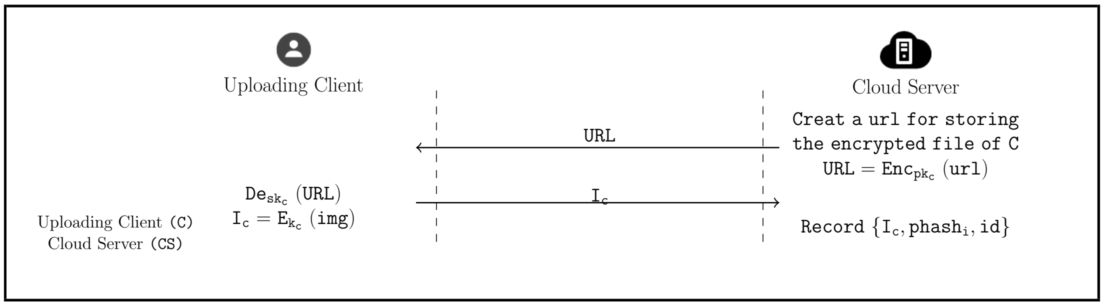

Uploading Client
Cloud Server
Creat a url for storing the encrypted file of C
$\mathtt{URL} = \mathtt{Enc_{pk_c}}\ (\mathtt{url})$
URL
$\mathtt{De_{sk_c}}\ (\mathtt{URL})$
$\mathtt{I_c} = \mathtt{E_{k_c}}\ (\mathtt{img})$
$\mathtt{I_c}$
Record $\{\mathtt{I_c}, \mathtt{phash_i}, \mathtt{id}\}$
Uploading Client (C)
Cloud Server (CS)

Figure 5: First Upload

4.3 Deduplication

When $\mathtt{C}$ enters the deduplication phase, it means that $\mathtt{CS}$ may have stored an image similar to the uploaded image, therefore it needs to interact with $\mathtt{CS}$ and $\mathtt{PC_i}$ at this stage. Specifically, $\mathtt{C}$ needs to perform a new round of $\mathtt{phash}$ value comparison with $\mathtt{PC_i}$, $\mathtt{CS}$ judges whether deduplication is required based on the comparison between $\mathtt{C}$ and $\mathtt{PC_i}$. Finally, $\mathtt{C}$ will be confirmed as a parallel client or upload files according to the above judgment. The process is shown in Figure 6, and the detailed steps can be described as follows.

Step 1: $\mathtt{CS}$ determines that $\mathtt{C}$ may have similar or identical images with $\mathtt{PC_i}$ through Hamming distance.

After $\mathtt{CS}$ verifies the labelling consistency provided by $\mathtt{C}$, accepts the short hash of $\mathtt{C}$ to match with the short hash provided by $\mathtt{PC_i}$. After narrowing down the range of target customers who may have similar or identical images, we need to make a comparison between $\mathtt{C}$ and $\mathtt{PC_i}$ improved short hash lengths, allowing for more precise comparisons. Therefore, we need a new round of communication.

1. $\mathtt{CS}$ needs find $\mathtt{PC_i}$ from the cloud database.

2. $\mathtt{CS}$ needs build an $\mathtt{n'}$ bit $\mathtt{OutPut}$ function, $\mathtt{n'}$ is the length of output $\mathtt{phash}$

$$\mathtt{OutPut}(\mathtt{n'}, \mathtt{phash}) \qquad \mathtt{n'} \in (\mathtt{n}/2, \mathtt{n}].$$

3. $\mathtt{CS}$ encrypts $\mathtt{ID_{ci}}$ and $\mathtt{D'}$ with $\mathtt{pk_c}$, $\mathtt{D'}$ is the threshold for the current round of agreement to determine whether the image is similar

$$\mathtt{ID_i} = \mathtt{Enc}_{\mathtt{pk_c}}\ (\mathtt{ID_{ci}}||\mathtt{D'}).$$

4. $\mathtt{CS}$ encrypts $\mathtt{ID_c}$ and $\mathtt{D'}$ with $\mathtt{pk_{pci}}$

$$\mathtt{ID_c} = \mathtt{Enc}_{\mathtt{pk_{pci}}}\ (\mathtt{ID_c}||\mathtt{D'}).$$

5. $\mathtt{CS}$ sends $\mathtt{ID_i}$ to $\mathtt{C}$, and sends $\mathtt{ID_c}$ to $\mathtt{PC_i}$.

Step 2: Communication between $\mathtt{C}$ and $\mathtt{PC_i}$to determine whether the file of $\mathtt{C}$ is similar to the file of $\mathtt{PC_i}$.

