A secure two-phase data deduplication scheme

Pierre Meye*, Philippe Raipin*, Frédéric Tronel†, Emmanuelle Anceaume‡
*Orange Labs, France, {pierre.meye, philippe.raipin}@orange.com
†Supelec, France, frederic.tronel@supelec.fr
‡CNRS / IRISA, France, anceaume@irisa.fr

***Abstract*—Data grows at the impressive rate of 50% per year, and 75% of the digital world is a copy[1]! Although keeping multiple copies of data is necessary to guarantee their availability and long term durability, in many situations the amount of data redundancy is immoderate. By keeping a single copy of repeated data, data deduplication is considered as one of the most promising solutions to reduce the storage costs, and improve users experience by saving network bandwidth and reducing backup time. However, this solution must now solve many security issues to be completely satisfying. In this paper we target the attacks from malicious clients that are based on the manipulation of data identifiers and those based on backup time and network traffic observation. We present a deduplication scheme mixing an intra- and an inter-user deduplication in order to build a storage system that is secure against the aforementioned type of attacks by controlling the correspondence between files and their identifiers, and making the inter-user deduplication unnoticeable to clients using deduplication proxies. Our method provides global storage space savings, per-client bandwidth network savings between clients and deduplication proxies, and global network bandwidth savings between deduplication proxies and the storage server. The evaluation of our solution compared to a classic system shows that the overhead introduced by our scheme is mostly due to data encryption which is necessary to ensure confidentiality.**

***Keywords*—Cloud storage; Intra-user deduplication; Inter-user deduplication; Confidentiality; Deduplication proxy;**

I. Introduction

The amount of data to be stored by cloud storage systems increases extremely fast. It is thus of utmost importance for Cloud Storage Providers (CSPs) to dramatically reduce the cost to store all the created data. A promising approach to achieve this objective is through data deduplication [1], [2]. Put simply, data deduplication keeps a single copy of repeated data. When a client wishes to store some piece of data, and if a copy of this data has already been saved in the storage system, then solely a reference to this existing copy is stored at the storage server. No duplicate is created. There are diverse forms of data deduplication. It can be done by a client solely on the data he/she has previously stored in the system, a technique commonly called *intra-user deduplication*, or it can be achieved by taking into account the data previously stored by all the clients. In this case it is designated as *inter-user deduplication*. Data deduplication also improves users experience by saving network bandwidth and reducing backup time when the clients perform the deduplication before uploading data to the storage server. This form of deduplication is termed as *client-side deduplication*, and when it is handled by the storage server it is called *server-side deduplication*. Due to its straightforward economical advantages, data deduplication is gaining popularity in both commercial and research storage systems [1], [3], [2].

However, several works have recently revealed important security issues leading to information leakage to malicious clients [4], [5], [6], [7]. These security concerns arise especially in systems performing an inter-user and client-side deduplication which is unfortunately the kind of deduplication that provides the best savings in terms of network bandwidth and storage space. Hereafter, we summarize in three categories the most common types of attacks described in the literature.

Manipulation of data identifiers: A common technique to deduplicate data is by hashing the content of the data and using this (unique) hashed value as the identifier of the data. Then the client sends this identifier to the storage server to determine whether such an identifier already exists in the system. An attacker can easily exploit this technique to perform various types of attacks. The CDN attack [5] turns a cloud storage system into a Content Delivery Network (CDN). Suppose that Bob wants to share a file F with Alice. Bob uploads F to a cloud storage system and sends F identifier to Alice. When Alice receives it, she tries to upload F to the same cloud storage system by sending F identifier. The storage system will detect that this identifier already exists. Consequently, solely a reference meaning that Alice also owns file F will be stored in the system which is actually wrong. At this point, when Alice wants to download F, she just needs to request it from the cloud storage system. There is also an attack called targeted collision [7] in which, a malicious client uploads a piece of data (*e.g*, a file) that does not correspond to the claimed identifier. Suppose that Bob wants to cheat Alice. If no control is made by the cloud storage system, Bob can upload a file F_1 with the identifier of a file F_2. Then, if Alice wishes to upload F_2 with its real identifier, the system will detect the existence of F_2 identifier in the system and will not store F_2. Rather, the system will store a reference meaning that Alice also owns file F_1 which is the file corresponding to the identifier of F_2 in the system. Later, when Alice will request the file corresponding to the identifier of F_2, the system will send F_1 to Alice.

