RARE: Defeating Side Channels based on Data-Deduplication in Cloud Storage

Zahra Pooranian
University of Padova
28 PUBLICATIONS **305** CITATIONS
SEE PROFILE

Mauro Conti
University of Padova
249 PUBLICATIONS **2,691** CITATIONS
SEE PROFILE

RARE: Defeating Side Channels based on Data-Deduplication in Cloud Storage

Zahra Pooranian[1], Kang-Cheng Chen[2], Chia-Mu Yu[3,4], Mauro Conti[1]
[1]Department of Mathematics, University of Padua, Italy
[2]Department of Computer Science and Engineering, Yuan Ze University, Taiwan
[3]Department of Computer Science and Engineering, National Chung Hsing University, Taiwan
[4]Taiwan Information Security Center (TWISC), Taiwan

***Abstract*—Client-side data deduplication enables cloud storage services (e.g., Dropbox) to achieve both storage and bandwidth savings, resulting in reduced operating cost and high level of user satisfaction. However, the deduplication checks (i.e., the corresponding essential message exchange) create a side channel, exposing the privacy of file existence status to the attacker. In particular, the binary response from the deduplication check reveals the information about the existence of a copy of the file in the cloud storage. This behavior can be exploited to launch further attacks such as learning the sensitive file content and establishing a covert channel. While current solutions provide only weaker privacy or rely on unreasonable assumptions, we propose RAndom REsponse (RARE) approach to achieve stronger privacy. The idea behind our proposed RARE solution is that the uploading user sends the deduplication request for two chunks at once. The cloud receiving the deduplication request returns the randomized deduplication response with the careful design so as to preserve the deduplication gain and at the same time minimize the privacy leakage. Our analytical results confirm privacy guarantee and results show that both deduplication benefit and privacy of RARE can be preserved.**

***Index Terms*—Cloud Storage, Data Deduplication, Data Privacy, Side Channel.**

I. INTRODUCTION

Cloud storage services such as Dropbox and Google Drive have been very popular, offering the virtually unlimited amount of storage promise and enabling service users to easily backup and synchronize their data among devices. However, despite the cloud storage's huge storage space, since the same or different users could by chance upload duplicated data, this kind of duplication wastes network resources, consumes excess power, and complicates data management. Thus, *data deduplication* technique has been widely used by cloud storage providers to eliminate the unnecessary data copies.

The commercial cloud storage services are in favor of the *cross-user client-side fixed-size-chunk-level data deduplication* to reach the highest deduplication gain. In particular, the cross-user data deduplication sees the cloud storage as a virtually single disk shared by all of the cloud users, deduplicating all of the files from different users and consequently maximizing the deduplication opportunity. The client-side data deduplication, in contrast to server-side data deduplication, is featured by the user that uploads the file hash as the duplication check request (`dc request`). Due to the collision-resistance property of cryptographic hash functions, the cloud may check the status of file existence and then sends a binary duplication check response (`dc response`) to the user, which may, in turn, suppress the explicit uploading of the file in the case that the deduplication occurs.

Unfortunately, dc response may be used as a side channel for privacy violation. The design of the countermeasure for the above side channel, in fact, introduces conflicting requirements; on the one hand, the user needs a deterministic response from the cloud to know whether the further uploading of the file is necessary, which is the fundamental idea behind the data deduplication system. On the other hand, any deterministic response can be seen as an indicator of privacy leakage.

A. Contribution

We propose RAndom REsponse (RARE) approach to eliminate the deduplication response-based side channel for cloud storage services and at the same time preserve the deduplication benefit. The privacy leakage of side channel is due to the deterministic relation between the `dc request` and `dc response`. Therefore, our developed technique, random response[1], aims to reach the probabilistic relation by allowing the cloud to randomize the dc response. The objective is to keep the deduplication gain to the certain degree and eliminate the leakage of chunk existence status. Since duplicate check of single chunk does not give sufficient room for dc response randomization, we perform the duplicate check on two chunks at once (*double chunking*, see Sec. IV-A). Moreover, *dirty chunks*, the chunks that have been queried but are not uploaded eventually can be exploited to perform repeated duplicate checks and receive independent dc responses. Dirty list for marking dirty chunks are used to disable the deduplication on dirty chunks. In summary, our proposed RARE has the following characteristics.