$\mathtt{C}$ and $\mathtt{PC_i}$ use the $\mathtt{output}$ function to reconstruct their $\mathtt{phash}$, and pass the $\mathtt{phash}$ to each other use the Hamming distance and $\mathtt{D'}$ to further judge whether the two files are similar. The process is as follows.

1. $\mathtt{C}$ uses $\mathtt{output}$ function to regenerate $\mathtt{ph_c}'$

$$\mathtt{ph_c}' = \mathtt{OutPut}(\mathtt{n'}, \mathtt{ph_c}).$$

2. $\mathtt{C}$ encrypts $\mathtt{ph_c}'$ with $\mathtt{pk_{pci}}$ and sends $\mathtt{Enc}_{\mathtt{pk_{pci}}}\ (\mathtt{ph_c}')$ to $\mathtt{PC_i}$.

3. $\mathtt{PC_i}$ uses $\mathtt{output}$ function to regenerate $\mathtt{ph_{ci}}'$

$$\mathtt{ph_{ci}}' = \mathtt{OutPut}(\mathtt{n'}, \mathtt{ph_{ci}}).$$

4. $\mathtt{PC_i}$ encrypts $\mathtt{ph_{ci}}'$ with $\mathtt{pk_c}$ and sends $\mathtt{Enc}_{\mathtt{pk_c}}\ (\mathtt{ph_{ci}}')$ to $\mathtt{C}$.

5. $\mathtt{C}$ and $\mathtt{PC_i}$ calculate the Hamming distance $\mathtt{d}(*)$ between $\mathtt{ph_{ci}}'$ and $\mathtt{ph_c}'$, and judge whether the two files are similar, as shown in Algorithm 1.

Algorithm 1 Algorithm for Judging File Similarity (First Round)

```
$d_1 = d(ph_c', ph_{ci}')$
if $d_1 > D'$ then
    Execute First Upload;
else
    $ph_c'' = OutPut(n, ph_c)$
    $Enc_{pk_{pci}}(ph_c'')$.
end
```

We emphasize the following 2 points:

- To avoid the collusion attack of CS and C (or $\mathtt{PC_i}$), the length of the phash constructed in this round should be less than n. If the length of the current round of phash reaches n, CS can obtain the complete phash of the image and use the dictionary attack to obtain the clear text information of the image.
- The precondition for CS to perform the First Upload operation is that the feedbacks from $\mathtt{PC_i}$ and C to CS are both Execute First Upload. If the feedback from P and C is inconsistent, CS is confident to suspect that there exists a malicious adversary.

Step 3: This communication will finally determine whether the image of C has a similar image in CS.

C and $\mathtt{PC_i}$ continue to exchange phash and then calculate the Hamming distance, respectively. If the Hamming distance is lower than the threshold, we can think that C and $\mathtt{PC_i}$ have similar or identical files. The process is as follows.

1. C uses *output* function to regenerate $\mathtt{ph_c}''$
$$\mathtt{ph_c}'' = \mathtt{OutPut}(\mathtt{n}, \mathtt{ph_c}).$$
2. C encrypts $\mathtt{ph_c}''$ with $\mathtt{pk_{pci}}$ and sends $\mathtt{Enc_{pk_{pci}}}(\mathtt{ph_c}'')$ to $\mathtt{PC_i}$.
3. $\mathtt{PC_i}$ uses *output* function to regenerate $\mathtt{ph_{ci}}''$
$$\mathtt{ph_{ci}}'' = \mathtt{OutPut}(\mathtt{n}, \mathtt{ph_{ci}}).$$
4. $\mathtt{PC_i}$ encrypts $\mathtt{ph_{ci}}''$ with $\mathtt{pk_c}$ and sends $\mathtt{Enc_{pk_c}}(\mathtt{ph_{ci}}'')$ to C.
5. C and $\mathtt{PC_i}$ calculate the Hamming distance $\mathtt{d}(*)$ between $\mathtt{ph_{ci}}''$ and $\mathtt{ph_c}''$, and judge whether the two files are simila, as shown in Algorithm 2.

Algorithm 2 Algorithm for Judging File Similarity (Second Round)

```
$d_1' = d(ph_c'', ph_{ci}'')$
if $d_1' > D$ then
    Execute First Upload;
else
    Execute Deduplication.
end
```

6. If CS executes Deduplication, adds C into $\mathtt{PC_i}$.
7. C and $\mathtt{PC_i}$ have passed the mutual authentication, and $\mathtt{PC_i}$ will encrypt the $\mathtt{k_c}$ and URL of the decrypted image with $\mathtt{pk_c}$ and send it to C.

We emphasize the following two points:

- The length of phash used for comparison here should be n. The longer the length of the phash, the more accurate the comparison and the higher the deduplication rate.
- CS performs a deduplication operation on the image of C, which means that S and $\mathtt{PC_i}$ have similar or identical images. C will join $\mathtt{PC_i}$ and play a role in assisting certification in the deduplication work of CS in the future.

