TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 01/18 pp1–12
DOI: 10.26599/TST.2021.9010071
Volume 28, Number 1, February 2023

Threat Model and Defense Scheme for Side-Channel Attacks in Client-Side Deduplication

Guanxiong Ha, Hang Chen, Chunfu Jia*, and Mingyue Li

Abstract: In cloud storage, client-side deduplication is widely used to reduce storage and communication costs. In client-side deduplication, if the cloud server detects that the user's outsourced data have been stored, then clients will not need to reupload the data. However, the information on whether data need to be uploaded can be used as a side-channel, which can consequently be exploited by adversaries to compromise data privacy. In this paper, we propose a new threat model against side-channel attacks. Different from existing schemes, the adversary could learn the approximate ratio of stored chunks to unstored chunks in outsourced files, and this ratio will affect the probability that the adversary compromises the data privacy through side-channel attacks. Under this threat model, we design two defense schemes to minimize privacy leakage, both of which design interaction protocols between clients and the server during deduplication checks to reduce the probability that the adversary compromises data privacy. We analyze the security of our schemes, and evaluate their performances based on a real-world dataset. Compared with existing schemes, our schemes can better mitigate data privacy leakage and have a slightly lower communication cost.

Key words: cloud storage; deduplication; side-channel; privacy

1 Introduction

The rapid growth of data volume has required cloud service providers to use the data deduplication to reduce storage and communication costs[1–3]. After the deduplication, the cloud server could identify data redundancy and only store a single copy of user data. Based on the deduplication location, deduplication can be classified as a server or client side deduplication[2]. In the server-side deduplication[4,5], clients always upload data to the cloud server. After receiving the uploaded data, the cloud server performs deduplication to save storage space. In the client-side deduplication, clients compute hash values for user data as data tags and send them to the cloud server. After receiving data tags, the cloud server checks whether the data have been stored based on data tags and returns deduplication responses to clients. For example, if the data have been stored, then the deduplication response will be set to 1. Otherwise, it will be set to 0, as shown in Fig. 1. When a client receives a response of 0, it should upload the data; otherwise, it does not need to upload the data. Compared with server-side deduplication, client-side deduplication can reduce storage and communication costs and has been widely used in cloud storage[6]. However, deduplication responses in the client-side deduplication could be exploited by adversaries to launch side-channel attacks[7] to violate data privacy, because the data transmission between clients and the server can be monitored by adversaries and data tags could be used to detect the data existence in the server.

In view of the above problems, this paper makes the following contributions.

• We propose a new threat model against side-channel attacks in the client-side deduplication. Different from existing defense schemes[8,9], our threat

• Guanxiong Ha, Hang Chen, Chunfu Jia, and Mingyue Li are with the College of Cyber Science, Nankai University, Tianjin 300350, China, and also with the Tianjin Key Laboratory of Network and Data Security Technology, Tianjin 300350, China. E-mail: cfjia@nankai.edu.cn.

* To whom correspondence should be addressed.

Manuscript received: 2021-09-18; accepted: 2021-09-29

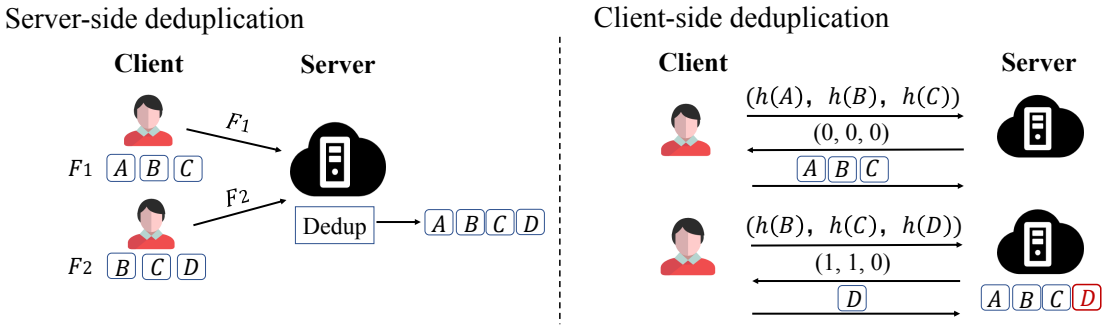


Fig. 1 Deduplication in the cloud storage.

model considers a stronger adversary, which can learn the approximate ratio of stored chunks to unstored chunks in outsourced files. We assume that the adversary could maliciously construct outsourced files with a certain number of stored chunks (uploaded by it before) and unstored chunks (random chunks). Then, it places a specific target chunk in maliciously constructed files and performs side-channel attacks by constantly uploading constructed files with different ratios of stored chunks to unstored chunks. The adversary can observe deduplication responses and data transmissions during the deduplication check, and try to learn the existence of the target chunk.

- Under our threat model, we propose two defense schemes against side-channel attacks, namely basic and enhanced schemes. We argue that the reason why the adversary could launch side-channel attacks is that the deduplication responses leak the information of data existence. Therefore, both our schemes design interaction protocols between clients and the server to disturb the correlation between deduplication responses and data existence.
- We analyze the security for our basic and enhanced schemes and two existing schemes[8,9] under our threat model, and then evaluate the computational and communication overheads in these four schemes based on a real-world dataset. The results of the security analysis and performance evaluation show that our schemes can effectively mitigate data privacy leakage and reduce the communication cost for the system.

2 Related Work

2.1 Data deduplication

Data deduplication is an effective method to save storage overhead for cloud storage systems[2,10,11]. The cloud server can detect redundant data in cross-user uploaded data by deduplication, and only store unique data. Based on the data granularity, deduplication can be divided into file-level or chunk-level deduplication. In the file-level deduplication[11,12], the user file is treated as the basic unit for deduplication. By contrast, in the chunk-level deduplication[5,13–16], clients divide user files into chunks and the chunk is the basic unit. Compared with file-level deduplication, chunk-level deduplication usually has a higher deduplication ratio.

In the chunk-level client-side deduplication, we suppose that the outsourced file is F, the client first splits F into multiple chunks $\{m\}$, where m denotes the data chunk. Then, the client computes the hash values $\{h\}$ for chunks as data tags and sends them to the cloud server. The cloud server uses data tags for deduplication check. If the data tag h_i is not found, then the server will set the deduplication response to 0 and return it to the client. Then, the client needs to upload the chunk m_i and the server stores $(h_i = H(m_i), m_i)$, where $H(\cdot)$ denotes a cryptographic hash function. Otherwise, if h_i has been found in the server, then the client will receive the response of 1 and does not need to upload m_i.

