An Evaluation of Recent Secure Deduplication Proposals

Vladimir Rabotka and Mohammad Mannan*

Abstract

Deduplication is widely used by cloud storage providers to cut costs, by storing and uploading a single instance of identical files shared across multiple user accounts. However, cross-account deduplication introduces several new side-channel attacks on user privacy; see e.g., Harnik et al. (IEEE Security and Privacy Magazine, 2010), Mulazzani et al. (USENIX Security, 2011). As a response, several solutions have been proposed to mitigate different deduplication privacy concerns. In this paper, we summarize notable attacks on deduplication, and analyze recently proposed privacy-preserving secure deduplication solutions in terms of privacy-gain, deployment and bandwidth costs, and security limitations (if any). In particular, we identify weaknesses in a secure deduplication proposal based on the use of a home gateway device (Heen et al., New Technologies, Mobility and Security, 2012); we also explore how these weaknesses may lead to three separate attacks. Overall, our analysis may help storage providers to evaluate competing solutions, and the research community to better design privacy-preserving deduplication solutions by addressing limitations of current proposals.

Keywords: Cloud storage, privacy, deduplication

*Corresponding author. E-mail addresses: v_rabotk@encs.concordia.ca (V. Rabotka), m.mannan@concordia.ca (M. Mannan)

An Evaluation of Recent Secure Deduplication Proposals

Vladimir Rabotka and Mohammad Mannan*

1. Introduction

Popular cloud storage services boast users in the millions, with gigabytes of free storage offered to each user. To leverage common files shared across user accounts, several cloud storage services use data deduplication. Deduplication eliminates the need to upload and store redundant copies of user data, by verifying before each upload if a file (or, more generally, a data block) already exists in the server's storage. If so, the file is not uploaded and the corresponding user account is simply linked to the existing file on the server. Data deduplication is believed to save significant storage and bandwidth costs.

For example, a recent empirical study [29] on 857 desktop computers reports that with deduplication, the storage requirement is only about 32% of the original storage size (see also Harnik et al. [20] for an efficient estimation of deduplication ratios).

Serious privacy concerns may arise when deduplication is used by popular storage services. Harnik et al. [21] explore several side-channel attacks; for example, the presence of a specific file in the cloud can be verified (by observing network traffic), and linked to a specific user, e.g., by having access to a file that uniquely identifies a target user. An attacker can also fill out a template file with specific details of a target victim (e.g., salary in an employment contract or diagnosis in a medical record template), and infer the existence of such a file in the cloud. Mulazzani et al. [30] demonstrate several attacks on Dropbox due to the use of deduplication (see also [40]). For example, an attacker can obtain access to an existing file, simply by supplying the hash of the file (i.e., without possessing the content of the file); the original

*Corresponding author. E-mail addresses: v_rabotk@encs.concordia.ca (V. Rabotka), m.mannan@concordia.ca (M. Mannan)

Preprint submitted to Elsevier Journal of Information Security and Applications, October 19, 2015

owner of the file remains oblivious to the attack. Halevi et al. [19] provide example scenarios in which hash values of sensitive files may be exposed and the file content accessed by attackers.

To counter known attacks, several academic solutions have been proposed over the past few years. Most proposals can be placed into one of four categories: (i) encryption-based solutions; (ii) probabilistic uploads; (iii) proof-of-ownership schemes; and (iv) gateway-based solutions. In this paper, we analyze representative solutions from each category in terms of their effectiveness (i.e., privacy-gain), deployment and operational costs, and security weaknesses (if any).

Client-side encryption of user data seems to be an obvious solution to several deduplication attacks. However, a straightforward use of encryption can eliminate advantages of deduplication and incur a high penalty in storage and bandwidth consumption for cloud providers (e.g., the same file will generate a different ciphertext for each user who uploads it). Several proposed solutions (e.g., [13, 34, 27],) aim to provide confidentiality against a storage provider, and still allow for deduplication.

Probabilistic upload-based solutions (e.g., [21, 19, 26]) attempt to improve user privacy by requiring additional uploads (e.g., randomly requesting uploads for an already uploaded file). The primary goal is to confuse an attacker about whether a target file exists in the cloud or not. However, these solutions do not offer strong deployment incentives for service providers, as they significantly increase bandwidth costs due to the extra uploads.

Using proof of ownership (PoW) [7] schemes, a server can verify that a user is in full possession of a file without the need of a full upload, before linking the server copy to the user's account. Several PoW-based schemes have been proposed in the recent past, focusing primarily on efficiency gains (e.g., [52, 37, 19, 50, 18, 12]). Other deduplication goals, such as proof of data possession (PoD) [2, 47], and proof of retrievability (PoR) [5, 25, 33] have also been proposed. We do not address PoD and PoR solutions here, since we consider scenarios of curious-but-honest cloud storage providers.