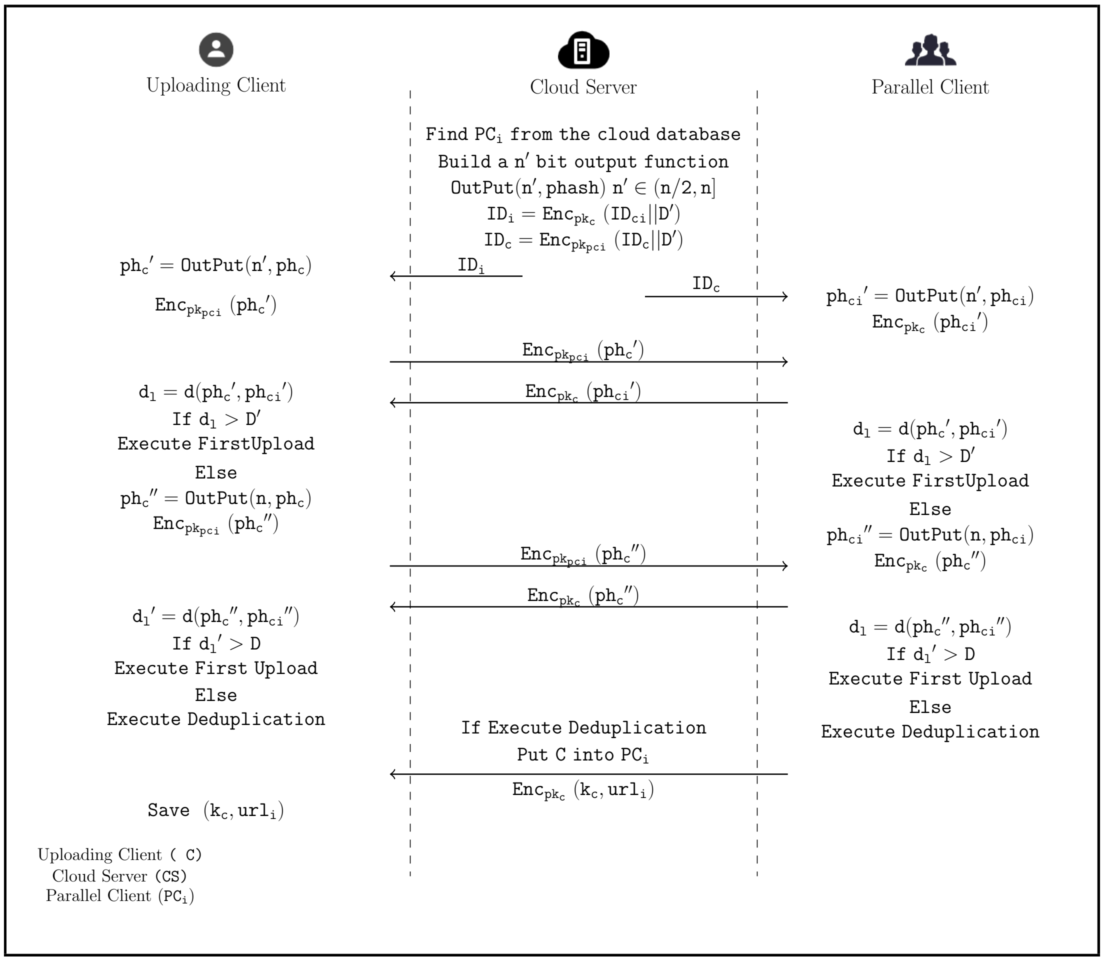


Figure 6: Deduplication

4.4 Download

Only client C has previously stored an image in CS can enter the download phase, andC has saved $(\mathtt{k_c}, \mathtt{url_i})$. Therefore, C can directly find the image storage location through URL and use

$\mathtt{k_c}$ to decrypt the image. At this point, the whole download phase completes. The process is described as follows.

1. $\mathtt{C}$ finds the information $\mathtt{k_c}, \mathtt{url}$ related to the restored image $\mathtt{img}$.
2. $\mathtt{C}$ finds the storage address through $\mathtt{url}$, and decrypts the $\mathtt{img}$ through $\mathtt{k_c}$. The decryption formula is: $\mathtt{img} = \mathtt{De}_{\mathtt{k_c}}(\mathtt{E}_{\mathtt{k_c}}(\mathtt{img}))$.

5 Security Analysis

In this section, we extend the description to data confidentiality, collusion resistance, brute-force attack resistance, label consistency, etc., and make a detailed analysis of the security peoperties of our scheme.