2.2 Side-channel attacks

Harnik et al.[7] found that the client-side deduplication can be used as a side-channel. In cross-user client-side deduplication, the adversary can establish a side-channel using deduplication responses and violate the privacy of user data. For example, adversaries can use the side-channel to launch the following attacks:

- **Identifying the existence of specific files:** Suppose an adversary wants to learn whether a specific file F has been uploaded to the cloud server by other users, it can observe deduplication responses when uploading F. If the server asks the adversary to upload F, then it learns that F is not stored on the server. Otherwise, F has already been uploaded by other users.
- **Establishing a covert channel:** Multiple adversaries can establish a covert channel to

communicate with one another through deduplication responses. For example, two adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ first agree on a specific file F. Then, $\mathcal{A}_1$ can transmit one bit to $\mathcal{A}_2$ by uploading or deleting F. If $\mathcal{A}_2$ uploads F and receives the deduplication response of 0, then it learns that the bit sent by $\mathcal{A}_1$ is 0. Otherwise, the bit is 1.

• **Learning-the-remaining-information (LRI) attack:** In the LRI attack[17], the adversary knows most of the contents of a specific file F, and it tries to learn the remaining unknown information by uploading all possible versions of F to launch side-channel attacks. Suppose that F can be divided into chunks $\{m_1, m_2, \ldots, m_n\}$ and only m_i is unknown to the adversary, then the adversary can launch brute-force attacks if m_i is predictable[18]. If the adversary can determine that m_i is drawn from a chunk dictionary $S_m = \{m_{i1}, m_{i2}, \ldots, m_{il}\}$, then it can repeatedly upload files from a file dictionary $S_F = \{F_{i1}, F_{i2}, \ldots, F_{il}\}$, as shown in Fig. 2. If the adversary finds that m_{ij} does not need to be uploaded when uploading F_{ij}, then it can be sure that $m_i = m_{ij}$.

The LRI attack may occur in many application scenarios. In the real world, many files are presented in a public known standard document template (e.g., medical reports of patients or salary contracts in a company[17]), which means that the adversary could know most of the file contents, except for a few sensitive information. Nevertheless, sensitive information in files can often be predicted by adversaries. For example, results in a medical report for a patient are usually only drawn from a small domain (e.g., a yes or no in a medical test), which are not difficult to be predicted. The salaries for employees in a company may be just in the range of 1000 to 10 000, and a multiple of 500, which is also easy to be predicted[7, 17].

Therefore, the adversary can launch side-channel attacks on many real-world files to violate the privacy of user data, which is a non-negligible threat in cloud storage.

2.3 Existing defense schemes against side-channel attacks

Harnik et al.[7] proposed a defense scheme against side-channel attacks by configuring random thresholds t_F for each file in the server. If the number of uploads for a file F does not reach t_F, then the server will perform the server-side deduplication to protect against side-channel attacks. Only when the number of uploads for a file has already reached t_F can the server perform the client-side deduplication. Lee and Choi[19] found that the probability that the adversary in Ref. [7] learns data information is still non-negligible and proposed a new method to select thresholds for files. Armknecht et al.[20] defined the security for client-side deduplication and proposed a criterion for designing deduplication strategies.

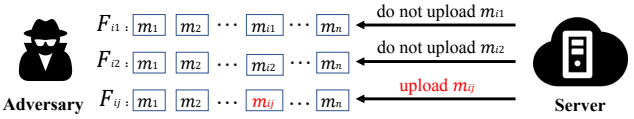


Fig. 2 Learning the remaining information of specific files.

However, for defense schemes using thresholds, the adversary cannot violate data privacy only when the number of uploads for user data is still not reaching the thresholds set by the server. Moreover, it is heuristic to choose proper thresholds, and the choice of thresholds probably makes the system difficult to design[8, 9]. Furthermore, adding thresholds to the client-side deduplication incurs additional communication costs because clients may still need to upload data even though they have already been stored on the server. Some defense schemes deploy a trusted gateway between the server and clients to obfuscate the network traffic. Heen et al.[21] proposed a gateway-based deduplication scheme to reduce the risk of information leakage. The gateway is used to obfuscate the view of the adversary. Shin and Kim[22] designed a differentially private client-side deduplication scheme, which used a gateway to achieve data deduplication and privacy protection. However, the introduction of an extra trusted third party may make schemes less practical.

Pooranian et al.[8] proposed random response (RARE). Yu et al.[9] proposed zero-knowledge deduplication response (ZEUS) and ZEUS$^+$. These schemes all perform the deduplication check on two chunks at once to design the randomized deduplication response. Their threat models assume that the probability that each chunk is stored on the server is quite small. This may not be a very reasonable assumption because the adversary could use random chunks (almost impossible to be stored on the server) and stored chunks (uploaded by the adversary before) to construct malicious files to launch side-channel attacks. For most chunks in these maliciously constructed files, the probability that they are stored on the server can be inferred by the adversary. In other words, the probability that each chunk has already been stored is not always quite small.

3 System Model

This section first introduces the architecture of our

scheme, then describes the proposed new threat model and security definition.

3.1 Architecture

Our schemes consist of two entities: clients and a cloud server.

• **Clients:** To outsource a user file F to the cloud server, the client divides F into fix-sized chunks $\{m\}$ and computes the hash values $\{h\}$ for these chunks as data tags. The client sends data tags to the server to ask for deduplication responses, which determine how the client uploads data.

• **Cloud server:** The cloud server provides data storage services for multiple users and performs cross-user chunk-level client-side deduplication to minimize storage and communication costs. After receiving the data tags uploaded by clients, the cloud server checks whether they were stored before. Then, it sends deduplication responses back to clients.

3.2 Threat model

The adversary of the side-channel attack is usually a malicious client, which tries to learn the existence of a target chunk c in the server using deduplication responses. Some previous schemes[8, 9] supposed that the probability that each chunk is stored in the server is quite small, which means that the adversary is unaware of the existence of all chunks in the outsourced file. However, this may not be a reasonable assumption. We argue that the adversary could learn the existence of most chunks in the outsourced file. So, the probability that a chunk is stored in the server is not always very small, which breaks the assumption in previous threat models.