- **Parameterless Configuration.** The proper choice of parameters in a system can be a huge burden for engineers. Our RARE does not have parameters that need to be determined manually.
- **No Independent Server.** While some existing solutions achieve their privacy guarantee based on a particular configuration of the independent server, our RARE only involves the interactions between the user and cloud.

[1]Despite the similarity, our developed technique, random response, is in its nature different from the randomized response [15], which is usually used to preserve the individual privacy and at the same time extract the useful statistic from crowd.

TABLE I: Comparisons Between Different Side Channel Defenses (✓: has this property, ×: does not have this property).

	No Parameter	No Indep. Server	Two-Side Privacy
Mozy [12]	×	✓	×
Harnik et al. [7]	×	✓	×
Lee and Choi [11]	×	✓	×
Heen et al. [6]	✓	×	✓
Wang et al. [16]	×	✓	×
Shin and Kim [14]	✓	×	✓
Armknecht et al. [1]	×	✓	×
ZEUS [19]	✓	✓	×
ZEUS$^+$ [19]	×	✓	✓
RARE (this paper)	✓	✓	✓

- **Stronger Privacy.** Most of the prior solutions still suffer from information leakage. Our RARE reaches the minimal information leakage.

A comparison table about privacy and assumptions made by different side channel defenses is shown in Table I.

The remainder of the paper is structured as follows. We review the related work in Sec. II. The system model is reported in Sec. III. Sec. IV details our proposed solution description. The performance evaluation of the proposed solution is reported in Sec. V. Finally, Sec. VI concludes our work.

II. Related Work

In the following, we have a brief overview of the data deduplication technique (in Sec. II-A), and then describe the deduplication response-based side channel (in Sec. II-B). Finally, we briefly review the state-of-the-art solutions and their weaknesses (in Sec. II-C).

A. Data Deduplication

Data deduplication [17], [18] is an effective means widely implemented by cloud storage providers to reduce the storage and bandwidth requirements. The basic idea behind the data deduplication is to avoid storing the same file twice, which directly results in storage saving. The most aggressive implementation of data deduplication is the cross-user client-side fixed-size-chunk-level data deduplication. The term "cross-user" means that all of the cloud users share a single virtual storage, so that all of the files, irrespective of the uploading users, can be deduplicated. The term "client-side" means that the user is able to detect whether the file to be uploaded has already a copy in the cloud by sending out the deduplication check request, so as to reduce the unnecessary bandwidth waste. The term "fixed-size-chunk-level" means that the basic unit for duplicate check and cloud storage backend is fixed-size chunks. For example, Dropbox uses SHA-256 as the hash function in their implementation and 4MB as their chunk size.

The implementation of the duplicate check is, in fact, straightforward; for a chunk c to be uploaded, the user first calculates and sends the hash (or say *dc request*) $h(c)$ to the cloud, where $h(\cdot)$ denotes the cryptographic hash function. Once the cloud unable to find a copy of $h(c)$ in the memory (i.e., chunk existence), the user will receive a positive *dc response* (i.e., a signal of deduplication triggered) and has no need to upload c again. Otherwise, the user uploads c and cloud keeps $h(c)$ in the memory for subsequent duplicate checks. Throughout the paper, the term "duplicate check" refers to the procedures of exchanging dc request and dc response.

B. Side Channel

The dc response exposes the file status. In particular, the positive (negative) response, referring to as the file inexistence (file existence), gives the uploading user information about the status of file existence. This privacy leakage can serve as a primitive that leads to the following potential attacks.

- **File Confirmation.** The expose of the file existence status itself is the most straightforward privacy leakage.
- **Learning Secret Content.** The one-shot file confirmation usually does not have the much negative impact. However, with the consideration that the file has only limited min-entropy, the file confirmation threat, together with the brute-force strategy, may lead the attacker to learn the secret content of a file, which has the severe negative impact in the real world.
- **Covert Channel.** Two parties agreeing upon a common file can use the file existence status as a medium for communications. In particular, one party virtually transmits one bit to another by uploading or deleting the file, because another party may see the negative (positive) response as 1 (0).