Network traffic observation: Monitoring the network traffic on the client side gives an attacker the capability to determine whether an inter-user deduplication has been applied on a given piece of data. Such an attacker can exploit this knowledge to identify data items stored in a cloud storage system by other users or even learn the content of these data items [8], [5]. For instance, to determine whether a file F is stored in

[1]"The digital universe decade. Are you ready?", by John Gantz and David Reinsel, IDC information, may 2010

a system, the attacker just tries to upload F and observes the network traffic from his/her device toward the storage server. If F is uploaded, it means that F does not exist in the storage system. The attacker is not even obliged to upload the complete file. He/she can prematurely abort the upload, so that the attack can be repeated later. On the other hand, if the file is not uploaded, the attacker can conclude that the file already exists in the system. This fact makes the storage system vulnerable to brute force attacks on file contents [8], [5]. For instance, to determine which copies of a movie exist on a specific cloud storage system, an attacker can upload each format or version of such a movie, and show that those that are not uploaded already exist in the storage system.

Backup time observation: Detecting an inter-user deduplication by observing the backup duration [9] gives an attacker the ability to perform the same attacks as the ones done with network traffic observations. However, observing the duration of a backup operation is less accurate than a network traffic monitoring as it depends on the size of the file and the state of the network. For small files, observation may not be accurate, while for larger ones, it gains in accuracy.

There have been lots of efforts devoted to deal with these issues and provide a secure and efficient data deduplication [8], [10], [11], [4], [5], [12], [13], [6], [14], [7], [15], [16]. However, we are not aware of any single solution that is capable of addressing, at the same time, the three types of attacks that malicious users can attempt on the deduplication system. This is the objective of this paper.

A. Our contributions

Our work takes place in the context of an Internet Service Provider (ISP) providing also the storage system[2]. That is to say, the ISP which is also the CSP has strong economical reasons to *(i)* save storage space and network bandwidth as it masters all the network and storage infrastructure, and *(ii)* provide a secure storage service to its consumers. We propose a deduplication solution to build a storage system that positively answers the following three questions:

1) Can we guarantee that the identifier used to trigger deduplication corresponds to the claimed data item?
2) Can we determine that a client actually owns the data item corresponding to the identifier issued to the storage system?
3) Can we make the inter-user deduplication transparent (*i.e.* unnoticeable even in the case of backup time or network traffic observation) to clients and still provide network bandwidth savings?

Our deduplication scheme is simple and robust against the aforementioned attacks while remaining efficient in terms of storage space and bandwidth savings for both clients and the CSP. We consider data deduplication at a file level granularity but our solution can be extended to the block level.

Specifically our approach is a two-phase deduplication that leverages and combines both intra- and inter-user deduplication techniques by introducing *deduplication proxies* (DPs) between the clients and the storage server (SS). Communications from clients go through these DPs to reach the SS which allows to split the deduplication process as illustrated in Figure 1.

[2]For instance the french ISP Orange provides various cloud storage system.

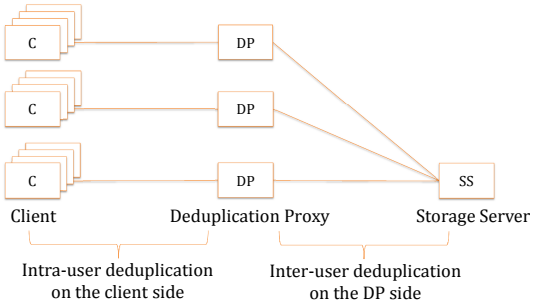

Fig. 1: Architecture of the two-phase deduplication scheme.

The intra-user deduplication: This phase is performed by clients. It aims at saving clients network bandwidth by guaranteeing that each client uploads no more than once each of his/her file to a DP (*i.e.* per-client savings). In addition, we strengthen data confidentiality by letting clients encrypt their files before uploading them. Regarding security, we exploit the fact that the scope of the intra-user deduplication is limited to the files previously stored by the client trying to store a file. It allows the storage system to be protected against both CDN attacks and network traffic observations ones. Briefly, in the former case, each client uploads each of his/her files to his/her associated DP exactly once even if the same files already exist in the storage system (due to a prior upload by an other client). This accordingly prevents any client from determining whether a file already exists in the storage system through network traffic observations.

The inter-user deduplication: This phase is performed by the DPs. Alongside the economical benefits offered by this scheme thanks to global storage space savings (*i.e.* only a single copy of a file is stored independently of the number of owners of this file), it provides also global network bandwidth savings between DPs and the SS as a single copy of any file has to be sent to the SS. Note that to make indistinguishable the upload of a file to the SS from the sending of a reference — when the referred file already exists in the storage system — the concerned DP may introduce an extra delay before sending the notification to the client. The goal of adding this delay is to make similar the duration of storing a reference and storing the entire file. This aims to protect the system against attacks based on backup time observation.

Consistency of files and their identifiers: To protect the storage system against targeted-collision attacks, consistency between a file and its identifier is checked by the DPs before sending the file to the SS and before applying the inter-user deduplication (*i.e.* sending only a reference to the SS).

The remainder of the paper is organized as follows. Section II details the system model and the assumptions we made about the system. Section III focuses on the two fundamental operations that allow to store and retrieve files in the system. Section IV presents a performance evaluation of our solution, and Section V a discussion on security issues. Section VI exposes the related work. Conclusion and future work are presented in Section VII.