In our threat model, the adversary could construct some random chunks $M_r = \{m_{r_1}, m_{r_2}, \ldots, m_{r_p}\}$ and stored chunks $M_s = \{m_{s_1}, m_{s_2}, \ldots, m_{s_q}\}$, and place them in an outsourced file F_m along with a target chunk c. Then, the adversary could upload many outsourced files $\{F_{m_1}, F_{m_2}, \ldots, F_{m_n}\}$ with different ratios of random chunks to stored chunks to launch side-channel attacks, as shown in Fig. 3. Evidently, the probabilities that chunks in the outsourced file are stored in the server are not equal. For chunks in M_r, the probabilities are close to 0. By contrast, for chunks in M_s, the probabilities should be 1. The deduplication responses for chunks in M_r and M_s should be 0 and 1, respectively. Thus, the adversary can adjust the number of 0 and 1 in deduplication responses by adjusting the size of M_r and M_s in F_m. The number of 0 and 1 in deduplication responses will affect the probabilistic

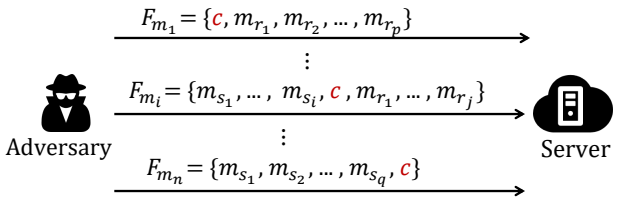


Fig. 3 Side-channel attacks in our threat model.

advantage that adversary $\mathcal{A}$ learns the existence of the target chunk c. We will analyze this point in detail in Section 5.

Our threat model is reasonable in the real world. The adversary can upload some chunks before launching side-channel attacks, and select some of these uploaded chunks as M_s. Moreover, the random chunks M_r are easy to be constructed.

3.3 Security definition

The chunks in outsourced files constructed by the adversary can be divided into the target chunk c and other chunks $M_o = \{m_{o_1}, m_{o_2}, \ldots, m_{o_n}\}$. We use P_o to denote the probability that an arbitrary chunk in M_o is stored in the server. Clearly, P_o is related to the size of M_s and M_r in outsourced files. For example, if the ratio of the size of M_s and M_r is 1:2, then P_o is equal to 1/3. P_c denotes the probability that the target chunk c is stored in the server. For the adversary without any prior knowledge, P_c is equal to 1/2 because it can only randomly guess the existence of c is yes or no[8, 9]. The adversary can receive several deduplication responses from the server by constantly uploading files containing different sizes of M_s and M_r to launch side-channel attacks. Then, it can use these deduplication responses as the prior knowledge to infer the existence of c.

We use $dr = f(c)$ to denote the deduplication responses that the adversary receieves during the deduplication check, where $f(\cdot)$ denotes the deduplication check protocol and c denotes the target chunk. The probabilistic advantage that an adversary $\mathcal{A}$ learns the existence of c can be defined in Eq. (1),

$$Adv(\mathcal{A}) = |P[C|dr] - P_c| \tag{1}$$

where C denotes the case that c is stored in the server. The adversary can use deduplication responses dr as the prior knowledge to try to learn the existence of c. In the server-side deduplication, $P[C|dr]$ is equal to 1/2 and $Adv(\mathcal{A}) = 0$ because dr does not reveal any existence information for chunks. By contrast, in the client-side deduplication, $P[C|dr]$ is equal to 1 or 0 and $Adv(\mathcal{A}) = 1/2$ because dr directly reveals existence

information. Therefore, $Adv(\mathcal{A})$ is in the range of $0 - 1/2$. Our security goal is to minimize the impact of the deduplication responses dr on the existence of chunks. In other words, we want to minimize $Adv(\mathcal{A})$.

4 Proposed Defense Scheme

4.1 Main idea

Some existing schemes[8,9] attempt to defense against side-channel attacks by carefully designing deduplication responses. They use the idea of exclusive-or (XOR) obfuscation to encode deduplication responses. For a chunk m_i in the outsourced file, the client either uploads m_i directly or uploads $m_i \oplus m_i'$, where m_i' is another chunk in the file. There are two situations when a chunk is uploaded. These two different situations can be exploited by adversaries to learn the existence of chunks. Thus, we want to design schemes in which all chunks are uploaded after the XOR operation. Then, it will become difficult for adversaries to use deduplication responses to learn the chunk existence. When the number of stored chunks is not less than that of unstored chunks, we can design a scheme to realize that all chunks are uploaded after being XORed once. Hence, deduplication responses do not reveal any existence information for chunks. When the number of unstored chunks is larger than that of stored chunks, some chunks will be XORed twice, whereas the other chunks are XORed only once. This information may reveal the existence of chunks to some extent. Our solution is to group the deduplication results of stored and unstored chunks into different sets and try to equate the probabilities that the chunk XORed twice belongs to stored chunks and unstored chunks to minimize privacy leakage.

Accordingly, we put forward the basic and enhanced schemes to defense side-channel attacks.

4.2 Basic scheme

Suppose that a client uploads a user file F to the cloud server. It first splits F into serveral chunks $\{m\}$, then calculates their hash values as data tags $\{h\}$, and uploads these tags to the cloud server. The cloud server checks whether uploaded data are stored based on tags. If the tag has already been stored, then the server sets the deduplication result to 1. Otherwise, the deduplication result is set to 0. We use N_s and N_r to denote the numbers of stored and unstored chunks in a file, respectively. P denotes the ratio of N_r to N_s ($P = N_r/N_s$). Our basic scheme will generate deduplication responses in different ways depending on the value of P, which can be divided into the following cases.

$P = 1$.

(1) The server puts tags with the deduplication results of 0 and 1 into two sets to construct $H_0 = \{h_{01}, h_{02}, \ldots, h_{0l}\}$ and $H_1 = \{h_{11}, h_{12}, \ldots, h_{1l}\}$, both of length l.

(2) As shown in Fig. 4, for each h_{0i} in H_0, the server randomly selects a tag h_{1j_i} from H_1 to construct a pair (h_{0i}, h_{1j_i}). The server can get l pairs $S = \{(h_{01}, h_{1j_1}), (h_{02}, h_{1j_2}), \ldots, (h_{0l}, h_{1j_l})\}$.

(3) The server sends S as deduplication responses to the client.

After receiving deduplication responses, the client performs the XOR operation on chunks corresponding to tags in pairs in S and sends the results $\{m_{01} \oplus m_{1j_1}, m_{02} \oplus m_{1j_2}, \ldots, m_{0l} \oplus m_{1j_l}\}$ to the server. Because $\{m_{1j_1}, m_{1j_2}, \ldots, m_{1j_l}\}$ have already been stored, the server can restore $\{m_{01}, m_{02}, \ldots, m_{0l}\}$. In this case, the amount of uploaded chunks is exactly the same as that of the client-side deduplication, both of which are of length l. Therefore, our scheme does not add additional communication costs. Moreover, different from RARE[8], ZEUS, and ZEUS$^+$[9], the uploaded data in our scheme are all the results of the XOR operation between chunks, and a single data chunk will not be uploaded directly. In the view of the client, each chunk is uploaded after being XORed. The client can only observe the times of the XOR operation performed on a chunk. In this case, all chunks are XORed only once. Thus, deduplication responses do not reveal any existence information for chunks.