C. Prior Defenses Against Side Channel

Harnik et al. [5] first identify the risk of side channel in the deduplicated cloud storage. In particular, they find the existence of the side channel in the deduplicated cloud storage, because the cloud needs to deterministically return the deduplication response according to the existence status of a specific file. Addressing this weakness, Harnik et al. propose a solution, random threshold (RT), to randomize the deduplication threshold. In particular, instead of associating a single deduplication threshold to each file, a random deduplication threshold only known by the cloud is associated with different files in RT. Any user without the knowledge of deduplication threshold cannot infer the status of file existence from the dc response. However, RT in its nature has a limitation that when the number of copies of the requested file exceeds the deduplication threshold, because of the necessary deduplication gain, the deduplication response invariably suppresses the explicit file uploading, exposing the privacy of file existence. Despite the above weakness, a couple of works have been done based on the same design philosophy. For example, Lee and Choi [11] propose a method to randomize the deduplication threshold on the fly, in contrast to determining deduplication thresholds beforehand in RT. Armknecht et al [1] argue the similarity between [5] and [11]. Wang et al. [16] determines the deduplication thresholds in RT based on a game-theoretic approach.

Another line of research that addresses the same problem introduces an extra trusted hardware between the cloud and users, so as to the obfuscate the network traffic. For example, Heen et al. [6] consider a setting that a trusted gateway bridges the cloud and users, so that a number of files are transmitted to the gateway, which then forwards to the cloud. The gateway in this sense can randomize the deduplication response. On the other hand, Shin and Kim [14] have a design of the deduplication protocol with differential privacy based on the existence of an independent trusted server bridging the cloud and users.

Despite the lack of theoretical foundation, Mozy [12] takes a pragmatic approach; it setups a threshold θ for the size so that only the file with size greater than θ needs to be deduplicated. The rationale behind such a design is that the large-size files such as music and movie are not sensitive. Only small-size files such as document contain sensitive information. If the cloud does not deduplicate the sensitive content, the corresponding privacy concern can be eliminated.

Yu et al. [19] also identify the privacy weakness of RT and propose ZEUS as an alternative to provide stronger privacy guarantee. Moreover, Yu et al. further propose ZEUS$^+$ to enhances the privacy; ZEUS$^+$ achieves an even stronger privacy, compared to ZEUS, by sacrificing the advantage of parameterless configuration.

III. System Model

This section first aims to provide an overview of the network model. Then, we describe the threat model that we used in this paper. Finally, we discuss the privacy notion.

A. Network Model

We consider a trusted cloud storage with cross-user client-side data deduplication. Basically, in order to upload a file f into the cloud S, first, the file should be divided into chunks c with bit length ϕ. In principle, to upload a chunk c, one needs to upload the deduplication check request $h(c)$ and waits for the binary deduplication check response, so as to determine whether further explicit uploading of c is necessary.

However, the user, acting as the attacker $\hat{c}$ in our consideration, with the chunk c is aimed to determine the existence status of c from the interactions between the user and cloud.

B. Threat Model

We assume that the user (including external attacker) cannot verify the existence of chunk c unless the user uploads c. Moreover, we suppose that the probability (P) of the event that an arbitrary chunk is in the cloud is quite small. The objective of the external attacker $\hat{c}$ is to know the existence status of the chunk. Considering the conventional deduplication framework, the attacker has to perform the duplicate check to learn whether $\tilde{c}$, the chunk of attacker's interest, has already a copy in the storage system. As mentioned before, the duplicate check carries out by exchanging dc request and dc response. In our context, if the dc request is positive (negative), then $\hat{c}$ will conclude that the chunk (non)existence in the cloud storage. In addition, it is not mandatory for the attacker to complete the chunk uploading. Therefore, we supposed that they have obligation to terminate transmission in the middle.

C. Privacy Notion

There are two privacy notions, *existence privacy* and *inexistence privacy*, which are described as follow.