5.1 Data Confidentiality

In our single server-based deduplication scheme, we particularly emphasize security. Data outsourced to the cloud server for $\mathtt{C}$ cannot be known by $\mathtt{CS}$ and $\mathtt{PC_i}$, which do not own similar files. Obviously, in our scheme, $\mathtt{CS}$ stores the $\{\mathtt{ID_i}, \mathtt{Enc}_{\mathtt{pk_c}}, \mathtt{phash_i}\}$ from $\mathtt{C}$, and $\mathtt{CS}$ does not have the key to decrypt $\mathtt{img}$, so it does not have the conditions to decrypt $\mathtt{img}$. Our encryption method is public key encryption, and the $\mathtt{pk_c}$ and $\mathtt{sk_c}$ designed in this method are mathematically related. We have disclosed $\mathtt{pk_c}$ to the public, but relying on $\mathtt{pk_c}$ alone cannot deduce $\mathtt{sk_c}$ (or we cannot crack $\mathtt{sk_c}$ in polynomial time). The only thing that can leak $\mathtt{sk_{pci}}$ in the whole scheme is $\mathtt{PC_i}$, but once $\mathtt{PC_i}$ leaks the $\mathtt{sk_{pci}}$, it is equivalent to leaking its own private information, so this situation will not happen. In our scheme, only legal users can know the $\mathtt{url}$, which effectively prevents adversaries from obtaining confidential data. For the security discussion of stealing user data through $\mathtt{phash}$, we will discuss it in detail in Section 5.4.

5.2 Brute-Force Attacks Resistance

In our scheme, brute force attacks are mainly manifested in two aspects: First, a brute force attack obtains $\mathtt{phash}$ of $\mathtt{img}$. Once the adversary obtains the original $\mathtt{phash}$, it may pose a threat to $\mathtt{C}$ or $\mathtt{PC_i}$. $\mathtt{CS}$ would record the $\mathtt{phash_i}$ of $\mathtt{C}$, but the $\mathtt{phash_i}$ is a short hash (with strong collision properties). Therefore, even if $\mathtt{CS}$ has $\mathtt{img}$ with all plaintext states, it cannot guess the plaintext information of $\mathtt{C}$ through $\mathtt{phash_i}$. The adversary can also try to construct a $\mathtt{phash}$ with length $\mathtt{n}$, but the probability of successful collision is extremely low. Brute-Force Attacks Resistance is mainly divided into offline guessing and online guessing.

1. Offline guessing: In our scheme, the $\mathtt{phash}$ length of the original data is 64 bits of binary $\{0,1\}^{\mathtt{n}}(\mathtt{n} = 64)$. The probability that a malicious adversary wants to guess the phase directly offline and use brute force to crack it is $\mathtt{P}$. Each bit has two choices, 64 bits in total. Combined with $\mathtt{P} = (1/2)^{64}$, we think that the probability of successful guessing is too low, so we can resist offline guessing attacks.

$$\mathrm{P} = \underbrace{1/2 * 1/2 * 1 * ... * 1/2 * 1/2 * 1/2}_{(64)\ bit} = (1/2)^{64}$$

2. Online guessing: The adversary guesses during the mutual verification process between $\mathtt{PC}$ and $\mathtt{C}$. The $\mathtt{phash}_{\mathtt{i}}^{'}$ in the verification process is 26 bits longer than the $\mathtt{phash_i}$

stored in the $\mathtt{CS}$, so the probability of correct guessing is $\mathtt{U}$. In particular, the number of communications between $\mathtt{PC}$ and $\mathtt{C}$ is limited, so the adversary cannot always guess. Combined with $\mathtt{U}$, we believe that the probability of successful online guessing is extremely low, so it can resist online guessing attacks.

$$\mathrm{U} = \underbrace{1/2 * 1/2 * 1 * ... * 1/2 * 1/2 * 1/2}_{(26)\ bit} = (1/2)^{26}$$

To sum up, our scheme can resist Brute-Force Attacks. In addition, brute force to obtain the decrypted key has been discussed in Section 5.1.

5.3 Resistance to Replay Attacks

Replay attacks refer to an attacker sending a packet that has been received by a destination host to deceive the system to obtain other users' private information. In our scheme, assume that the adversary obtains $\mathtt{phash_i}$ in phase First Upload. The adversary can use $\mathtt{phash_i}$ and $\mathtt{CS}$ to communicate and enter the Deduplication stage smoothly, but in the Deduplication stage, $\mathtt{PC_i}$ and the adversary will perform mutual verification and calculate the Hamming distance $\mathtt{d(hash_i,\ hash_{ci})}$. In this process, the length of $\mathtt{phash_i}$ satisfies ($\mathtt{n}' > \mathtt{n/2}$). There is no doubt that ($\mathtt{d} > \mathtt{D}$), so $\mathtt{PC_i}$ will not pass the verification of the opponent.