$P < 1$.

(1) Similar to the previous case, the server gets $H_0 = \{h_{01}, h_{02}, \ldots, h_{0l}\}$ and $H_1 = \{h_{11}, h_{12}, \ldots, h_{1l}, \ldots, h_{1l'}\}$. In this case, the lengths of H_0 and H_1 are not equal ($l < l'$).

(2) The server divides H_1 into two sets $H_1' = \{h_{11}, h_{12}, \ldots, h_{1l}\}$ and $H_1'' = \{h_{1l+1}, h_{1l+2}, \ldots,$

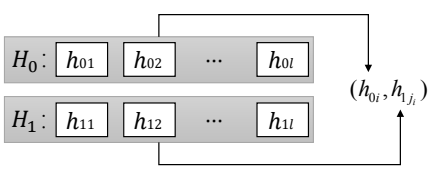


Fig. 4 Basic scheme ($P = 1$).

$h_{1l'}\}$, as shown in Fig. 5.

(3) For each h_{0i} in H_0, the server randomly selects a tag h_{1j_i} from H_1' to construct a pair (h_{0i}, h_{1j_i}). The server can get l pairs $S_0 = \{(h_{01}, h_{1j_1}), (h_{02}, h_{1j_2}), \ldots, (h_{0l}, h_{1j_l})\}$.

(4) The server pairs the tags from H_1'' randomly to construct another set S_1 that contains $(l'-l)/2$ pairs. If $(l'-l)$ is an odd number, then the client uploads a random chunk to let $l = l+1$.

(5) The server sends $S = \{S_0, S_1\}$ as deduplication responses to the client.

The client performs the XOR operation on chunks corresponding to tags in pairs in $S = \{S_0, S_1\}$ and sends the results to the server. In this case, the amount of uploaded chunks is $(l + \lceil (l'-l)/2 \rceil)$, which is more than that of the client-side deduplication. The difference between the two is $\lceil (l'-l)/2 \rceil$. Similar to the previous case, all chunks are all XORed only once. Thus, deduplication responses also do not reveal any existence information for chunks.

$P > 1$.

(1) The server gets $H_1 = \{h_{11}, h_{12}, \ldots, h_{1l}\}$ and $H_0 = \{h_{01}, h_{02}, \ldots, h_{0l}, \ldots, h_{0l'}\}$ $(l < l')$.

(2) The server divides H_0 into two sets $H_0' = \{h_{01}, h_{02}, \ldots, h_{0l}\}$ and $H_0'' = \{h_{0l+1}, h_{0l+2}, \ldots, h_{0l'}\}$, as shown in Fig. 6.

(3) For each h_{1i} in H_1, the server randomly selects a tag h_{0j_i} from H_0' to construct a pair (h_{1i}, h_{0j_i}). The server can get l pairs $S_0 = \{(h_{11}, h_{0j_1}), (h_{12}, h_{0j_2}), \ldots, (h_{1l}, h_{0j_l})\}$.

(4) The server selects $(l'-l)/2$ tags each from H_1 and H_0' to construct H_t of length $(l'-l)$.

(5) For each $h_{0i'}$ in H_0'', the server randomly selects a tag h_{tj_k} from H_t to construct a pair $(h_{0i'}, h_{tj_k})$. Then the server can get $(l'-l)$ pairs $S_1 = \{(h_{0l+1}, h_{tj_1}), (h_{0l+2}, h_{tj_2}), \ldots, (h_{0l'}, h_{tj_{(l'-l)}})\}$.

(6) The server sends $S = \{S_0, S_1\}$ as deduplication responses to the client.

The client performs the XOR operation on chunks corresponding to tags in pairs in $S = \{S_0, S_1\}$, and sends the results $M_{S_0} = \{m_{11} \oplus m_{0j_1}, \ldots, m_{1l} \oplus m_{0j_l}\}$ and $M_{S_1} = \{m_{0l+1} \oplus m_{tj_1}, \ldots, m_{0l'} \oplus m_{tj_{(l'-l)}}\}$ to the server. Because m_{1i} in every pair (m_{1i}, m_{0j_i}) in M_{S_0} has been stored before, the server could restore every m_{0j_i}. Therefore, the server can restore all chunks in M_{S_0}. Similarly, chunks corresponding to tags in H_t can be restored by the server, so chunks corresponding to tags in all pairs in S_1 can also be restored. In other words, all chunks in M_{S_1} also could be restored by the server.

In this case, the number of uploaded chunks is the same as that of the client-side deduplication, which is l'. If the number of 0 in deduplication responses is more than three times that of 1 ($P > 3$), the length of H_t cannot reach that of H_0'', and the server cannot restore all chunks corresponding to tags in H_0''. A straightforward way to solve this problem is to let the client upload all chunks when $P > 3$. However, the adversary may use this information to launch side-channel attacks. For example, the adversary could construct a file F_m with $P = 3$ and then place a target chunk c into F_m to construct F_m'. If it finds that all chunks in F_m' need to be uploaded, then it can learn that $P > 3$ and c have been stored. Therefore, the server needs to set a security parameter ξ. Every time P for an uploaded file is greater than 1; the server needs to select a random value for ξ. The server will let the client upload all chunks when $P > 3-\xi$. The selection for ξ is analyzed in Section 5.1.

However, there is a security vulnerability in the basic scheme. Unlike the above two cases, a chunk may be XORed once or twice when $P > 1$. However, the XOR times of a chunk may be exploited by the adversary to learn the chunk existence. For this vulnerability, we propose a kind of attack in Section 4.3 and an enhanced scheme to protect against this attack in Section 4.4.

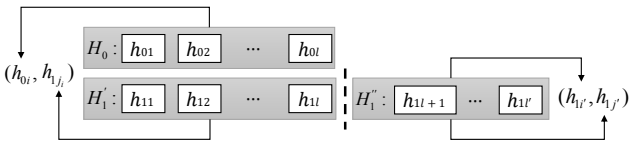


Fig. 5 Basic scheme ($P < 1$).

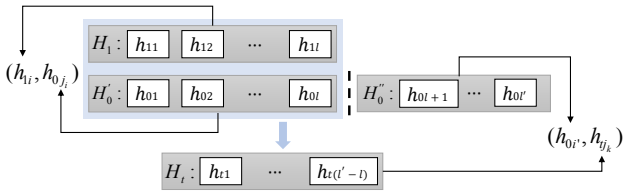


Fig. 6 Basic scheme ($P > 1$).