Basically, the existence privacy definition is based on the case that the attacker can not gain any information about the chunks existence status in the cloud unless she uploads them. More in detail, when a duplicate check protocol guarantees the existence privacy, it means that its dc response does not leak information on the existence status of c. We formally define the notion of existence privacy as follow.

Definition 1. *Let us define duplicate check protocol by $f(c, aux)$, where c is the attacker chunk's interest and aux is the required auxiliary data for the duplicate check. Assume that C denotes the case that c is in the cloud. The attacker has no prior knowledge of the existence status of c, and her aim is to know the existence status of c. If $P[C|f(c, aux)] = P[C]$, regardless that $f(c, aux)$ has clearly demonstrated the existence of c, $f(c, aux)$ achieves existence privacy.*

On the contrary, the inexistence privacy achieves when the attacker cannot identify the chunk inexistence. Correspondingly, the formal definition of inexistence privacy is as follow.

Definition 2. *Let us define duplicate check protocol by $f(c, aux)$, where c is the attacker chunk's interest and aux is the required auxiliary data for the duplicate check. Assume that $\bar{C}$ denotes the case that c is not in the cloud. The attacker has no prior knowledge of the existence status of c, and her aim is to know the existence status of c. If $P[\bar{C}|f(c, aux)] = P[\bar{C}]$, regardless that $f(c, aux)$ has clearly demonstrated the inexistence of c, $f(c, aux)$ achieves inexistence privacy.*

In addition, if both existence privacy and inexistence privacy are fulfilled we achieve *two-side privacy*.

We conclude that the meaning of a two-side private deduplicate check protocol is that the dc response does not give us any extra information about the existence status of a determined chunk, in according to the Definition 1 and Definition 2. Likewise, in the following, we describe a weaker version of existence privacy.

Definition 3. *Let us define duplicate check protocol by $f(c, aux)$, where c is the attacker chunk's interest and aux is the required auxiliary data for the duplicate check. Assume that C denotes the case that c is in the cloud. The attacker has no prior knowledge of the existence status of c, and her aim is to know the existence status of c. If $P[C|f(c, aux)] = 1/2$, regardless that $f(c, aux)$ has clearly demonstrated the existence of c, $f(c, aux)$ achieves weaker existence privacy.*

In the Definition 1, $P[C|f(c, aux)] = P[C] = P$, while in Definition 3, $P[C|f(c, aux)] = P[C] = 1/2$. Clearly, however numerical probability in Definition 1 is increased from P to $1/2$ in Definition 3, but this increase does not have any effect on the verification probability of the chunk existence status by the attacker.

In our context, the two-side private deduplicate check protocol means that we succeed to have both weak existence privacy and inexistence privacy, unless stated otherwise.

IV. Proposed solution

In this section, we first present the basic idea of the RARE algorithm (Sec. IV-A). Then, we describe in more detail each part of the RARE algorithm (Sec. IV-B). Finally, we analyze the security of RARE algorithm (Sec. IV-C).

A. Basic Idea of RARE

There is a large number of deduplication techniques proposed in the literature. However, they failed to meet two-side privacy without using any extra hardware and depending on heuristically chosen parameters. Before describing the algorithm, we provide an overview of the basic idea behind the design of RARE scheme. The rational behind the proposed RARE scheme lies in two observations.

- Duplicate check of single chunk does not give sufficient room for dc response randomization.
- Sybil attacker can perform a huge number of independent duplicate checks on the same chunk, attempting to counteract arbitrary randomization technique for duplicate checks.

We introduce a novel design on the duplicate check; instead of uploading a single chunk, duplicate check uploads two chunks at once. Without a proper modification, the duplicate check of two chunks is equivalent to performing ordinary duplicate check twice. In order to hide the chunk existence status, we carry out the encodings on both the dc response and the chunks to be uploaded. Clearly, for the dc request $\langle h(c_1), h(c_2)\rangle$ in double chunk uploading, the dc response consists of a single number that indicates the number of chunks needed to be uploaded, instead of a pair of ordinary dc responses. The final design is the RARE table shown in Table II, which can be interpreted as follows. If both the queried chunks (e.g., c_1 and c_2) are not in S, c is indeed required to upload two individual chunks. Otherwise, c is asked to upload either two individual chunks or the exclusive-or (XOR) $c_1 \oplus c_2$ of two chunks, each with probability 1/2. In such a design, S is able to derive another chunk given a chunk in possession.