Heen et al. [22] propose a deployment-friendly, home gateway-based solution to address privacy attacks due to deduplication. It is assumed that the user's network service provider (NSP) is also the cloud storage provider. The home gateway device is deployed by the NSP. As stated [22], in some countries (e.g., UK, France), NSPs already offer cloud storage services, which may favor home gateway-based solutions. Our analysis of Heen et al.'s proposal identifies several potential weaknesses that may be exploited to launch

known side-channel attacks. We also attempt to fix these weaknesses, and analyze our proposed counter-measures.

The remainder of this paper is organized as follows. In Section 2 we provide the necessary background on deduplication and review known deduplication attacks. After analyzing current mitigation attempts based on encryption (Section 3), probabilistic uploads (Section 4) and proof of ownership schemes (Section 5), we discuss the home gateway-based solution proposed by Heen et al. (Section 6), presenting three attacks as well as possible ways of mitigating the attacks. A storage-gateway solution [38], which offers differential privacy is analyzed in Section 8. Existing solutions are compared in Section 9 in terms of security, deployment costs and target environments.

2. Background and known attacks

In this section, we briefly introduce the concept of deduplication, and discuss currently known attacks exploiting deduplication as used by cloud storage providers (CSPs). We later use these attacks to evaluate different privacy-enhanced deduplication proposals.

2.1. Deduplication

Deduplication eliminates duplicate copies of redundant data from a CSP. Data is stored and/or transferred only once. Subsequent copies are replaced by pointers to the one physical data instance. Deduplication approaches vary according to organizational needs. For instance, in *server-side deduplication*, data is always uploaded from the client to the CSP, but only one copy is stored on the server. This approach saves storage space but not bandwidth. In *client-side deduplication*, when a client wishes to upload a file to the cloud, a unique representation of the file (e.g., cryptographic hash) is sent to the storage provider. This unique representation is much smaller than the file itself and acts as a fingerprint for the file. If the file is already present in the cloud (e.g., identical hash), the file is linked to the client's account without performing an actual upload. This approach saves both storage space and network bandwidth.

By observing the amount of upload traffic and comparing it to the file size, an attacker can learn if a file was deduplicated (e.g., only the hash was transferred) or not (a full upload was performed). An attacker will measure the traffic between the CSP and her own machine and thereby identify if a particular file is present or absent in the cloud. This fact can be exploited when client-side deduplication is performed over different user accounts, and

the physical copy of a file is shared across different and otherwise unrelated accounts.[1] Side-channel attacks on deduplication require only a valid account with the same CSP as the victim. No further traffic analysis is necessary (e.g., encrypting data in transit does not prevent or mitigate deduplication attacks).

2.2. Side Channel Attacks On Deduplication

Harnik et al. [21] propose three side channel attacks on user privacy by exploiting cross-user deduplication. We summarize the attacks below.

1. A single file can uniquely identify a user, exposing the user's identity. For instance, an organization can set up a trap, where different versions of a sensitive document are made available to several suspects, and observe which version gets uploaded to the cloud; cf. a recently proposed e-book DRM [49], and a related IBM patent to detect email leaks [9]. The attacker will try to upload the file which specifically identifies the target user, and observe the network traffic to see whether deduplication takes place. This attack requires the attacker to possess the file in question (or at least the fingerprint of the file), and no additional information about the victim is revealed. However, this attack may still be leveraged in certain cases.
2. An attacker can fill a template file with several details specific to the victim (e.g., salary in an employment contract or diagnosis in a medical record template), and infer the existence of such a file in the cloud. For example, if the attacker has a copy of a template file for a contract renewal offer (e.g., by working at the same company as the victim), the attacker can fill out the template with the victim's name and various monetary values, e.g., in $500 increments. By observing when deduplication takes place, the attacker can establish which version of the file exists in the cloud and thus infer the victim's contractual offer. This is a threat to user privacy since users and organizations alike store and back up an increasing number of sensitive files to the cloud. [41, 4, 51]
3. The deduplication feature can be abused to maintain a covert channel to communicate with a command and control server (C&C). Malicious software can store one of two versions of a single file in the cloud. The

[1] Note that, for privacy reasons, not all CSPs use cross-user data deduplication; see, e.g., SpiderOak [17] (but see also [48]).

C&C server then uploads both versions to the cloud. By observing for which file deduplication takes place, the C&C server will receive one bit of information from the victim's machine (e.g., bot software was installed successfully). Likewise, the C&C server can abuse client-side deduplication to send one bit of information to the bot client (e.g., start a DDoS attack). By increasing the number of files, multiple bits of information can be sent and received.

The above-listed side-channel attacks implicitly assume no correlation between files stored by the CSP. This assumption may not always hold, as pointed out by Shin et al. [38], who introduce the related-files attack. In real-world scenarios, correlated files such as individual files that make up a software package, or different formats of the same office document are often uploaded together and stored at the same CSP. An attacker can amplify side-channel attacks by uploading not only the targeted file, but also additional files related to it, increasing the probability with which deduplication can be inferred, and thereby weakening the effectiveness of probabilistic upload solutions (see Section 4).