4.3 Attack for the basic scheme

We present an example of an attack that exploits the vulnerability in the basic scheme. As shown in Fig. 7, we assume that an adversary constructs files $\{F_m\}$ with $P = 1$, $P < 1$, and $P > 1$, and then it places the target chunk c into $\{F_m\}$ to construct $\{F_m'\}$ to upload. Based on the different values of P, the attack can be divided

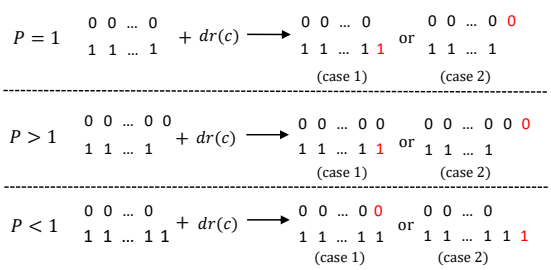


Fig. 7 Attack for the basic scheme (*dr*(*c*) denotes the deduplication result for *c*).

into the following situations:

- $P = 1$. If c has already been stored (case 1 in Fig. 7), then the adversary can learn that each chunk is XORed only once. Otherwise (case 2 in Fig. 7), one chunk is XORed twice.
- $P > 1$ (For simplicity, we let $N_r = N_s + 1$). If c has already been stored, then the adversary can learn that each chunk is XORed only once. Otherwise, two chunks are XORed twice.
- $P < 1$ ($N_s = N_r + 1$). All chunks are XORed once, regardless of whether c has been stored or not.

When P is not less than 1, the adversary can determine whether c is stored by observing the number of chunks XORed twice.

4.4 Enhanced scheme

We argue that the reason for the attack described in Section 4.3 is that there are chunks XORed twice only when $P > 1$ in the basic scheme. We design an enhanced scheme to protect against the attack. Regardless of the value of P, there are chunks XORed twice in the enhanced scheme. As a result, the adversary cannot violate data privacy by observing the number of chunks XORed twice. We describe the enhanced scheme in detail below.

(1) Similar to the basic scheme, the server divides the deduplication results of 0 and 1 for the outsourced file into $H_0 = \{h_{01}, h_{02}, \ldots, h_{0l_0}\}$ and $H_1 = \{h_{11}, h_{12}, \ldots, h_{1l_1}\}$. Let the smaller between l_0 and l_1 be l.

(2) The server sets two ratios P_{min} and P_{max} ($0 < P_{min}, P_{max} < 1$) as security parameters for the outsourced file, and then selects a random probability value $P_e \in [P_{min}, P_{max}]$.

(3) The server selects $(P_e \cdot l)$ tags from H_0 and H_1 to construct H_0' and H_1', respectively. The left tags in H_0 and H_1 construct a new set H_n.

(4) For each tag in H_0', the server randomly selects a tag in H_1' to pair. The server takes these $(P_e \cdot l)$ pairs as the deduplication response S_0, as shown in Fig. 8. Then

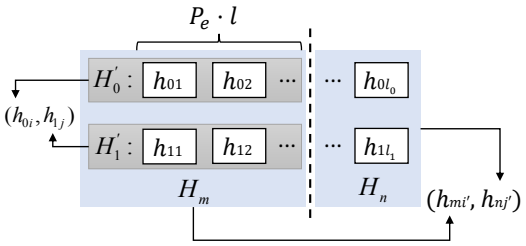


Fig. 8 Enhanced scheme.

the server merges H_0' and H_1' into H_m.

(5) For each tag in H_n, the server randomly selects a tag in H_m to pair. Then, the server takes these pairs as the deduplication response S_1, and sends $S = \{S_0, S_1\}$ to the client.

We use N_{x_2} to denote the number of chunks XORed twice. In the enhanced scheme, N_{x_2} is equal to the size of H_n. In other words, $N_{x_2} = (1 - P_e)(l_0 + l_1) = (1 - P_e) \cdot 2 \cdot l + |N_r - N_s|$. Because P_e is randomly selected each time the client uploads data, we can use it to mitigate the impact of the existence of the target chunk c on N_{x_2}. Thus, the adversary cannot learn the existence of c by observing N_{x_2}. The selection of P_e will affect the probability advantage of the adversary. The detailed security analysis will be given in Section 5.

5 Security Analysis

5.1 Selection of security parameters

5.1.1 Selection of ξ

As described in Section 4.2, the adversary may construct a file F_m and its P is close to 3. Then it places a target chunk c into F_m to launch side-channel attacks. Therefore, ξ should be selected to prevent the adversary from simply adding a single chunk c to F_m to make $P > 3$. In other words, we need to ensure that $(N_r + 1) \leqslant 3 \cdot Ns$. We can use this condition to derive $P = \frac{N_r}{N_s} \leqslant \left(3 - \frac{1}{N_s}\right)$. Therefore, the selection of ξ for a file needs to satisfy that $\xi > \frac{1}{N_s}$. When P for an uploaded file is greater than $3 - \xi$, the server will let the client upload all chunks for this file.

5.1.2 Selection of P_e

In the attack presented in Section 4.3, the adversary adds a target chunk c into files and then observes the change of N_{x_2}. The deduplication response of one chunk has a maximum effect of 2 on the value of N_{x_2} for the whole file, in the case that P for F_m is greater than 1, as shown in Fig. 7. We want N_{x_2} for the outsourced file in the two cases to be nearly equal. Thus, the selection of P_e in the

enhanced scheme needs to satisfy that $|((1-P_{e_1})\cdot 2\cdot l+2)-(1-P_{e_2})\cdot 2\cdot l|$ can be less than or equal to 0, where P_{e_1} and P_{e_2} are the randomly selected values for P_e in the two cases. We can use this condition to derive $(P_{e_1}-P_{e_2})\geqslant\frac{1}{l}$. Therefore, for an uploaded file, the selection of $P_{\min}$ and $P_{\max}$ needs to satisfy that $(P_{\max}-P_{\min})\geqslant\frac{1}{l}$.