One can see from RARE that c gains no extra existence status information if receiving 2 from S, and gains the only extra information that at least one queried chunk has a copy in S otherwise. However, such a RARE design is still flawed; if always receiving 2 from the repeated duplicate check on $\langle h(c_1), h(c_2)\rangle$, $\hat{c}$ confirms the nonexistence of c_1 and c_2. Given the chunk c_3 of $\hat{c}$'s interest, $\hat{c}$ repeatedly performs duplicate check on $\langle h(c_1), h(c_3)\rangle$. The attacker $\hat{c}$ confirms the existence of c_3 if receiving at least one dc response 1, and confirms the nonexistence otherwise. The root cause is the abuse of duplicate check; $\hat{c}$ performs duplicate check but does not upload queried chunks or XORed chunk eventually. Here, we propose to use the dirty bit to mark the chunks (i.e., *dirty chunks*) that have been queried but are not uploaded eventually. We can implement the previous policy by keeping all hashes of dirty chunks in a list (termed as *dirty chunk list*, $\mathcal{L}$). Therefore, the cloud is able to check if the receiving dc request is on the dirty list or not. If either one in duplicate check $\langle h(c_1), h(c_2)\rangle$ is in $\mathcal{L}$, S always returns 2.

TABLE II: RARE table.

c_1 **existence**	c_2 **existence**	**dc response**
0	0	2
0	1	1 or 2
1	0	1 or 2
1	1	1 or 2

Algorithm 1 RARE.

```
    Input: file $f$ with chunk size $\phi$, dirty chunk list $\mathcal{L}$;
 1: user partitions $f$ into chunks $c_1, \dots, c_n$;
 2: user sets $\hat{n} = n$;
 3: if bit length $|c_n| \neq \phi$ then
 4:     user performs padding to $c_n$;
 5: end if
 6: if $n$ is odd then
 7:     user picks random chunk $c_{n+1}$ and $\hat{n} = n + 1$;
 8: end if
 9: for $i \in \{1, 3, \dots, \hat{n} - 1\}$ do
10:     user performs duplicate check on $\langle h(c_i), h(c_{i+1})\rangle$;
11:     if $h(c_i) \notin \mathcal{L}$ and $h(c_{i+1}) \notin \mathcal{L}$ then
12:         cloud replies 1 or 2 according to Table II;
13:     else
14:         cloud replies dc response 2;
15:     end if
16:     if user receives dc response 1 then
17:         user uploads $c_i \oplus c_{i+1}$ or $c_i$ and $c_{i+1}$ to the cloud;
18:         if cloud does not receive $c_i \oplus c_{i+1}$ then
19:             $\mathcal{L} = \mathcal{L} \cup \{c_i, c_{i+1}\}$;
20:         end if
21:     else
22:         user uploads $c_i$ and $c_{i+1}$ to the cloud;
23:         if cloud does not receive $c_i$ and $c_{i+1}$ then
24:             $\mathcal{L} = \mathcal{L} \cup \{c_i, c_{i+1}\}$;
25:         end if
26:     end if
27: end for
```

B. RARE Description

Algorithm 1 reports the description of RARE, where the user attempts to upload a file f to the cloud. First of all, the file f is divided into chunks (line 1). Since we have double chunk uploading in RARE, then the users have to check whether the chunks number is even and whether the size of the last chunk is equal to the predefined chunk size. If not, we need to generate a sequence of bit depending on adequate length and concatenate it to the file f (lines 3-8). Now, the user carries out duplicate check on chunks pairs, $\langle h(c_i), h(c_{i+1})\rangle$, $i \in [1, 3, \dots, \hat{n} - 1]$ (line 10). When cloud receives $\langle h(c_i), h(c_{i+1})\rangle$, it checks if the involved chunks are dirty (line 11). More in detail, if the current chunks are on the dirty, the cloud always returns 2 to the user. Otherwise, cloud decides to return either 1 or 2 based on Table II (line 12). According to the received dc response, the user either uploads $c_1 \oplus c_2$ or uploads c_1 and c_2 explicitly to the cloud (lines 16-27).