II. System model

Our system consists of the following three types of components:

Client (C): Any authenticated user accessing the storage system.

Storage Server (SS): A server in charge of storing and serving clients files. The storage server also maintains an index of all the files stored in the storage system and their owners.

Deduplication Proxy (DP): A server associated with a given number of clients. Clients communicate with the SS via their associated deduplication proxy. A deduplication proxy is involved in both the intra-user and the inter-user deduplication (See Section III).

The orchestration of the proposed architecture is illustrated in Figure 1. The SS and the DPs are operated by the CSP.

A. Threat model

We suppose that the CSP is trusted to behave correctly to ensure the clients files availability and long term durability, however it follows the honest but curious adversary model. Namely, it may take actions to disclose file contents. Regarding clients, we assume that some of them are malicious. They may try to perform the types of attacks mentioned in Section I. We suppose that there is no coalition between the entities provided by the CSP (*i.e.* the SS and the DPs) and the clients. Finally, we consider that communication channels are reliable through the use of cryptographic protocols such as Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

B. Data encryption

We make the hypothesis that entities have access to a hash function, denoted by `hash` in the following, that is pre-image resistant (*i.e.* given a hash value, it should be hard to find any file that hashes to that hash value), second pre-image resistant (*i.e.* given a file F_1, it should be hard to find a different file F_2 such that F_1 and F_2 hash to the same value), and collision resistant (*i.e.* it should be hard to find two different files F_1 and F_2 that hash to the same value).

While the adoption of cloud storage services to outsource data is steadily growing, its benefits come with some inherent risks implied by the fact that full control on data is given to the CSPs [17], [18], [19]. Clients (*e.g.*, individuals, scientists, enterprises, governmental agencies, etc) may want to restrict the control or access of the CSP to their sensitive data in order to preserve some confidentiality. We assume that clients have the ability to encrypt and decrypt data with both asymmetric ($\texttt{encrypt}_{asym}$ and $\texttt{decrypt}_{asym}$) and symmetric ($\texttt{encrypt}_{sym}$ and $\texttt{decrypt}_{sym}$) cryptographic functions and manage their own private and public personal keys for the asymmetric encryption scheme. In our approach, clients encrypt their files using a convergent encryption scheme [8], [10], [20], [13], [15]. Specifically, to encrypt a file F with this scheme, the hashed value of F is used as an encryption key to encrypt F using a cryptographic symmetric function. Thus, the same file encrypted by different clients will result in equal ciphertexts. This allows the CSP to detect duplicates and to apply an inter-user deduplication on encrypted files without any knowledge of their plaintext contents.

III. System design

This section describes the `put` and `get` operations, that allow clients to respectively store and retrieve their files.

A. The `put` operation

When a client wishes to store a file in the storage system, he/she triggers a `put` operation. This operation, illustrated in Figure 2, involves two types of interactions, the first one between the client and a DP, and the second one between this DP and the SS. In the following, the client invoking the `put` operation for a file F is denoted by its identifier C_{id}. Before interacting with the DP, C_{id} builds the different parameters of the functions of the `put` operations. First, it creates the *encryption key* by hashing F content as follows,

$$F_{hash} \leftarrow hash(F).$$

Then, C_{id} encrypts the file F by applying a symmetric encryption function on F using F_{hash} as a key leading to the ciphertext $\{F\}_{F_{hash}}$. We have

$$\{F\}_{F_{hash}} \leftarrow encrypt_{sym}(F_{hash}, F).$$

Finally, C_{id} creates F identifier F_{id} as

$$F_{id} \leftarrow hash(\{F\}_{F_{hash}}).$$

The previous steps allow different clients to create the same identifier for a given file, and enable to check the consistency between an encrypted file and its identifier by just comparing the hashed value of the encrypted file and the received identifier. Once client C_{id} has created the parameters of the `put` operation, he/she sends a `checkForDedup` request to the DP to inform it that he/she wants to store F. This request is directly forwarded to the SS that checks whether a data deduplication and which kind of deduplication must be applied (*i.e.* either an intra-user deduplication or an inter-user one). This is achieved by looking for F_{id} and its potential owners in the file ownership index maintained by the SS. According to the outcome of this search, the SS sends one of the following three responses to the DP.

1) (C_{id}, F_{id}, `Intra-user deduplication`). C_{id} has already stored the file corresponding to F_{id}. Thus, by construction of our deduplication process, the file upload is not necessary; the DP directly forwards the response to C_{id}.
2) (C_{id}, F_{id}, `File upload`). In this case, the file corresponding to F_{id} does not exist in the storage system, which requires that C_{id} must upload it; the DP directly forwards the response to the client.
3) (C_{id}, F_{id}, `Inter-user deduplication`). The file corresponding to F_{id} has already been stored in the storage system by a client different from C_{id}. Thus, an inter-user deduplication must be applied by the DP. However, in order to hide this inter-user deduplication to the client, instead of directly forwarding this response to C_{id}, the DP sends him/her a `file upload` response, and memorizes that this upload is requested due to an inter-user deduplication.