5.2 Security analysis for defense schemes

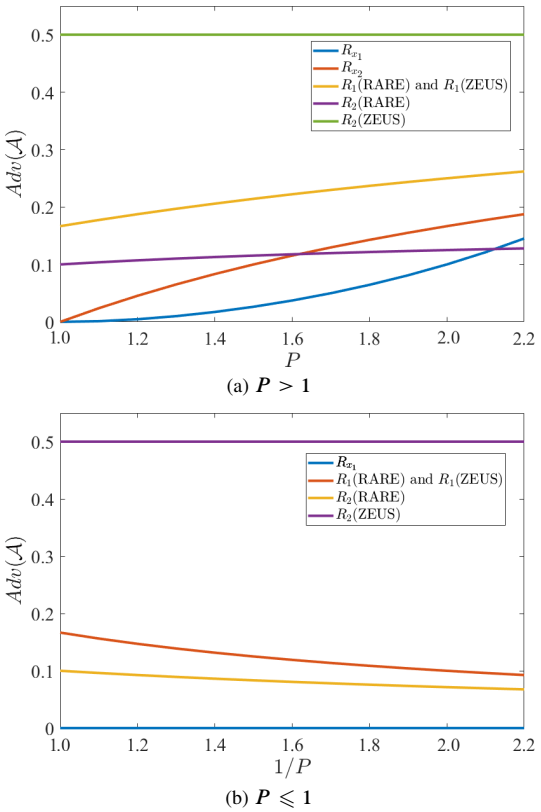
In this section, we use the probabilistic advantage of the adversary described in Section 3.3 to analyze the security for our schemes and two previous schemes[8, 9]. In RARE[8] and ZEUS[9], the adversary can observe two cases when uploading a chunk (described in Section 4.1). If the deduplication response for a chunk is 2, then it is uploaded directly. Otherwise, if the deduplication response is 1, then the chunk is uploaded after being XORed. In our schemes, the adversary can learn that a chunk is XORed once or twice by deduplication responses.

For a target chunk c, we use R to denote the deduplication response for it. R_1 and R_2 denote the events when the deduplication responses for c are 1 and 2 in RARE or ZEUS, respectively. R_{x_1} and R_{x_2} denote the events when c is XORed once and twice in our schemes, respectively. C and $\overline{C}$ denote the events when c is stored and not stored in the server, respectively. Given the deduplication response R, the probability that c is stored can be formulated in Eq. (2).

$$Pr[C|R]=\frac{Pr[R|C]Pr[C]}{Pr[R|C]Pr[C]+Pr[R|\overline{C}]Pr[\overline{C}]} \quad (2)$$

According to our security definition, the probabilistic advantage for adversary $\mathcal{A}$ can be formulated as $Adv(\mathcal{A})=|Pr[C|R]-1/2|$. The security of schemes can be formulated as $\max\{|Pr[C|R_i]-1/2|\}$, where $R_i\in\{R_1,R_2,R_{x_1},R_{x_2}\}$. In other words, the security is evaluated by the deduplication response that has the greatest influence on the probability that a chunk is stored.

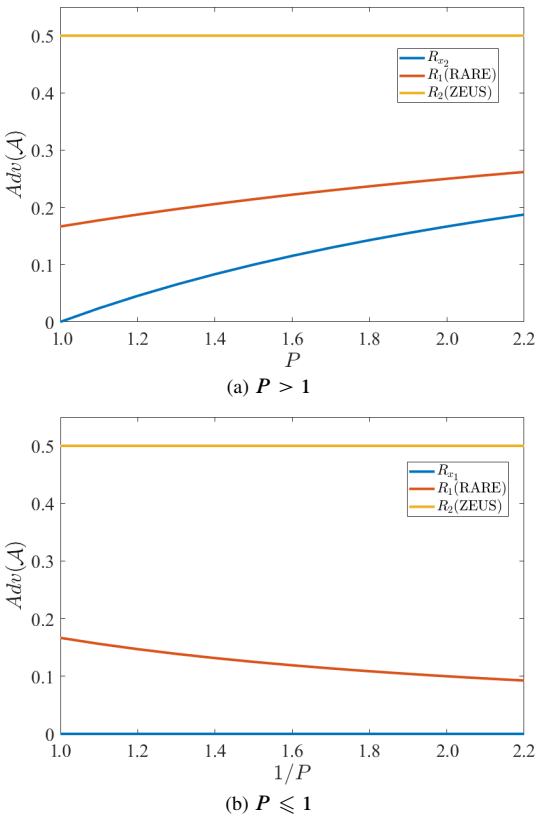
We analyze $Adv(\mathcal{A})$ as the value of P changes in RARE, ZEUS, and our basic scheme. The results are shown in Fig. 9. R_{x_2}, R_1, and R_2 (ZEUS) have the greatest influence on $Adv(\mathcal{A})$ when $P>1$. Therefore, we use these responses to analyze the security of schemes. As shown in Fig. 10, $\max\{|Pr[C|R_i]-1/2|\}$ of our scheme is lower than that of the other two when $P>1$. Similarly, when $P\leqslant 1$, R_1 and R_2 (ZEUS) are the most influential responses in RARE and ZEUS. The



(a) $P>1$

(b) $P\leqslant 1$

Fig. 9 ***Adv*($\mathcal{A}$) in RARE, ZEUS, and our basic scheme.**



(a) $P>1$

(b) $P\leqslant 1$

Fig. 10 **max{|*Pr*[*C*|R_i]−1/2|} in RARE, ZEUS, and our basic scheme.**

$Adv(\mathcal{A})$ of our basic scheme is 0, because all deduplication responses are all R_{x_1}.

The probability in our basic and enhanced schemes, RARE, and ZEUS can be formulated in Eqs. (3)–(7).

$$\text{RARE}: Pr[C|R] = \begin{cases} Pr[C|R_1] = \begin{cases} \frac{P+1}{P+2}, & P>1; \\ \frac{1}{P+2}, & P\leqslant 1; \end{cases} \\ Pr[C|R_2] = \frac{P+1}{3P+2} \end{cases} \tag{3}$$

$$\text{ZEUS}: Pr[C|R] = \begin{cases} Pr[C|R_1] = \begin{cases} \frac{P+1}{P+2}, & P>1; \\ \frac{1}{P+2}, & P\leqslant 1; \end{cases} \\ Pr[C|R_2] = 0 \end{cases} \tag{4}$$

Basic scheme ($P > 1$) :

$$Pr[C|R] = \begin{cases} Pr[C|R_{x_1}] = \frac{-P^2+3P}{-P^2+4P+1}, \\ Pr[C|R_{x_2}] = \frac{P}{P+1} \end{cases} \tag{5}$$

Basic scheme ($P \leqslant 1$): $Pr[C|R] = Pr[C|R_{x_1}] = 1/2$ (6)