5.4 Collusion Resistance

As shown in Figure 3, we assume that any two of the three entities can collude, even if collusion affects the entity's reputation. Collusion can be divided into the following cases.

1. $\mathtt{CS}$ and $\mathtt{C}$ conspire to defraud $\mathtt{PC_i}$ of private information. $\mathtt{CS}$ and $\mathtt{C}$ conspire means going directly to the Deduplication stage. At this stage, the $\mathtt{PC_i}$ will use its own $\mathtt{ph_{ci}}$ and the $\mathtt{phash_c}$ of $\mathtt{C}$ to calculate the Hamming distance. $\mathtt{C}$ does not have the ability to provide $\mathtt{phash_c'}$ after changing the length, so the check will not pass. We also limit the number of online communications of each entity to more effectively prevent information disclosure.

2. $\mathtt{CS}$ and $\mathtt{PC_i}$ conspire to obtain the information of $\mathtt{C}$. When $\mathtt{CS}$ deliberately lets $\mathtt{C}$ enter the Deduplication stage so that $\mathtt{PC_i}$ has the opportunity to steal $\mathtt{phash_C}$, then decrypt the $\mathtt{img}$ of $\mathtt{C}$ through the dictionary. In our scheme, $\mathtt{C}$ and $\mathtt{CS}$ are verified simultaneously for the first time. $\mathtt{PC_i}$ does not provide the correct $\mathtt{phash_{ci}}$. $\mathtt{C}$ executes the Hamming distance calculation, and through ($\mathtt{d} > \mathtt{D}$), it will directly cancel the next communication with $\mathtt{PC_i}$, record the $\mathtt{ID}$ of $\mathtt{PC_i}$, and regard this $\mathtt{PC_i}$ as a malicious client.

3. $\mathtt{C}$ and $\mathtt{PC_i}$ conspire. $\mathtt{CS}$ will perform label consistency verification on $\mathtt{C}$ in the Upload Preparation phase. To pass the verification, $\mathtt{C}$ can only use the data of the accomplice $\mathtt{PC_i}$, which means that $\mathtt{PC_i}$ leaks its data to $\mathtt{C}$. Therefore, we analyze that there will be no collusion between $\mathtt{C}$ and $\mathtt{PC_i}$.

5.5 Tag Consistency

In our scheme, there are two places where tags can be verified as follows.

1. The first verification method is shown in Figure 2. During this process, C proves the consistency of img and `phash` to `CS` and also avoids the leakage of his own information. This method of verifying label consistency in advance makes up for the defect of post-verification (the loss is irreversible).

2. The second is in the mutual verification between `C` and `PC`. If the feedback results from `C` and `PC` to `CS` are inconsistent, it means that there is a problem with one party. At this time, `CS` only needs to check `C` and another $\mathtt{PC}'_{\mathtt{i}}$, and according to the result of the second check, it can be determined who has a problem with `C` or $\mathtt{PC}_{\mathtt{i}}$ (locating the adversary). `CS` checks for messages $\mathtt{M}_1$ from `C`, messages $\mathtt{M}_2$ from $\mathtt{PC}_{\mathtt{i}}$ and messages $\mathtt{M}_3$ from $\mathtt{PC}'_{\mathtt{i}}$, judgment criteria are as follows.

$$\begin{array}{c} \mathtt{if}\ (\mathtt{M}_1\ ==\ \mathtt{M}_3)\quad \mathtt{PC}_{\mathtt{i}}\ \mathtt{lied} \\ \mathtt{Else}\quad \mathtt{C}\ \mathtt{lied} \end{array}$$

6 Experiment and Performance Analysis

6.1 Simulation Settings

We implement public-key encryption `RSA` [11, 19] to encrypt image fingerprints and symmetric encryption `AES` to encrypt media data in our simulation. `RSA` security depends on the difficulty of factorization of large integers. Our system is implemented on Windows 10 with a 2.30GH i5-8300H and 8G RAM. The programming language we use is Python 3.90. Using the dataset Microsoft Common Objects in Context(MS COCO) dataset.

The effective deletion of approximate duplicate data directly determines the future development of our system. We selected 21844 non-repetitive images in the MS COCO dataset. These images vary in pixel size and spatial size. In addition, to calculate the deduplication rate $\varrho_{\mathtt{p}}$ as shown in equation (7). We add noise to these 21844 images, and the image is used as an approximate image atfer adding noise.