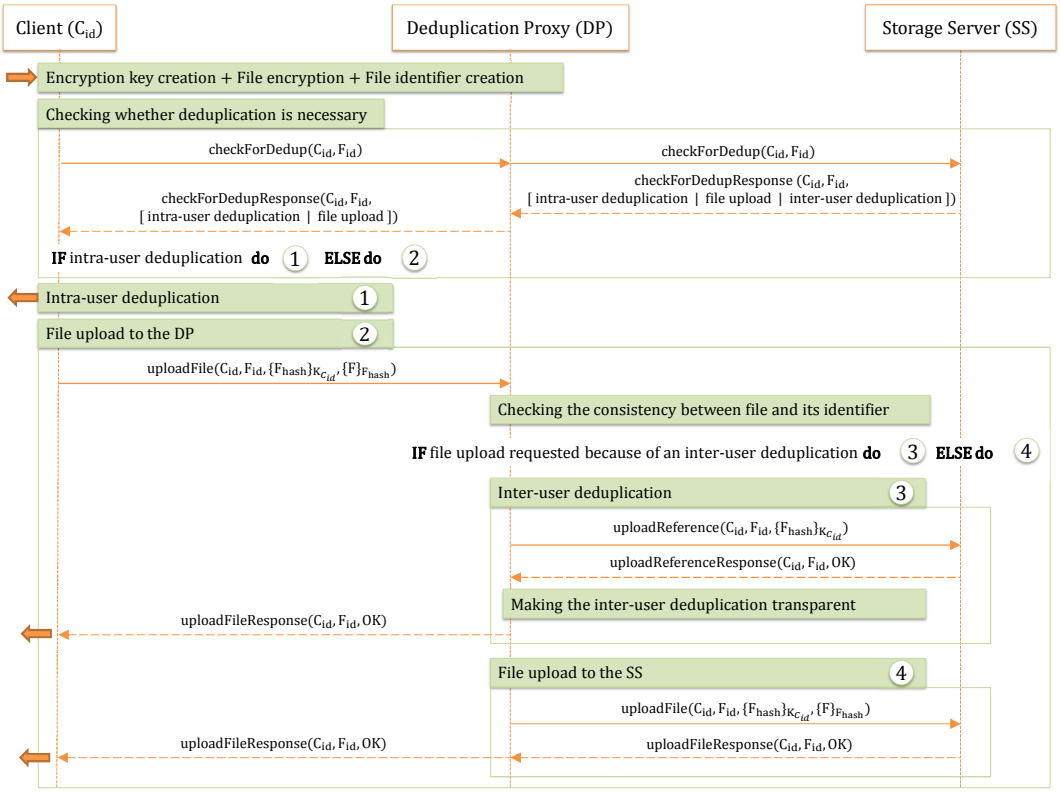

Fig. 2: A `PUT` operation

Upon receipt of the DP message in response to a `checkForDedup` request, client C_{id} behaves as follows:

1) Receipt of an `Intra-user deduplication` response. It ends the `put` operation.
2) Receipt of a `File upload` response. The client uploads to the DP his/her client identifier, the encrypted file $\{F\}_{F_{hash}}$, the file identifier F_{id}, and the decryption key encrypted with its public key as follows,

$$\{F_{hash}\}_{K_{C_{id}}} \leftarrow encrypt_{asym}(K_{C_{id}}, F_{hash}),$$

which guarantees that $\{F_{hash}\}_{K_{C_{id}}}$ can only be decrypted using C_{id} private key. Client C_{id} waits for the response of the DP.

When the DP receives the `uploadFile` message from C_{id}, it applies a consistency check in order to prevent target-collision attacks. A targeted-collision is detected when the hash value of the encrypted file is not equal to the identifier received. In that case, the operation is aborted, and a notification is sent via an `uploadFileResponse` to client C_{id}. This ends the `put` operation. If the consistency check is successful and the file upload has been requested because of an inter-user deduplication, then only a reference (*i.e.* the client identifier C_{id}, the file identifier F_{id}, and the encrypted decryption key $\{F_{hash}\}_{K_{C_{id}}}$) is uploaded to the SS to be stored. Recall that the DP has to ensure that this inter-user deduplication is unnoticeable to the client. Thus, before sending the notification (*i.e.* an `uploadFileResponse` message) to C_{id}, the DP delays it to make the duration of the whole `put` operation similar to a transmission of the file to the SS. The added delay is thus a function of the size of the file and the state of the network. On the other hand, if the file upload has been requested because that file does not exist in the storage system, then the `uploadFile` message is simply forwarded to the SS in the case where the consistency check has been successfully passed.