C. Security Analysis of RARE

Duplicate check is the only way for the attacker to breach the privacy existence status of a specific chunk. However, RARE table and dirty chunk list work as the side channel defense with the inexistence privacy and weak existence privacy. The privacy evaluation of RARE (see Table II) is as follow.

Theorem 1. *RARE achieves weak existence privacy and inexistence privacy.*

Proof. If both c_1 and c_2 in the dc request $\langle h(c_1), h(c_2)\rangle$ are not controlled by the attacker, where c_2 is the chunk of attacker's interest, the attacker can confirm the nonexistence of c_1 and c_2 if always receiving 2. Moreover, if attacker receives at least one dc request 1, then attacker confirms the existence of c_2. These privacy leakages are because of the duplicate check abuse, due to the fact that attacker performs duplicate check but does not upload the requested chunks. As we discussed this problem in the previous section, we can handle this problem by using dirty chunk list. The RARE table along with the dirty bit policy ensure the privacy of chunk existence status.

The only time that attacker is able to find the existence of the certain chunk in the cloud is when receiving dc response 1. Given chunk c_2 of $\hat{c}$'s interest and the dc response 1, the probability of c_2 in the cloud is formulated as follows: □

$$
\begin{aligned}
&Pr[C_2|R_1] && (1.1)\\
&= \frac{Pr[R_1|C_2]Pr[C_2]}{Pr[R_1|C_2]Pr[C_2] + Pr[R_1|\bar{C}_2]Pr[\bar{C}_2]} && (1.2)\\
&= \frac{1/2 \times Pr[C_2]}{1/2 \times Pr[C_2] + Pr[R_1|\bar{C}_2](1 - Pr[C_2])} && (1.3)\\
&= \frac{1/2 \times Pr[C_2]}{1/2 \times Pr[C_2] + (Z_1 + Z_2)(1 - Pr[C_2])} && (1.4)\\
&= \frac{1/2 \times P}{1/2 \times P + (0 \times (1-P) + 1/2 \times P)(1-P)} = \frac{P}{2P - P^2}. && (1.5)
\end{aligned}
$$

$Z_1 \triangleq Pr[R_1|\bar{C}_2 \cap \bar{C}_1]Pr[\bar{C}_1]$ and $Z_2 \triangleq Pr[R_1|\bar{C}_2 \cap C_1]Pr[C_1]$ refer to law of total probability. Where C_2 and $\bar{C}_2$ represent the event when chunk c_2 is in the cloud or is not, respectively. We used R_1 to show the event when the cloud response is equal to 1. Eq. (1.2) refers to conditional probability. In Eq. (1.3), $P[R_1|C_1] = 1/2$. In Eq. (1.5), P denotes the probability that an arbitrary chunk is in S. As P is usually rather small, the attacker still cannot make sure c_2 is in the cloud even when seeing dc response 1.

V. Performance Evaluation

We first describe the scenario under investigation (Sec. V-A) and then the obtained results (Sec. V-B).

A. Scenario Description

In the following, we first detail the metric we used for validating our algorithm in different chunk sizes, then, we describe the dataset used in our evaluation. The metric we adopted our algorithm is a *communication cost*. The communication cost is defined as the number of bits required during the entire chunk uploading process, including the duplicate check (i.e., dc request and dc response) and explicit chunk uploading (i.e., the chunk c, if necessary). Moreover, we consider dirty chunk as the case that no deduplication is applied on chunks and all of the dc requests relevant to dirty chunks will not trigger deduplication. In other words, this criteria is used to prevent the attacker from gaining existence status information by iteratively performing dc requests on the same chunks. Thus, the use of dirty chunks actually compromises the deduplication benefit. Hence, the same set of evaluations applies to different ratios of dirty chunks.