Harnik et al. [21] also assume the attacker playing only the role of a valid user. However, side channel attacks can also be carried out by the CSP itself. For instance, the CSP can compute the hash for a copyright-protected file and then search its hash index table for clients who have the same hash value linked to their accounts. Such identification becomes a valid threat to privacy, when users rely on deduplication-friendly encryption for data confidentiality. We distinguish between server and client side channel attacks in our final comparison (Section 9), where a server side-channel attack represents any method by which a CSP can obtain access to a user's unencrypted data or infer a connection between an unencrypted file (or its hash) and users who have versions of the said file linked the their account (e.g., encrypted, encoded or plaintext version).

2.3. Hash-Only Attacks On Deduplication

Deduplication can also be abused by exploiting the connection between a file (or data block) and its corresponding hash value. A file will be deduplicated if its unique representation (e.g., hash) matches the representation of a file already present in the cloud. This means that an attacker can deduplicate files to his account by only presenting their hash values. The corresponding file will be linked to the attacker's account, without the consent or knowl-

edge of the file's owner. By employing this technique, an attacker can use the cloud storage provider as a content distribution system. A C&C server can also maintain a covert channel to communicate with bots just by sharing hash values.

Mulazzani et al. [30] show how Dropbox's cross-user deduplication can be exploited to access arbitrary files, given their hash values. An open source project called Dropship [44] (currently defunct), exploited Dropbox's file hashing scheme, allowing users to copy to their accounts files they did not previously posses. Hash values can be distributed through a secondary channel, such as forum posts.

Xu et al. [50] introduce the poison hash attack, also called the target collision attack. The adversary controls the upload client and is therefore able to subvert the default hash computation, allowing her to supply a hash value that corresponds to a different file. This allows the attacker to upload an infected or forged file, under the hash value of the corresponding clean version. If the attacker is the first user to upload the file (e.g., newly released freeware or Linux ISO files), all subsequent uploads of the clean file will be deduplicated to the attacker's version of the file. The poison hash attack presumes the adversary is able to reverse engineer the upload client, as performed by Mulazzani et al. for Dropbox [30].

The covert-channel attack described in Section 2.2 can be amplified when combined with a hash-only attack, in order to leak larger amounts of data [19]: malicious software on a client computer can leak to an adversary the hash values of all files that are uploaded to a CSP. The hash leak is low bandwidth and would enable the attacker to download the corresponding files through a hash-only attack on the same CSP.

3. Encryption Solutions

Users may avoid deduplication attacks by uploading only encrypted versions of their sensitive files. Traditional encryption appears to be inherently incompatible with data deduplication. If different users encrypt the same file with different keys, the resulting ciphertexts will be different, making it impossible for the CSP to deduplicate different uploads of the same file. Several third-party tools for achieving such encryption support are available today (e.g., BoxCryptor [6]).[2]

[2] When using a user-specific and password-derived encryption key (e.g., BoxCryptor [6]), we believe that a brute-force attack can be launched by exploiting deduplication:

Where to encrypt. Encryption can occur at the client side or in the cloud. If user data is encrypted only after it reaches the cloud, the provider can read the data, be forced to hand over the data to an external party or leak data through insider attacks. Thus, third parties may still obtain access to the user's unencrypted data without the data owner having any knowledge of the privacy breach.

If encryption occurs at the client side, and only the user is in possession of the encryption key, the cloud provider can no longer gain any insight about the data content. The main deterrent for service providers to introduce client-side encryption is the inability to perform deduplication on the uploaded user data. While deduplication provides significant savings in both storage space and bandwidth, client-side encryption ensures user data privacy but offers no immediate incentive to CSPs, especially the ones that provide free storage for public use. Another challenge that comes with client-side encryption is key management across different devices, as well as between users who wish to share their files with one another.

As only few CSPs offer client-side encryption (e.g., Tarsnap [43]), several third-party add-ons have been developed to enable encryption for popular unencrypted storage providers; such add-ons include: Safebox [32], BoxCryptor [23], Viivo [45]. BoxCryptor and Viivo derive the encryption key from (low-entropy) user passwords, whereas Safebox uses the actual password to encrypt files. Note that password-derived keys are inherently unsafe for encryption, as user-chosen passwords can be easily brute-forced. Another solution, DFCloud [36] proposes to leverage the widely-available Trusted Platform Module (TPM) to manage encryption keys and define a key sharing protocol between users. While requiring an additional server, this solution prevents not only server-side data leakage, but also the leakage of client-side credentials by malicious programs. Deduplication however is not possible.

Proposals explored in this section seek to use client-side encryption yet allow for deduplication. The goal is to protect user data from a curious

If a popular file is encrypted with the user's password-derived key, an attacker can guess a password, derive the key, encrypt the plaintext version of the file, upload the encrypted version, and observe if deduplication occurs. Deduplication indicates that the encrypted version of the file is present in the cloud, which means the password guessed by the attacker was correct and thus the user's password is exposed, without the victim even being aware that an attack took place. However, this attack may be less of a concern in reality as it can be easily mitigated by choosing encryption modes that use a random IV.