B. The `get` operation

The `get` operation is invoked by clients to retrieve their own files from the storage system. Figure 3 illustrates the process of a `get` operation. Our deduplication scheme pushes all the complexity to the `put` operation. In `get` operations, the DPs only forward the requests and responses they respectively receives from the clients and the SS. Specifically, upon receipt of a `get` request from a client C_{id} about a file of identifier F_{id}, the DP simply forwards it to the SS which looks for F_{id} in its file ownerships index. If F_{id} is not found among the identifiers of files belonging to C_{id}, the SS sends a `getResponse` message to the DP with an error notification to terminate the `get` request. Otherwise, the SS sends a `getResponse` containing the ciphertext $\{F\}_{F_{hash}}$ corresponding to the identifier F_{id} and the encrypted decryption key $\{F_{hash}\}_{K_{C_{id}}}$ corresponding to the client C_{id}. In both cases, the DP only forwards the `getResponse` to the client C_{id} who will apply an asymmetric decryption function on the encrypted key using its private key $K^{-1}_{C_{id}}$ to

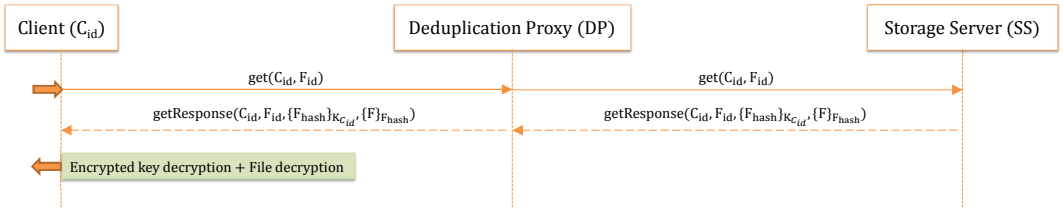

Fig. 3: A `GET` operation.

recover key F_{hash}, where

$$F_{hash} \leftarrow decrypt_{asym}(K_{C_{id}}^{-1}, \{F_{hash}\}_{K_{C_{id}}}).$$

Finally, the plaintext of F is obtained with

$$F \leftarrow decrypt_{sym}(F_{hash}, \{F\}_{F_{hash}}).$$

IV. Implementation

We have implemented two storage system prototypes to compare the performance overhead of our proposition with respect to a classic storage system with no data encryption. Specifically, we have developed a classic storage system with a client and a storage server software modules, and one implementing our proposition with a client, a deduplication proxy and a storage server software modules. All these software modules are implemented in python 2.7.6 and access the pycrypto library for the cryptographic operations. We use the SHA256 algorithm as the hash function, RSA1024 for asymmetric encryption operations, and AES for the symmetric encryption operations with keys that are 256 bits long. The storage servers use the MongoDB[3] database to store the meta-data of the stored files as well as files owners. The software modules are executed on three different virtual machines (VMs) running on Ubuntu 12.04.4 LTS with 1GB of memory and an AMD Opteron(TM) octa-core Processor 6220@3GHz. The network topology is as follows: the VM executing the DP software is located on the network path between the VM running the different clients modules and the one executing the different SSs modules.

Figure 4 and Figure 5 compare the different costs induced by the `put` and `get` operations in a classic storage system and in the one we propose. Each result regarding both `put` and `get` operations is the average of 1000 experiments run with files whose sizes range from 2MB to 64MB. The first straightforward observation that can be drawn from both figures is that the cost of both operations fully depends on the size of the stored files. In `put` operations, the overhead in our scheme is due to the encryption key creation, the encryption key encryption[4], the file encryption, the consistency check of a file and its identifier performed by the DP module, and the communication overhead introduced by handling the file by the DP. On the other hand, for `get` operations, the overhead in our proposition comes from the encryption key decryption[4], the file decryption, and the communication overhead of the DP. However, this overhead is small, as it entails around 0.5s on average per file of 64MB length.

[3]http://www.mongodb.org/

[4]The encryption key encryption/decryption is negligible (around 0.75 ms) and is not visible on Figures 4-6 because of the figure scale.

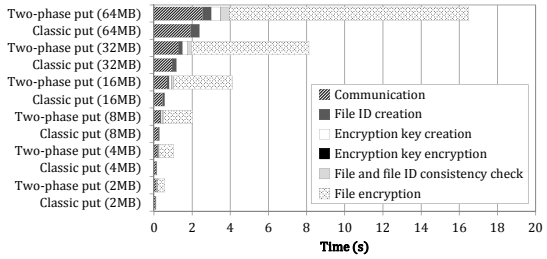

Fig. 4: Average delays observed during a `put` operation in a classic system with no data encryption and in our system with data encryption. Most of the overhead introduced by our solution is due to cryptographic operations which are necessary to ensure data confidentiality.