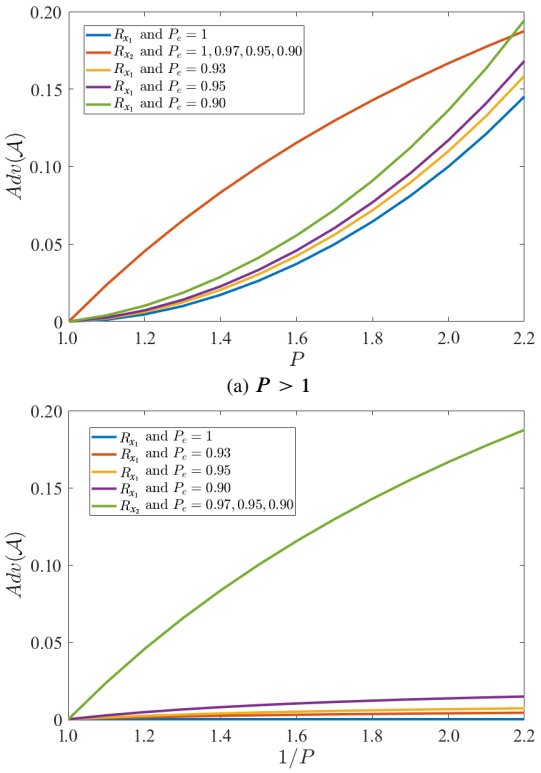
Enhanced scheme: $Pr[C|R] =$

$$\begin{cases} Pr[C|R_{x_1}] = \\ \quad \begin{cases} \frac{-P^2+3P-2P(1-P_e)}{-P^2+4P-2P(1-P_e)+1-2(1-P_e)}, & P>1; \\ \frac{P(1-P_e)-1}{(P+1)(1-P_e)-2}, & P\leqslant 1; \end{cases} \\ Pr[C|R_{x_2}] = \frac{P}{P+1} \end{cases} \tag{7}$$

We also analyze the influence of P_e on $Adv(\mathcal{A})$ in the enhanced scheme and the results are shown in Fig. 11. If $P > 1$, then P_e has no effect on $|Pr[C|R_{x_2}] - 1/2|$ and $|Pr[C|R_{x_1}] - 1/2|$ will increase as P_e goes down. However, R_{x_2} is almost always the most influential response. Thus, P_e almost has no significant impact on security. If $P \leqslant 1$, then the deduplication responses in the enhanced scheme could be R_{x_1} and R_{x_2}, which may decrease the security to some extent.

6 Performance Evaluation

We implement prototypes of our basic and enhanced schemes with C++ and use the Enron email dataset for performance evaluation. Our experiments are run on a machine equipped with i5-10210U/1.60 GHz Quad Cores CPU and 16 GB RAM, installed with 64-bit Ubuntu 20.04.1 LTS. We use MD5 from OpenSSL as the hash function to generate data tags. We randomly select 1000 files from the dataset to upload to the cloud server and then randomly select 200 files to upload for performance testing. We also implement prototypes for RARE and ZEUS and compare their performance to ours.



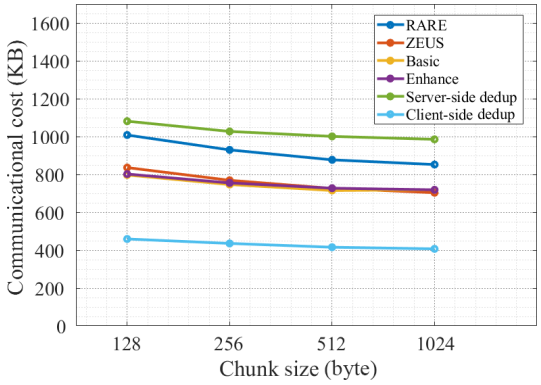
(a) $P > 1$

(b) $P \leqslant 1$

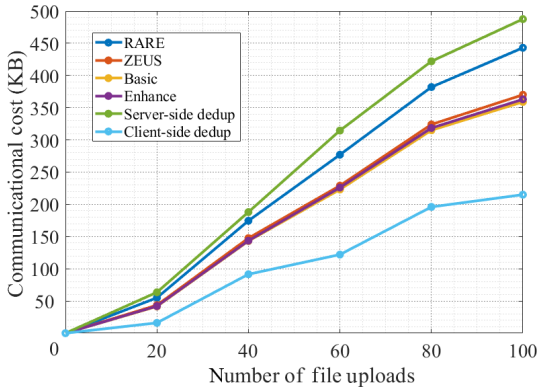
Fig. 11 Influence of P_e on $Adv(\mathcal{A})$.

We evaluate the communication costs between our basic and enhanced schemes, RARE, ZEUS, and the client-side and server-side deduplications at different chunk sizes. As shown in Fig. 12a, the client-side and server-side deduplications have the lowest and highest communication costs, respectively. The communication costs for ZEUS, and our basic and enhanced schemes are close, whereas those for RARE are significantly higher. We also evaluate the communication costs in different schemes as the number of uploaded files increases. As shown in Fig. 12b, our schemes have slightly lower communication costs than RARE and ZEUS. The reason is that the clients in our scheme need to upload fewer chunks compared with existing schemes. Particularly, the value of P_e for most files tends to be large, more than 90%, so the enhanced scheme does not increase the communication cost significantly compared with the basic scheme.

We evaluate the XOR times for chunks at different chunk sizes. As shown in Fig. 13a, the XOR times in RARE are the fewest, and those of ZEUS and our basic and enhanced schemes are close. We also evaluate the computational overheads in the four schemes, which contain the time overheads for clients to perform the XOR operation and for the server to generate

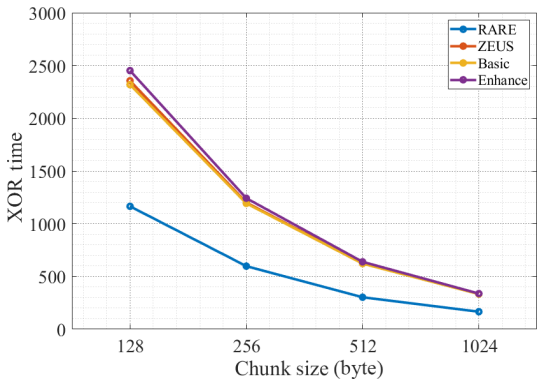


(a) $P > 1$

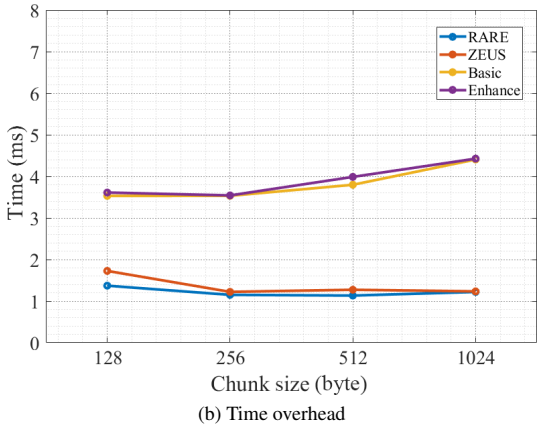


(b) $P \leqslant 1$

Fig. 12 Communicational costs in different schemes.