We used Enron Email Dataset for our evaluation in which the traces are presented in Fig. 1. In Enron Email dataset [4], as the real application, ordinary users can backup the email to the cloud storage. The carried out test is done on Apple Mac 64 bit OS Sierra version 10.12 on Intel Core i5 2.9 GHz and 8GB of RAM. The simulations are carried out by exploiting the numerical software of the Python 3.7.6 platform. In order to implement the method, we use the hash function SHA-256 from OpenSSL library. We deleted all of the files smaller than 5KB from Enron dataset. Then, we picked 1000 files uniformly at random and uploaded them to the cloud. Finally, we chose 200 files uniformly at random to perform duplicate checks and explicit chunk uploading, if necessary.

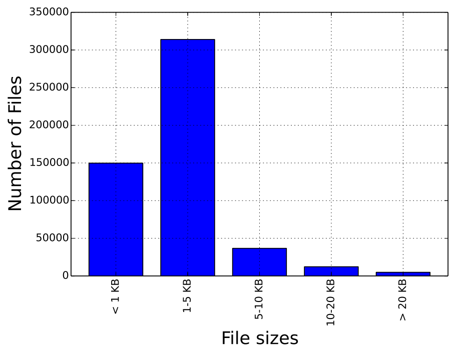
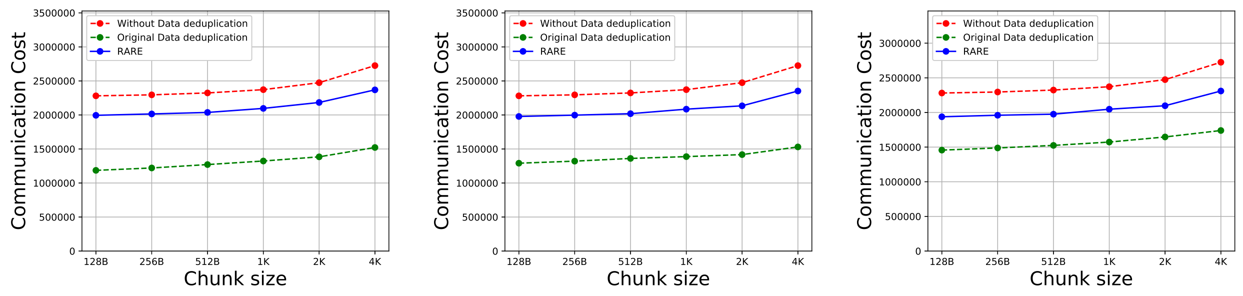


Fig. 1: The statistics of Enron email dataset [4].

B. Simulation Results

In this subsection, we run the RARE scheme and evaluate the resulting average communication cost under an Enron Email dataset input arrival sequence with various values of chunk size. Figs. 2(a), 2(b), and 2(c), report the communication cost vs. chunk size between original data deduplication, RARE and without data deduplication methods *without* considering dirty chunks, *with* considering 10% dirty chunks and *with* considering 25% dirty chunks, respectively. In these figures, the number of dirty chunks would grow with the increasing number of dc requests. We randomly select a fixed percentage of chunks as dirty chunks to test the impact of the number of dirty chunks on the communication cost. Obviously, more dirty chunks are used, more communication costs are consumed. Because, when the cloud finds either chunk in the dc request dirty, it cancels the deduplication functionality for the dc request and imply more communication costs.

We conclude that: if no dirty chunk is applied, it means that, benign users infrequently have the abnormal disconnection,



(a) No dirty chunk. (b) 10% dirty chunks. (c) 25% dirty chunks.

Fig. 2: Communication cost (bits) for different chunk sizes.

the vulnerability is much higher and the attacker may be unwilling to create the deduplication-based side channel. As a result, there would be very few percentages of dirty chunks in the cloud. From the figure, it is clear that the original data deduplication has the lowest communication cost compared to the other method. This is due to the fact that this method finds the maximum opportunity for the data deduplication and could decrease the communication cost much higher that RARE and without data deduplication. Although this method has lowest communication cost, our algorithm, RARE supports the weak existence privacy and inexistence privacy (according to Theorem 1). Moreover, the gap between the communication costs of original data deduplication and RARE actually is dependent on the dataset characteristic.