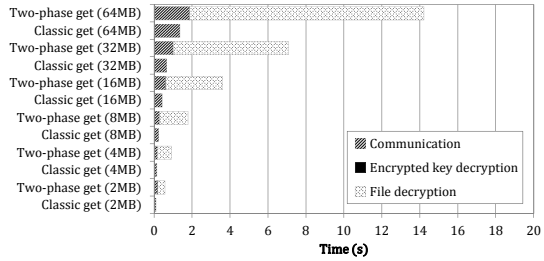

Fig. 5: Average delays observed during a `get` operation in a classic system with no data encryption and in our system with data encryption. Most of the overhead introduced by our solution is due to cryptographic operations which are necessary to ensure data confidentiality.

Actually, most of the overhead incurred in our solution is due to file encryption during a `put` operation and file decryption during a `get` operation. Indeed, up to 12-13s are needed on average for encrypting/decrypting a file that is 64MB length.

Figure 6 gives a performance comparison between an inter-user deduplication applied in a classic scheme and the intra- and inter-user deduplication applied in our proposition. The inter-user deduplication experiments consist in storing files at a given client and having one thousand clients accessing them through the `put` operation. The intra-user deduplication experiment consists in storing thousand times the same files at a single client. The results plotted in Figure 6 represent the average duration values of the these operations. We observe that the intra-user deduplication in our scheme consumes more communication resources than an inter-user deduplication in a classic scheme due to the use of a DP (around 18ms of overhead for a file of 64MB length). However most of the overhead

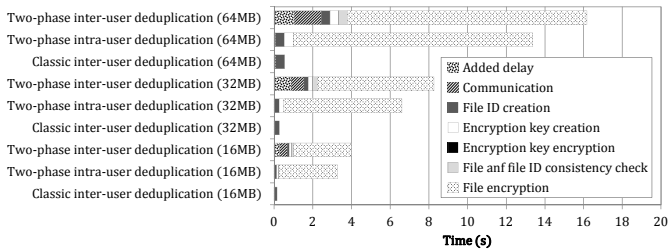

Fig. 6: Performance comparison between a classic system with no data encryption where an inter-user deduplication is applied upon a `put` operation and our proposition with data encryption where an intra- and inter-user deduplication are applied. Most of the overhead introduced by our solution is due to cryptographic operations which are necessary to ensure data confidentiality.

comes from the encryption key creation, the encryption key encryption[4] and the file encryption which depends on the file size. We also observe that operations involving the inter-user deduplication in our scheme have higher durations than the ones involving an intra-user deduplication. This is due to *(i)* the overhead of communications when applying the inter-user deduplication because the file has to be uploaded to the DP and then a reference is uploaded to the the SS and *(ii)* the added delay by the DP to make unnoticeable the inter-user deduplication to client. Actually in our scheme, a `put` operation involving an inter-user deduplication has a similar duration than a file upload from the client to the SS.

V. Discussion

In this section, we discuss the impact of the proposed solution regarding the data security and the network bandwidth savings.

As previously described our solution aims at protecting the storage system against the manipulation of data identifiers, network traffic observation and backup time observation (See Section I). This is achieved by placing deduplication proxies between clients and the storage server, and by running a two-phase deduplication scheme. Now, by taking advantage of the intra-user deduplication performed between clients and their DP, from a client point of view, the visibility of what is stored in the storage system is limited to his/her own account. If a client has not previously stored its file then he/she must upload it to his/her associated DP, which protects the system against the CDN attacks and the attacks based on network traffic monitoring. In order to prevent the targeted-collision attacks, the DP performs a consistency control on both the file and the identifier received from a client before sending this file to the storage server or before applying the inter-user deduplication. To make the inter-user deduplication unnoticeable to clients in order to face the attacks based on backup time observation, the DP adds some delay before terminating the `put` operation (*i.e.* before sending the `uploadFileResponse`). Recall that a delay may be added only when an inter-user deduplication is applied. In addition, by using a convergent encryption scheme, the CSP can perform an inter-user deduplication on client data without any access to their plaintext and achieve global storage space savings.

A. Impact of the localization of the deduplication proxy

The benefits of our proposition regarding the network bandwidth savings highly depends on the localization of the DPs in the storage infrastructure which may also impact the level of security of the solution. In the following, we briefly discuss the pros and cons of the DPs localization.

Deduplication proxies collocated with clients devices: If deduplication proxies are hosted by clients, then both intra- and inter-user deduplication need to be applied at the client sides, which provides global network bandwidth savings. However, this gain comes at the expense of data security. Indeed, clients may observe the network traffic between the DP and the SS, or have the opportunity to corrupt or replace the DP software. Such a solution protects efficiently the storage system solely against attacks based on backup time observation.