(a) XOR time



(b) Time overhead

Fig. 13 XOR time and time overhead in different schemes.

deduplication responses. The results are shown in Fig. 13b. Compared with RARE and ZEUS, our basic and enhanced schemes add a certain amount of time overhead as they require more XOR operations, and the server needs to pair data tags to generate deduplication responses. However, the time overhead in this part is extremely low. When uploading 200 files, our schemes only add less than 3 ms of time overhead as compared with RARE and ZEUS.

Based on the above analysis, our schemes effectively reduce communication costs for the system and do not introduce non-negligible computational overheads.

7 Conclusion

Although the client-side deduplication can be used to save storage and communication costs for cloud storage systems, deduplication responses are easily to be used as a side-channel by the adversary to violate data privacy. We argue that the threat models in existing defense schemes against side-channel attacks need to be strengthened. Thus, we propose a new threat model, that considers an adversary that could construct files containing a certain number of stored and unstored chunks to launch side-channel attacks. We propose basic and enhanced defense schemes against this kind of attack. The security analysis and performance evaluation show that the proposed schemes can effectively mitigate the privacy leakage of user outsourced data, and can effectively reduce the communication cost for the system.

Acknowledgment

This work was supported by the National Key R&D Program of China (No. 2018YFA0704703), National Natural Science Foundation of China (Nos. 61972215, 61972073, and 62172238), and Natural Science Foundation of Tianjin (No. 20JCZDJC00640).

References

[1] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. C. Zhang, and Y. K. Zhou, A comprehensive study of the past, present, and future of data deduplication, *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[2] Y. Shin, D. Koo, and J. Hur, A Survey of secure data deduplication schemes for cloud storage systems, *ACM Computing Surveys*, vol. 49, no. 74, pp. 1–38, 2017.

[3] D. T. Meyer and W. J. Bolosky, A study of practical deduplication, presented at 9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, 2011.

[4] J. Li, Z. Yang, Y. Ren, P. Lee, and X. Zhang, Balancing storage efficiency and data confidentiality with tunable encrypted deduplication, presented at 15th EuroSys Conference on Computer Systems, Heraklion, Greece, 2020.

[5] J. Li, P. P. C. Lee, Y. Ren, and X. Zhang, Metadedup: Deduplicating metadata in encrypted deduplication via indirection, presented at 35th Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 2019.

[6] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, Dark clouds on the horizon: Using cloud storage as attack vector and online slack space, presented at 20th USENIX Security Symposium, San Francisco, CA, USA, 2011.

[7] 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.

[8] Z. Pooranian, K. Chen, C. Yu, and M. Conti, RARE: Defeating side-channels based on data-deduplication in cloud storage, presented at IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Honolulu, HI, USA, 2018.

[9] C. Yu, S. P. Gochhayat, M. Conti, and C. Lu, Privacy aware data deduplication for side-channel in cloud storage, *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 597–609, 2020.

[10] F. Armknecht, J. Bohli, G. O. Karame, and F. Youssef, Transparent data deduplication in the cloud, presented at 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 2015.

[11] M. Bellare, S. Keelveedhi, and T. Ristenpart, Messagelocked encryption and secure deduplication, presented at 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, 2013.

[12] J. Li, Y. K. Li, X. Chen, P. P. C. Lee, and W. Lou, A hybrid cloud approach for secure authorized deduplication, *IEEE Trans. Parallel Distributed Syst.*, vol. 26, no. 5, pp. 1206–1216, 2015.

[13] Y. K. Zhou, D. Feng, W. Xia, M. Fu, F. T. Huang, Y. C. Zhang, and C. G. Li, SecDep: A user-aware efficient finegrained secure deduplication scheme with multilevel key management, presented at 31st IEEE Symposium on Mass Storage Systems and Technologies, Santa Clara, CA, USA, 2015.

[14] R. M. Chen, Y. Mu, G. M. Yang, and F. C. Guo, BL-MLE: Blocklevel message-locked encryption for secure large file deduplication, *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 12, pp. 2643–2652, 2015.

[15] J. W. Li, P. P. C. Lee, C. F. Tan, C. Qin, and X. S. Zhang, Information leakage in encrypted deduplication via frequency analysis: Attacks and defenses, *ACM Trans. Storage.*, vol. 16, no. 1, pp. 1–30, 2020.

[16] J. Stanek and L. Kencl, Enhanced secure thresholded data deduplication scheme for cloud storage, *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 4, pp. 694–707, 2018.

[17] P. F. Zuo, Y. Hua, C. Wang, W. Xia, S. D. Cao, Y. K. Zhou, and Y. Y. Sun, Mitigating traffic-based side-channel attacks in bandwidth-efficient cloud storage, presented at 32nd IEEE International Parallel and Distributed Processing Symposium, Vancouver, Canada, 2018.

[18] S. Keelveedhi, M. Bellare, and T. Ristenpart, Dupless: Server-aided encryption for deduplicated storage, presented at 22nd USENIX Security Symposium, Washington, DC, USA, 2013.

[19] S. Lee and D. Choi, Privacy-preserving cross-user sourcebased data deduplication in cloud storage, presented at International Conference on Information and Communication Technology Convergence, Jeju Island, Republic of Korea, 2012.

[20] F. Armknecht, C. Boyd, G. T. Davies, K. Gjøsteen, and M. Toorani, Side-channels in deduplication: Trade-offs between leakage and efficiency, presented at 12th ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2017.

[21] O. Heen, C. Neumann, L. Montalvo, and S. Defrance, Improving the resistance to side-channel attacks on cloud storage services, presented at 5th International Conference on New Technologies, Mobility and Security(NTMS), Istanbul, Turkey, 2012.

[22] Y. Shin and K. Kim, Differentially private client-side data deduplication protocol for cloud storage services, *Secur. Commun. Networks.*, vol. 8, no. 12, pp. 2114–2123, 2015.

Guanxiong Ha received the MS degree in computer science and technology from Nankai University, Tianjin, China, in 2021. He is currently pursuing the PhD degree at the College of Cyber Science, Nankai University, Tianjin, China. His research interests include cloud data security and applied cryptography.

Hang Chen is pursuing the master degree at the College of Cyber Science, Nankai University, Tianjin, China. Her main research interests are cryptography and data deduplication.

Chunfu Jia is a PhD supervisor, a professor, and the head of department in Nankai University. His main research interests include network and system security, cryptography application, and malware analysis.

Mingyue Li is currently pursuing the PhD degree at the College of Cyber Science, Nankai University, Tianjin, China. Her research interests mainly include network security and searchable encryption technology.