CSP, while still allowing the CSP to save storage space by deduplicating data across multiple user accounts. By design, most encryption solutions focus on server-side deduplication and are therefore not susceptible to client side-channel attacks (Section 2.2).

3.1. Bit-Interleaving File System (BIFS)

One proposal which tries to find a compromise between encryption and deduplication is BIFS [34]. Instead of encrypting data, BIFS re-orders but does not substitute bits. BIFS divides a block of user data into slices of variable size. Slices are stored in a number of randomly selected chunks, with each chunk saved in a different location of the cloud storage. A chunk stores slices from multiple files and users. The pseudo-random slicing and mixing of slices from different files in the same chunk makes it difficult to recover the original file without the block descriptor, available only to the file owner.

By spreading user-data bits over several locations of the storage infrastructure, BIFS ensures that private user data cannot be identified, in practice, by unauthorized users, even if all bits and sectors on the storage media are visible. Therefore, on-disk data is protected from unauthorized access by the infrastructure provider (e.g., compromised server, malicious staff). As bits are only reordered and not substituted, data regularity is still preserved to some degree, allowing for bit-level compression and deduplication.

While it seems that meaningful data is unrecoverable from the CSP and that data cannot be linked back to the user, this is not entirely true. The proposed BIFS solution apparently ignores the cases of self-identifying data. Since bits are not replaced, meaningful information may still be retrievable from the slices. The provider can launch pattern matching attacks, searching data chunks for sensitive information such as credit card numbers, email addresses, monetary figures, PIN numbers, which are all smaller than the slice size and likely to be stored next to personally identifiable information (a slice range of 4096 to 5120 bits is used in the BIFS proof-of-concept implementation). A slice size smaller than 20 bytes, which is more than a typical email address or credit card number, makes deduplication not worthwhile. Larger slice sizes expose larger contiguous portions of user data. Data privacy must be sacrificed for significant deduplication savings. We do not include BIFS in our final comparison.

3.2. Convergent Encryption

Convergent encryption [13, 46] proposes a framework which maintains data privacy in the cloud, while still allowing the CSP to perform deduplication. Instead of using a random and user-specific key, the convergent encryption algorithm generates the encryption key from the file content itself, usually the hash of the plaintext file. This guarantees that the same plaintext will generate the same ciphertext, allowing the cloud storage provider to take advantage of deduplication while only having access to encrypted user data.

Convergent encryption can also be applied on directory trees [1] in addition to singular files: assuming UNIX-based directories (which contain the name of all files and subfolders), the hash of the directory object acts as a unique identifier for the whole subtree. If a directory hash is already in the cloud, there is no need to traverse the local directory tree and the whole tree is deduplicated. A user only needs to record the keys for each root filesystem that is already present in the cloud, reducing the number of files to hash.

Since the encryption key is created deterministically from the file content, a CSP can apply the convergent encryption algorithm for popular files (e.g., public, leaked or pirated files) and determine by which users they were uploaded. The CSP can also launch brute-force ciphertext recovery attacks on any uploaded file.

Convergent encryption in itself does not address side-channel attacks, which are feasible if convergent encryption is used in conjunction with client-side deduplication. In such a scenario, the attacker is able to produce the hash of a convergent encrypted file and use it to determine the existence of said file in the cloud.

Likewise, the hash-only attack and poison hash attack are not addressed by convergent encryption and their feasibility depends on the particular implementation (e.g., client or server-side deduplication).

3.3. DupLESS

Bellare et al. [3] address the shortcomings of convergent encryption by introducing a third-party key server, and replacing convergent encryption with secret-parameter, message-locked encryption (MLE); the organization-specific secret parameter is known only to the key server. Each client belonging to the same organization engages with the key server in an oblivious pseudo-random protocol (OSRP) to obtain a message-derived key, used to encrypt a file before sending it to the CSP. Users under the same key server

will receive the same key for identical files, allowing the CSP to deduplicate the encrypted version. To prevent the key server from learning any information about user data, DupLESS relies on RSA blind signatures [10]. The key server is assumed to be inaccessible to the attacker (e.g., requires authentication credentials unavailable to the attacker). In order to thwart brute-force ciphertext recovery attacks from compromised clients, the key server uses rate limiting, with a rate limit of $825,000$ queries/week/client being suggested as a conservative approach.

From the CSP's perspective, deduplication is possible only for files encrypted by using the same key server. Files cannot be deduplicated across different organizations without extending the DupLESS protocol to allow interactions between separate key servers. It is therefore unclear how the solution would scale to accommodate scenarios where large CSPs seek to employ deduplication across data belonging to disjoint organizations. Using a key server also introduces a single point of failure and compromise (cf. [15]). The overhead introduced by DupLESS causes a 14–17% upload latency, decreasing with larger file sizes.