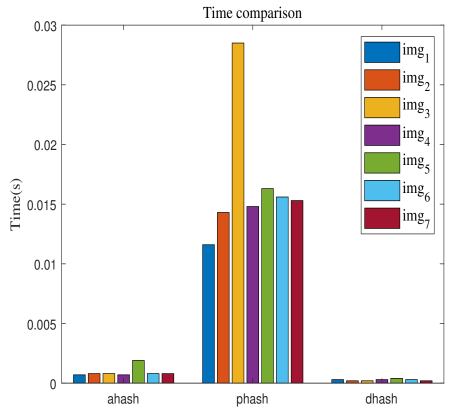
$$\varrho_{\mathtt{p}} = 1 - \varsigma/\tau \tag{7}$$

Here, ς is the simulated data that was blocked from being uploaded during the experiment, and τ is the total number of uploaded files. During the experiment, we extracted the `phash` length of 64 bits from `img`.

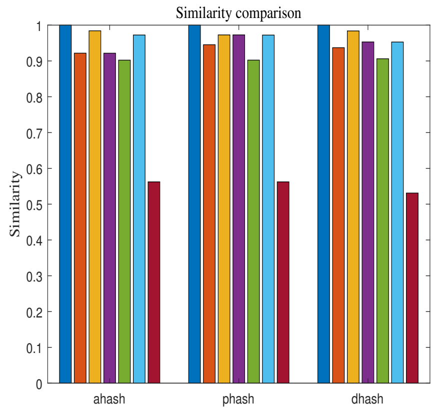
6.2 Simulation Results

Time spent on image feature extraction by perceptual hash (`ahash`, `phash`, `dhash`) **[1, 5]. Effect of different perceptual hash feature value extraction methods on similarity**.

We selected an image $\mathtt{img}_1$ with 219KB and 640×480 pixels, then the image was modified by parameters: Brightening $\mathtt{img}_2$, Enlarge $\mathtt{img}_3$, Add contrast $\mathtt{img}_4$, Sharpen $\mathtt{img}_5$, Add colour $\mathtt{img}_6$, and Rotate 45 degrees $\mathtt{img}_7$. We used $\mathtt{ahash} - \mathtt{phash} - \mathtt{dhash}$ to extract the feature values of these images and recorded the time. The result is shown in Figure 7 (a). We use (`ahash`, `phash`, `dhash`) methods to extract the eigenvalues of $(\mathtt{img}_1 - \mathtt{img}_7)$, respectively. Compare the similarity between the feature values of the modified image and the original image ($\mathtt{Threshold\ D} \leq 5$). The result is shown in Figure 7 (b).



(a) Time cost



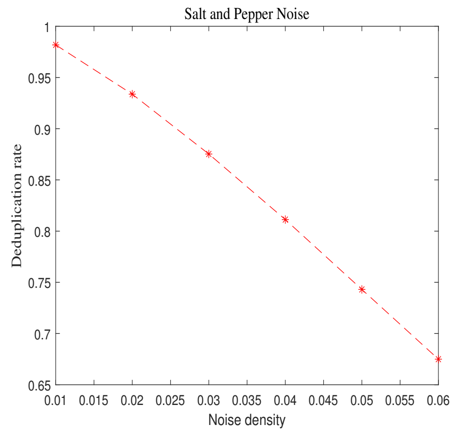
(b) Similarity comparison

Figure 7: *Comparison of* `ahash`, `phash and dhash`

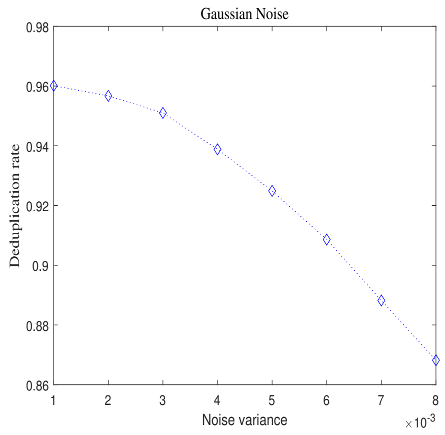
Figure 7, shows that although `phash` takes longer to extract feature values than `ahash dhash`, the similarity detection of `phash` is better than `ahash dhash`. The time spent by `phash` does not exceed 0.03 seconds, which is acceptable to the client, so we use phash to extract feature values later in the deduplication rate comparison.

Effects of various distortions on data deduplication.

There is no denying that any data may lose some messages during transmission. These distorted data often do not affect the use but cause data redundancy. Considering this situation, we tested the effects of salt and pepper noise, Gaussian noise, Poisson noise, and Motion blur on data deduplication in simulation experiments. We sampled 21844 images on the MS COCO dataset for testing and recorded the deduplication rate. Figure 8 9 records the impact of Salt and pepper noise, Gaussian noise, Poisson noise and Motion blur on the deduplication rate $\varrho_{\mathtt{p}}$.