A deduplication proxy collocated with the storage server: If the DP is physically collocated with the storage server, then even if clients do not upload more than once the same file to that location thanks to the intra-user deduplication, duplicates of a file owned by other clients will still have to be transferred over the network to reach that location before the inter-user deduplication can be applied. This is clearly not the most efficient choice in terms of bandwidth savings. On the other hand, data security is not affected by this solution.

Deduplication proxies hosted by independent nodes: We suggest that the two-phase deduplication relies on independent nodes to host the DPs. We argue that such nodes should be secure, physically out of reach of clients so that they can not observe the network traffic between a DP and the SS, but at the same time close enough to clients in order to improve the network bandwidth savings. We believe that to locate such nodes one could adopt a similar strategy to what is done in the field of Content Delivery Network (CDN) [21]. In order to minimize data access latencies and network bandwidth consumption toward content servers, cache servers are deployed in strategic locations in the Internet infrastructure as close as possible to the clients to satisfy their requests close to them [22]. For instance, an interesting approach would be to use the Points of Presence (POPs) of the ISP as secure nodes to host the DPs and associate them to clients accessing to Internet through them [23]. This might also allow CSPs to cache files in the DPs in order to minimize the access latencies of subsequent `get` operations on these files.

B. Limitation of allowing the inter-user deduplication

In this section we discuss the potential danger of a true inter-user deduplication regarding data confidentiality, and provide insights to deal with them. Allowing the inter-user deduplication on encrypted files gives the opportunity to determine that two equal ciphertexts originate from the same plaintext. This can be exploited by the CSP since given a file or just its identifier, the CSP can determine whether this file already exists in the storage system and identify the clients who have uploaded it. Moreover the CSP is able to perform brute force attacks on file contents even if their files are encrypted with a convergent encryption [8], [15]. Recall that in our proposition, the inter-user deduplication is unnoticeable to clients so these limitations are only applicable to the CSP. Nevertheless, one may find in such an approach an interesting

property for the CSP to detect illegal files in its storage system or/and to respond to judicial inquiries for example.

Countermeasure: A strategy to protect files against such honest but curious CSP would be to encrypt files using private encryption schemes. For example, individuals may use personal keys in the file encryption process. In the case of a group of clients or an enterprise, a key server that does not belong to the CSP can be used to manage the encryption keys so that all the clients of the key server will encrypt a given file with the same specific key [8], [13]. However it will make possible the inter-user deduplication solely within the members of the group or the enterprise and would reduce the storage space savings of the CSP since a given file encrypted with different keys would result in different ciphertexts. A data deduplication process independent of the owners (*i.e.* individuals, groups or enterprises using different key servers) would be impossible.

VI. Related work

To deal with the security issues introduced by a client-side and inter-user deduplication, a method called randomized solution is proposed in [5]. This solution defines a random threshold for each file in the storage system which corresponds to the number of different clients that have uploaded this file. Each time a client uploads for the first time a file, a counter associated with this file is created and initialized to 1. As long as the counter is under the threshold, deduplication is not applied. When the counter reaches the threshold, the inter-user deduplication is activated and applied to all the subsequent uploads of that file. This solution protects a file as long as its threshold is not reached. As soon as the system enables deduplication on a file, it becomes vulnerable. In [7], the authors propose a deduplication method that is secure against targeted-collision attacks. Similarly to our method, a file identifier is a hash value of the file content so it is simple to check the consistency of a file and its claimed identifier. However, this method does not address the others types of attacks mentioned in Section I. In [12], the authors propose a gateway-based deduplication scheme which relies on users gateways to run the deduplication process. In order to mitigate the attacks based on network traffic monitoring, gateways send a random network traffic compounded of encrypted packets with a Time-To-Live (TTL) parameter equal to 1. On the other hand, this method does not take into account neither backup time observation attacks, nor the possible corruption of the deduplication software module by the gateway owner. Several additional works suggest the notion of Proof Of Ownership (POW) [11], [4], [6], [14], [16] to face the attacks based on manipulation of data identifiers. A proof of ownership allows a client to prove that he/she actually owns the file corresponding to the identifier presented to the storage server. However, it does not address the issues caused by attacks based on network traffic or backup time observation that are performed by adversaries that actually own the files corresponding to the identifiers that they present to the storage server. Some works rely on specific encryption schemes to provide data confidentiality and deduplication against honest but curious CSPs [8], [10], [13], [15]. Table. I shows a comparison of some related work and our proposition regarding the storage space and network bandwidth savings, and the protection against the attacks mentionned in Section I. Compared to all the previously cited works, our two-phase deduplication method addresses all the attacks presented in Section I, and to the best of our knowledge this is the first one to achieve it.