Client side-channel and hash-only attacks are not possible because DupLESS is a server-side deduplication solution. Since the CSP is not in possession of the secret parameter, server side-channel attacks are also unfeasible.

In a follow-up paper, Duan [15] eliminates the need for a key server by introducing a distributed protocol, to be used by P2P systems. Duan first provides a formal proof of security for the DupLESS protocol in the random oracle model, showing that in the current deduplication paradigm using an additional secret when generating the encryption keys provides the best possible security. Duan then extends the DupLESS protocol to a P2P environment by using a modified Shoup RSA-based threshold signature scheme [39]. A private key is shared across a number of n players such that any subset of $t+1, t<n$ players can generate a signature, but any subset of t or less players are unable to produce a valid signature. When a client needs to upload a file or data block, it will request signature shares from $T \geq t+1$ signers, combine the shares to obtain the signature, and then use the signature to generate the encryption key. A blind signature variant of Shoup's scheme is used to avoid leaking any information about the file/data-block to the signer. The distributed key sharing scheme still assumes a centralized dealer distributing the keys. It is possible to make the scheme fully distributed by using a variant of Shoup's scheme that allows for distributed key generation.

The key retrieval overhead of the P2P DupLESS scheme causes the upload throughput to drop by 4–30%, depending on the data block size [15]. The overhead decreases as files get larger. As is the case for DupLESS, side-channel and hash-only attacks are not feasible.

To hinder deduplication attacks, both DupLESS and Duan's P2P version of DupLESS require that a perimeter be built around deduplication clients, by controlling access to a key server or access to a P2P network. This approach outsources the burden of preventing deduplication attacks to a separate entity (key server or P2P network) but has an inherent shortcoming. By fragmenting the deduplication landscape, CSPs have weaker incentives to store data because the deduplication potential is limited by the number of clients inside each enclave. Such an approach is appropriate for organization-specific data (unlikely to be shared by outsiders), but is not scalable.

3.4. Leakage-Resilient Encryption

Another proposal to offer leakage-resilient encryption, introduced by Xu et al. [50], uses randomly generated keys for each file and yet allows CSPs to identify identical files and employ deduplication. On first upload of a file F, the user chooses a random AES key K and produces two ciphertexts: the file encrypted with the key $\{F\}_K$, and the key encrypted with the file $\{K\}_F$. Both are stored by the CSP. When another user tries to upload the same file F, the server asks for proof of ownership (see Section 5) and then provides the key encrypted with the file. Since the user is in possession of the file F, she will be able to retrieve the encryption key K, and store it for later decryptions. The CSP links the encrypted version of the file to the second user's account.

This approach is still susceptible to both the client and server side-channel attacks. A malicious client can engage in a full protocol run with the server for any file it generates (e.g., by filling out template files) and observe the network traffic. The CSP can launch a successful attack without knowing the encryption keys, by simply observing disk usage information and deduplication tables stored in memory [31]. A malicious CSP can play the role of a client and at the same time observe the server's reaction to client requests. By first uploading random junk data (which is guaranteed to not be deduplicated), the malicious CSP can create a baseline for disk and memory usage of the server's deduplication process when a full upload is performed. By analyzing how disk and memory usage deviate from the established baseline

when uploading a file in question, the CSP can know if it was deduplicated or not. In this way, by simply monitoring the use of its resources, a malicious CSP can launch a successful guessing attack (e.g., the template side-channel attack). Hash-only attacks are mitigated by proof of ownership schemes proposed in the same paper (see Section 5).

3.5. Single-Server Cross-User Deduplication

Liu et al. [27] introduce a single-server cross-user deduplication scheme with client-side encryption, where a client uploading an existent file uses a password authenticated key exchange (PAKE) protocol to obtain the encryption key from another client who has previously uploaded the same file. If the file has not been previously uploaded, the CSP instructs the client to use a random key.

For each uploaded encrypted file F, the CSP stores a short hash $sh(F)$, which has a high collision rate and cannot be used by the CSP to determine the content of F. When a client C wishes to upload file F, the CSP identifies clients C_i that have uploaded files F_i with the same short hash $sh(F_i) = sh(F)$. The short hashes may equal because they were derived from the same file ($F = F_i$), or due to a hash collision. To determine if $F = F_i$, the client C engages in a same-input-PAKE protocol with every candidate client C_i. C's input is $hash(F)$ and C_i's input is $hash(F_i)$. For each client C_i, the same-input-PAKE protocol produces a session key k_i, while C ends up with a set of session keys k'_i, one for each PAKE run with a different C_i.