VI. Conclusions

Due to the prevalent use of client-side data deduplication in commercial cloud storage services, data deduplication-based side channel creates a realistic and significant privacy threat. In this paper, we develop a solution, RARE, based on the framework of random response, preventing the attacker from gaining the existence status information from duplicate checks. The implementation of RARE requires minimal modification of the ordinary deduplication mechanism, incurring minimal engineering effort, in addition to its privacy and performance guarantee.

Acknowledgments

This research was partly funded by MOST 105-2923-E-001- 002-MY2 (MOST-CNR joint project), MOST 104-2628-E-155- 001-MY2, MOST 105-2923-E-002-014-MY3, MOST 105-2218-E-155-010, MOST 106-3114-E-005-001, MOST 106-2218-E-155-007. This work is also partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061), the EU-India REACH Project (agreement ICI+/2014/342-896), by the project CNR-MOST/Taiwan 2016-17 “Verifiable Data Structure Streaming”, the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation, and by the grant ”Scalable IoT Management and Key security aspects in 5G systems” from Intel.

References

[1] F. Armknecht, C. Boyd, G. T. Davies, and Gjøsteen. Side channels in deduplication: trade-offs between leakage and efficiency. *ACM Conference on Computer and Communications Security (ASIACCS)*, 2017.
[2] M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.
[3] M. Dutch. Understanding data deduplication ratios. *SNIA Data Management Forum*, 2008.
[4] Enron Email Dataset. `https://www.cs.cmu.edu/~./enron/`
[5] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of ownership in remote storage systems. *ACM conference on Computer and Communications Security (CCS)*, 2011.
[6] O. Heen, C. Neumann, L. Montalvo, and S. Defranc. Improving the resistance to side-channel attacks on cloud storage services. *International Conference on New Technologies, Mobility and Security (NTMS)*, 2012.
[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. 4047, 2010.
[8] S. Keelveedhi, M. Bellare, and T. Ristenpart. Dupless: Server-aided encryption for deduplicated storage. *USENIX Security Symposium*, 2013.
[9] J. Liu, N. Asokan, and B. Pinkas. Secure deduplication of encrypted data without additional independent servers. *ACM Conference on Computer and Communications Security (CCS)*, 2015.
[10] J. Li, X. Chen, M. Li, J. Li, P. P. C. Lee, and W. Lou. Secure deduplication with efficient and reliable convergent key management, *IEEE Transactions on Parallel and Distributed System*s, vol. 25, no. 6, pp. 1615-1625, 2014.
[11] S. Lee and D. Choi. Privacy-preserving cross-user source-based data deduplication in cloud storage. *International Conference on ICT Convergence (ICTC)*. 2012.
[12] Mozy. `https://mozy.com/`
[13] V. Rabotka and M. Mannan. An evaluation of recent secure deduplication proposals. *Journal of Information Security and Applications*, vol. 27, pp. 318, Apr. 2016.
[14] Y. Shin and K. Kim. Differentially private client-side data deduplication protocol for cloud storage services. *Security and Communication Networks*, vol. 8, pp. 2114 - 2123, 2015.
[15] S. L. Warner. Randomised response: a survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 1965.
[16] B. Wang, W. Lou, and Y. T. Hou. Modeling the side-channel attacks in data deduplication with game theory. *IEEE Conference on Communications and Network Security (CNS)*, 2015.
[17] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou. A comprehensive study of the past, present, and future of data deduplication. *Proceedings of the IEEE*, vol. 104, pp. 16811710, Sept 2016.
[18] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Y. Zhang, and Q. Liu. FastCDC: a fast and efficient content-defined chunking approach for data deduplication. *USENIX Annual Technical Conference*, 2016.
[19] C.-M. Yu, S. P. Gochhayat, and M. Conti. Privacy aware data deduplication for side channel in cloud storage. in press by *IEEE Transactions on Cloud Computing*, vol. 99, pp. 114, January 2018.