VII. Conclusion and future work

In this paper, we have presented a two-phase deduplication scheme that *(i)* ensures that a client actually owns the file he/she wants to store by applying an intra-user deduplication on the client side *(ii)* ensures that a file corresponds to its claimed identifier through a control by a deduplication proxy located between clients and the storage server and *(iii)* applies an inter-user deduplication on the deduplication proxy side that makes this inter-user deduplication unnoticeable to clients by adding some delay to `put` operations so that the length of a file upload is indistinguishable from an upload of its reference. Our method provides protection against attacks from malicious clients, global storage space savings to the CSPs thanks to the inter-user deduplication, per-client bandwidth network savings between clients and the deduplication proxies, and global network bandwidth savings between the deduplication proxies and the storage server.

For future works, we plan to address the confidentiality issues against attacks that can be performed by the CSP. We also plan to extend our solution so that encrypted decryption keys can also be deduplicated without jeopardizing security properties. We will also consider how to extend the deduplication in our scheme to a block level granularity.

References

[1] M. Dutch, "Understanding data deduplication ratios," *SNIA Data Management Forum*, 2008.

[2] D. Russel, "Data deduplication will be even bigger in 2010," *Gartner*, February 2010.

[3] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST)*, 2011.

[4] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," in *Proceedings of the 18th ACM Conference on Computer and Communications security (CCS)*, 2011.

[5] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security Privacy*, vol. 8, 2010.

[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," in *Proceedings of the 20th USENIX Conference on Security (SEC)*, 2011.

[7] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure data deduplication," in *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability (StorageSS)*, 2008.

[8] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," in *Proceedings of the 22nd USENIX Conference on Security (SEC)*, 2013.

[9] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Memory deduplication as a threat to the guest os," in *Proceedings of the 4th ACM European Workshop on System Security (EUROSEC)*, 2011.

[10] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.

[11] R. Di Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for deduplication," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2012.

TABLE I: A comparison between some related works and our proposition.

Approaches	Storage space savings	Network bandwidth savings	Protection against clients attacks based on data identifier manipulation	Protection against clients attacks based on network traffic observation	Protection against clients attacks based on backup time observation	Protection against honest but curious CSP
Harnik *et al.* [5]	Global savings	Global savings	No more protection when the inter-user deduplication is enabled	No more protection when the inter-user deduplication is enabled	No more protection when the inter-user deduplication is enabled	NO
Storer *et al.* [7]	Global savings	Global savings	Protection only against targeted collision attacks	NO	NO	NO
Proof of ownership [11], [4], [14], [16]	Global savings	Global savings	YES	NO	NO	NO
Heen *et al.* [12]	Global savings	No bandwidth savings between the client and the gateway but global bandwidth savings between the client and the storage server	Limited to the ability of attackers to monitor the traffic going through the gateway	Limited to the ability of attackers to corrupt or replace the software module in the gateway	NO	NO
Bellare *et al.* [8]	Global savings only for the group of users using their client solution	Global savings only for the group of users using their client solution	NO	Online brute force attacks are slowed using a requests rate limitation	Online brute force attacks are slowed using a requests rate limitation	YES
Xu *et al.* [15]	Global savings	Global savings	YES	NO	NO	YES
Liu *et al.* [13]	Global savings only for the group of users using their client solution	Global savings only for the group of users using their client solution	NO	NO	NO	YES
Our proposition with DP on the client	Global savings	Global savings	NO	NO	YES	NO
Our proposition with DP on the storage server	Global savings	Per-client savings	YES	YES	YES	NO
Our proposition with DP on an independent node	Global savings	Per-client savings between clients and the DP, and global savings between the DP and the storage server	YES	YES	YES	NO

[12] O. Heen, C. Neumann, L. Montalvo, and S. Defrance, "Improving the resistance to side-channel attacks on cloud storage services," in *Proceedings of the 5th International Conference on New Technologies, Mobility and Security (NTMS)*, 2012.

[13] C. Liu, X. Liu, and L. Wan, "Policy-based de-duplication in secure cloud storage," in *Trustworthy Computing and Services*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2013, vol. 320, pp. 250–262.

[14] W. K. Ng, Y. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC)*, 2012.

[15] J. Xu, E.-C. Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIACCS)*, 2013.

[16] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," in *Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012.

[17] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proceedings of the 14th International Conference on Financial Cryptography and Data Security (FC)*, 2010.

[18] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, vol. 16, 2012.

[19] F. Rocha, S. Abreu, and M. Correia, "The final frontier: Confidentiality and privacy in the cloud," *Computer*, vol. 44, no. 9, pp. 44–50, 2011.

[20] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[21] G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks," *Communications of the ACM*, vol. 49, no. 1, pp. 101–106, 2006.

[22] C. Huang, A. Wang, J. Li, and K. W. Ross, "Understanding hybrid CDN-P2P: why Limelight needs its own Red Swoosh," in *Proceedings of the 18th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2008.

[23] P. Meye, P. Raïpin, F. Tronel, and E. Anceaume, "Toward a distributed storage system leveraging the DSL infrastructure of an ISP," in *Proceedings of the 11th IEEE Consumer Communications and Networking Conference (CCNC)*, 2014.