With an overwhelming probability, $k_i = k'_i$ if and only if $F_i = F$. The same-input-PAKE protocol is run through the CSP, no direct connection between C and C_i is required. At this point, in case of a match, $E(k_i, k_{F_i})$, the encryption key k_{F_i} symmetrically encrypted with the session key k_i, could be transferred from C_i to C. This however would allow a side-channel attack where C infers the absence of file F in the cloud, if it replaces some of the keys k'_i with bogus values and is instructed by the CSP to use the index of one of those keys. To defeat this attack, each client C_i extends the length of k_i by using a pseudo-random function and then splits the result into $k_{iL}||k_{iR}$, sending k_{iL} and $(k_{F_i} + k_{iR})$ to the CSP.

Similarly, client C extends and then splits all k'_i into $k'_{iL}||k'_{iR}$. C then sends its public key pk, k'_{iL}, and $Enc(pk, k'_{iR} + r)$, where $Enc()$ represents homomorphic additive encryption, and r is a random element chosen by C. Upon receiving these messages, the CSP verifies if there is an index j such that $k_{jL} = k'_{jL}$. If such an index j exists, the CSP uses the encryption's

homomorphic properties to compute $e = Enc(pk, k_{F_j} + k_{jR}) \ominus Enc(pk, k'_{jR} + r) = Enc(pk, k_{F_j} - r)$; e is sent back to C. If there is no index j such that $k_{jL} = k'_{jL}$, the CSP sends to C $e = Enc(pk, r')$, where r' is a random element chosen by the CSP.

Upon receiving e, C uses its private key sk to calculate $k_F = Dec(sk, e) + r$, and sends $E(H_1(k_F), F)$ to the CSP. The hash function H_1 is used to hash the length of the key k_F to the length expected by E. The CSP deletes $E(H_1(k_F), F)$, if it is already stored as $E(H_1(k_{F_j}), F_j)$, and then allows C to access $E(H_1(k_{F_j}), F_j)$. If the file is not present in the storage, the CSP stores $E(H_1(k_F), F)$.

By providing client-side encryption, the single-server cross-user deduplication protocol prevents server-side attacks such as the CSP identifying clients who have uploaded a specific file. The file is always uploaded from the client's machine to the CSP, which mitigates side-channel attacks but provides no bandwidth savings. To save bandwidth, Liu et al. expand their algorithm to use the randomized threshold solution proposed by Harnik et al. [21] and covered in Section 4, thereby achieving the same information leakage probability for the same bandwidth cost; see Table 2. Using the randomized threshold approach makes the protocol susceptible to hash-only attacks which can be mitigated through PoW schemes (see Section 5). In the basic (server-side) version of Liu et al.'s protocol, hash-only attacks are implicitly mitigated since the last step of the protocol involves a full upload of the file $E(H_1(k_F), F)$ from the client to the CSP. To fetch the client-side encryption key, (some) previous clients must be online when the protocol is run. Both versions of Liu et al.'s protocol, original and randomized threshold extension, are included in our final comparison.

4. Probabilistic Upload-based Solutions

As discussed in Section 2.2, by uploading a file and observing the network traffic, an attacker can establish if a given file is already in the cloud. Such side-channel attacks can be mitigated through probabilistic upload solutions by randomly performing artificial uploads (i.e., uploading a file already present in the cloud) in order to weaken the correlation between the existence of the file and deduplication. Probabilistic uploads do not address server side channel attacks, but have the side-benefit of partially mitigating hash-only attacks.

Harnik et al. [21] proposed the first randomized threshold deduplication technique which has been improved upon by Lee and Choi [26] to provide

Artificial upload	Full upload performed from a user's machine to the cloud, despite the uploaded file being available in the cloud
Bandwidth penalty	The average number of artificial uploads performed by a deduplication algorithm
Total info leakage	The probability with which an adversary can infer either the presence or the absence of a file by exploiting deduplication

Table 1: Terms used in deduplication analysis

better privacy guarantees. In this section, we discuss these two proposals in terms of privacy gain and bandwidth cost.

Randomized Threshold Deduplication [21]. For each file X, the storage server uniformly generates a random threshold value $t_X \in [2, d]$, with d a public parameter (e.g., $d = 30$). The value of t_X is known only to the server. For every file X, the server also keeps a counter c_X, which represents the number of clients who previously uploaded copies of X. A new copy of X is deduplicated only if $c_X \geq t_X$, or if the copy is uploaded by a client that has previously uploaded X. For the first $t_X - 1$ uploads of X, the occurrence of deduplication is effectively hidden from users. The solution still allows an attacker to identify if a file X has been uploaded by d users, or if no user has uploaded the file. This information is leaked when the randomly selected threshold t_X happens to be d or 2.

The Harnik et al. [21] proposal allows an attacker to establish that a file X has been uploaded by d users (with probability $\frac{1}{d-1}$), and that no user has uploaded the file (with probability $\frac{1}{d-1}$). The two events (X present, X absent) are mutually exclusive. However, for comparison purposes we consider the total information leakage probability of the proposed system as being $\frac{2}{d-1}$. By total information leakage probability we represent the combined probability with which an adversary can learn that either one copy or no copy of a target file X is present in the cloud.