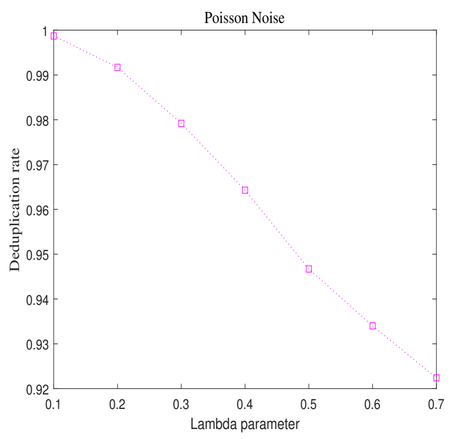


(a) Salt and pepper noise

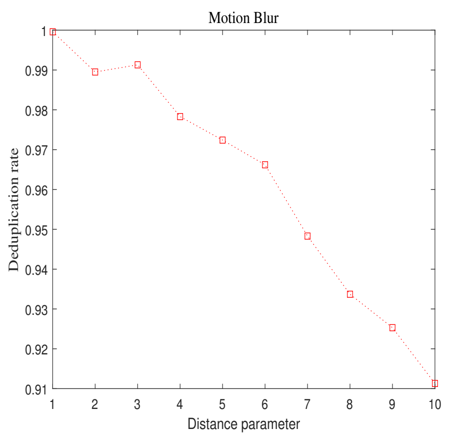


(b) Gaussian noise

Figure 8: Salt and pepper noise and Gaussian noise compared with the original image



(c) Poisson noise



(d) Motion blur

Figure 9: Poisson noise and Motion blur compared with the original image

Relationship of thresholds to deduplication rate

In this simulation experiment, we modify the pixels of the whole image with various noises to generate a series of redundant copies so as to explore an appropriate threshold value. We set various thresholds ($\mathtt{D} \in [\mathtt{1}, \mathtt{6}]$) and designed the original `phash` as 64 bits. From Figure 10 (a) that as the threshold increases, the deduplication rate continues to rise. When D=5, the redundant copy with noise can achieve a deletion effect of more than 95%.

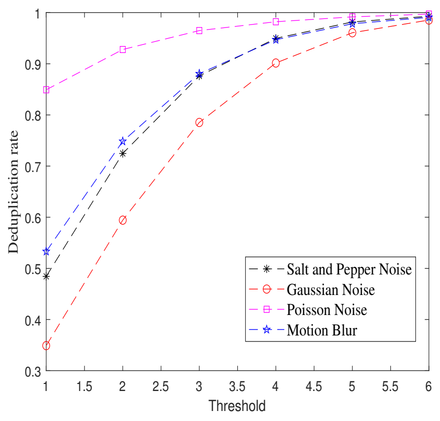
Compared with the fuzzy deduplication scheme of Jiang *et al.'s* [6].

Both we and Jiang *et al.'s* [6] used the threshold value ($\mathtt{D} = \mathtt{1}$, $\mathtt{D} = \mathtt{2}$, $\mathtt{D} = \mathtt{3}$) as variables to compare the deduplication rate ϱ_p. We used 21844 images in the MS COCO dataset, and Jiang took 22317 images modified by 1.5% in the CC_WEB_VIDEO dataset. We use motion blur (distance = 4) to modify the image by 1.5%. With a similar number of datasets and equal thresholds on both sides, our deduplication rate is significantly higher than that of Jiang *et al.'s* [6]. The experimental results are shown in Figure 10 (b).

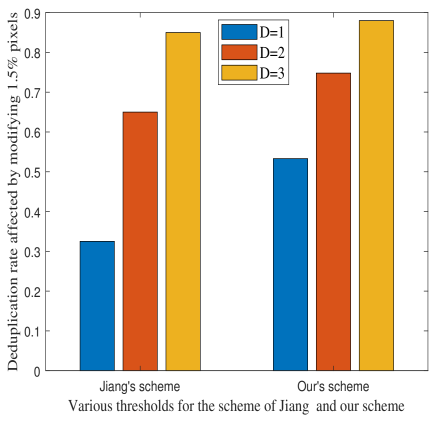
7 Conclusion and Discussion

This paper designs a fuzzy deduplication system for similar multimedia data without any additional servers. The brute-force guessing attacks and collusion attacks are considered in this work. In addition, we propose a notion of pre-verification to realize the pre-judgment of label consistency and therefore it eliminates the irreparable loss of post-verification. In the upload preparation stage, the client constructs the phash of the original data into a short hash of n/2 length and then performs the first data deduplication judgment. Therefore it reduces the range of fuzzy data comparison and preliminarily determining whether the cloud platform retains similar copies. In the deduplication phase, the upload client and the parallel client check for each other. The verification passes the parallel client, and the uploading client shares the file decryption key.