The information leakage probability comes with a cost in bandwidth consumption. For each file X, $t_X - 1$ artificial uploads must be performed. Since t_X is chosen uniformly at random from $[2, d]$, the mean (or expected) value will be $(d + 2)/2$. Therefore, on average, the Harnik et al. [21] solution will perform ($t_X - 1 = \frac{d+2}{2} - 1 = \frac{d}{2}$) artificial uploads. We refer to the aver-

Algorithm	HPS [21]	LC [26]
Parameters	d	$d, u;\ d = 3u$
Total info. leakage % - independent file	$\frac{2}{d-1}$	$(\frac{1}{3})^u + (\frac{1}{3})^d$
Total info. leakage % - n related files	$1 - (1 - \frac{2}{d-1})^n$	$1 - \{1 - (\frac{1}{3})^u - (\frac{1}{3})^d\}^n$
Avg. bandwidth cost	$\frac{d}{2}$	$\frac{d}{2}$

Table 2: Overview of parameters and results in Harnik et al. (HPS) and Lee-Choi (LC). Parameters d and u are all natural numbers.

age number of artificial uploads as the "bandwidth penalty"; see Table 1 for terms used.

Although not directly addressed by the Harnik et al. [21] proposal, hash-only attacks are inherently mitigated for the first $t_X - 1$ uploads of X which require a full upload from the client to the CSP. Hash-only attacks can be mounted however once $c_X \geq t_X$ and additional security controls such as PoW schemes (Section 5) are required in order to mitigate them.

Improved Randomized Threshold Deduplication [26].

Lee and Choi [26] expand the randomized solution by lowering the total information leakage probability. With d a public parameter, the server keeps a counter c_X for every file X, which represents the number of clients who have previously uploaded a copy of X. The threshold t_X is initialized with the value of d. For each new uploaded copy of X, the server randomly selects $r \in_R \{0, 1, 2\}$ and performs $t_X = t_X - r$. X is deduplicated at the client side, if $c_X \geq t_X$. The authors also show that the total probability with which an attacker can infer that either one copy or no copy of X was previously uploaded is lowest when d is divisible by 3. In this case, the total information leakage probability is $(\frac{1}{3})^u + (\frac{1}{3})^d$ where $d = 3u$. The associated bandwidth penalty is $\frac{d}{2}$.

As is the case for the Harnik et al. proposal, hash-only attacks become feasible once $c_X \geq t_X$.

By uploading not only the targeted file F, but also additional files correlated to F, the related-files attack can significantly increase the total information leakage probabilities for both Harnik et al. and Lee-Choi [38]. An attacker can infer the presence of a targeted file by observing deduplication occurring for any of its related files. Therefore, the total leakage probability

increases exponentially with the number of related files uploaded to the CSP; see Table 2.

5. Proof of Ownership solutions

Hash-only attacks, as discussed in Section 2.3, can be mitigated using data ownership proofs [7]. Before performing deduplication and linking data to a user's account, the CSP requests from the user random bits of the file, to ensure that the user is actually in possession of the file and not only of the hash values. Failure to provide the randomly requested bits of the file will cause the server to not deduplicate the file, and require a full upload, ensuring that only previously owned data is linked to a user's account.

A similar concept is used in US patent 8528085B1 [24], where for each uploaded file, the CSP generates a secret index referencing a random block of the file. The index and the corresponding data block (or its hash value) are stored along with the file's hash value and used by the CSP to challenge clients who try to upload a file already present in the cloud. The client is requested to produce the file block referenced by the server-specified index. If the block uploaded by the client is identical to the block in the server's storage, the entire file is deduplicated to the user's account. Each time that a client retrieves a file, its secret index is generated anew and the corresponding file block (or it hash value) is used for the next challenge. For each file, the CSP also stores an access control list (comprised of all accounts who have uploaded the file), along with the file's hash value, its secret index and corresponding file block. Clients who are already on the access list are not challenged with the PoW scheme.

Performing random seeks in the server's storage for each uploaded (or downloaded [24]) file is however costly for the server. As a response, a wide array of proof of ownership (PoW) schemes were proposed (e.g., [52, 37, 19, 50, 18, 12]), to provide ownership guarantees without incurring a significant cost to the CSP. Below, we summarize two representative PoW schemes.

PoW schemes of Halevi et al. [19]. Halevi et al. [19] introduced and formalized the notion of proof of ownership (PoW), using the concepts of proof of retrievability (PoR) [25] and proof of data possession (PoD) [2] as a starting point. The proposed PoW solution uses a proof protocol, run between a verifier that is in possession of the root value of a Merkle tree and the number of leaves in the tree, and a prover who claims to know the underlying buffer.