We have made a systematic analysis of the security of the scheme and designed several experiments to show the deduplication effect of our scheme in real application scenarios. The result shows that the scheme can effectively guarantee the confidentiality of data and complete



(a) Relationship of thresholds to deduplication rate



(b) ϱ_p affected by modifying 1.5% pixels

Figure 10: Influence of threshold selection and interference factor on deduplication ratio (ϱ_p)

the deletion of redundant data. On the other hand, how to select a short hash with appropriate length is a challenge. If the length is too short, the deduplication rate will decrease, but if the length is too long, it is difficult to resist the guessing attack. We will leave the exploitation of these open problems in our future work.

Acknowledgement

This work was sponsored by Natural Science Foundation of Sichuan, China (2023NS-FSC1400) and Chengdu Science and Technology Program (2021-YF08-00151-GX).

References

[1] Mehran Arefkhani and Mohsen Soryani. Malware clustering using image processing hashes. In *2015 9th Iranian Conference on Machine Vision and Image Processing (MVIP)*, pages 214–218. IEEE, 2015.

[2] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 296–312. Springer, 2013.

[3] Lu Chen, Feng Xiang, and Zhixin Sun. Image deduplication based on hashing and clustering in cloud storage. *KSII Transactions on Internet & Information Systems*, 15(4), 2021.

[4] John R Douceur, Atul Adya, William J Bolosky, P Simon, and Marvin Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings 22nd international conference on distributed computing systems*, pages 617–624. IEEE, 2002.

[5] Mengjuan Fei, Zhaojie Ju, Xiantong Zhen, and Jing Li. Real-time visual tracking based on improved perceptual hashing. *Multimedia Tools and Applications*, 76:4617–4634, 2017.

[6] Tao Jiang, Xu Yuan, Yuan Chen, Ke Cheng, Liangmin Wang, Xiaofeng Chen, and Jianfeng Ma. Fuzzydedup: Secure fuzzy deduplication for cloud storage. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[7] Xuan Li, Jin Li, and Faliang Huang. A secure cloud storage system supporting privacy-preserving fuzzy deduplication. *Soft Computing*, 20:1437–1448, 2016.

[8] Yuenan Li, Dongdong Wang, and Jingru Wang. Perceptual image hash function via associative memory-based self-correcting. *Electronics Letters*, 54(4):208–210, 2018.

[9] Jian Liu, Nadarajah Asokan, and Benny Pinkas. Secure deduplication of encrypted data without additional independent servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 874–885, 2015.

[10] Thomas Pönitz and Julian Stöttinger. Efficient and robust near-duplicate detection in large and growing image data-sets. In *Proceedings of the 18th ACM international conference on multimedia*, pages 1517–1518, 2010.

[11] Abhishek Sachdeva. A study of encryption algorithms aes, des and rsa for security. *Global Journal of Computer Science and Technology*, 13(E15):32–40, 2013.

[12] Dominik Schnitzer, Arthur Flexer, and Gerhard Widmer. A fast audio similarity retrieval method for millions of music tracks. *Multimedia Tools and Applications*, 58:23–40, 2012.

[13] Jonathan Takeshita, Ryan Karl, and Taeho Jung. Secure single-server nearly-identical image deduplication. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2020.

[14] Xiao Wu, Alexander G Hauptmann, and Chong-Wah Ngo. Practical elimination of near-duplicates from web video search. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 218–227, 2007.

[15] Xue Yang, Rongxing Lu, Kim Kwang Raymond Choo, Fan Yin, and Xiaohu Tang. Achieving efficient and privacy-preserving cross-domain big data deduplication in cloud. *IEEE transactions on big data*, 8(1):73–84, 2017.

[16] Xue Yang, Rongxing Lu, Jun Shao, Xiaohu Tang, and Ali A Ghorbani. Achieving efficient secure deduplication with user-defined access control in cloud. *IEEE Transactions on Dependable and Secure Computing*, 19(1):591–606, 2020.

[17] Christoph Zauner. Implementation and benchmarking of perceptual image hash functions. 2010.

[18] Yuan Zhang, Chunxiang Xu, Hongwei Li, Kan Yang, Jianying Zhou, and Xiaodong Lin. Healthdep: An efficient and secure deduplication scheme for cloud-assisted ehealth systems. *IEEE Transactions on Industrial Informatics*, 14(9):4101–4112, 2018.

[19] Xin Zhou and Xiaofei Tang. Research and implementation of rsa algorithm for encryption and decryption. In *Proceedings of 2011 6th international forum on strategic technology*, volume 2, pages 1118–1121. IEEE, 2011.

[20] Paul Zikopoulos and Chris Eaton. *Understanding big data: Analytics for enterprise class hadoop and streaming data.* McGraw-Hill Osborne Media, 2011.