A Merkle [28, 42] or hash tree is constructed by splitting an input buffer (e.g., the file to be deduplicated) into multiple blocks which are hashed and become the leaves of the tree. These hash values are grouped in pairs and each pair is again hashed, producing the parent of the group. The process is repeated until a single hash value, the root of the Merkle tree, remains. Each leaf node has a corresponding sibling path, which consists of the leaf's value and the values of all the siblings of nodes on the path from the given leaf node to the root of the tree. The values of all nodes on the path from a given leaf to the root can be computed from bottom to top, given just the index of the leaf and its sibling path.

To verify that the client is in possession of a file, the server selects a number of leaf indexes and asks the client to provide the corresponding leaf values and the sibling path for each leaf. The server will accept the proof if all sibling paths are valid. A sibling path is said to be valid if its length (e.g., the number of nodes it contains) is the same as the tree height and if the root value computed on the sibling path corresponds with the root value stored on the server.

Three PoW schemes are proposed by Halevi et al. [19], each asking the client for a valid sibling path for a subset of leaves of the file's Merkle tree. The Merkle tree is constructed from a buffer, derived from the original file, but processed in a different way for each scheme: (1) an erasure coding of the file, (2) a sufficiently large universal hash of the file or (3) a mixing and reduction phase.

If an adversary possesses the entire Merkle tree for a file, she will pass the verification process. The protocol allows to set a threshold for how short the Merkle tree size can be. In a proof-of-concept implementation, a size of 64MB is used. This effectively thwarts low-bandwidth hash-only attacks such as the amplified covert channel attack from Section 2.3. A content distribution system would suffer from such large (64MB) unique representations of a file, but would still offer usage incentives for large files. To increase the cost of transferring the unique representation of a file between content distribution network peers, Halevi et al. propose to use multiple Merkle trees (e.g., one over the file itself and one over an encoded version of the file). The server would only need to store the hash of the two roots while the attacker would have to obtain and transfer two entire trees.

As noted by Xu et al. [50], the schemes proposed by Halevi et al. do not protect partial information of user files against a malicious client. As a solution, Xu et al. construct a PoW scheme which is provably secure in

the random oracle model, for any input file distribution with sufficient min-entropy. However, both Halevi et al. and Xu et al. PoW schemes are very demanding for the client, in terms of I/O requirements and CPU cycles [12].

PoW scheme of Di Pietro et al. [12]. A different approach to the ownership problem is proposed by Di Pietro et al. [12]: the cloud server has two types of storage: a large capacity, but slow and resource-demanding back-end storage system, which stores user uploaded files, and a small capacity but low latency front-end storage system, used to store per file challenges served to clients.

The Di Pietro et al. PoW scheme comprises two phases. When a file is uploaded to the cloud provider for the first time (the file is not already present in the cloud), the storage server computes responses for a tunable number of PoW challenges and stores them on the fast storage front-end system. The original file is stored on the back-end storage. During the second phase, when an already existing file is uploaded to the cloud, the storage server will serve a previously unused challenge from the front-end storage and compare the client's response with the pre-computed value. When the pool of unused challenges is depleted, the server fetches the original file from the back-end storage system and produces a new set of challenges and the corresponding responses and saves them in the server's front-end storage.

Any PoW scheme can be abused to turn the CSP into an oracle. An attacker can use the outcome of any PoW scheme to query the CSP if a file is present or absent in the cloud. If a file is absent, all PoW schemes will ask the client for a full upload. However, when a file is already present in the cloud, the client is asked to perform additional tasks in an effort to prove to the CSP that it possesses the entire file. This difference in client-side requests can be leveraged by an attacker to infer if a file is being uploaded for the first time, or if the CSP is engaging in a PoW algorithm.

An intuitive solution, that hides from the client if a file is uploaded for the first time or validated by a PoW scheme, is to run the PoW algorithm every time, even for files that are not present in the cloud. The CSP would of course be unable to validate the proofs received from the client and at the end of the fake PoW run, the CSP must request a full upload to avoid data loss. This means that even if the client possesses the entire file and correctly answers all PoW challenges, the CSP will still request a full upload. Such noticeable behavior can again be used as a side-channel.

A PoW scheme alone does not mitigate against the side-channel attacks introduced in Section 2.2. On the contrary, PoW schemes offer an additional way to establish the presence or absence of a file in the cloud, and should therefore be used in conjunction with side-channel mitigation techniques.

6. Home Gateway-based Deduplication

To address side-channel attacks on deduplication, Heen et al. [22] propose to implement parts of the deduplication process in a user's gateway device that connects the user's home network to the Network Service Provider (NSP). Below, we summarize this solution for a clearer understanding of the weaknesses we describe in Section 7.

The gateway device is used to obfuscate the occurrence of deduplication from both regular users and attackers. The deduplication process is concealed by always copying a user file from the user's computer to the gateway device, and then mixing the deduplication traffic from the gateway device to the NSP with other types of network traffic, as well as obfuscation traffic.

Heen et al.'s approach requires advanced home modem/router devices to be used as gateway devices. Required features for the gateway device include internal storage (500MB is presumed in the paper [22]), and three internal modules: a gateway server (to copy data from the user's PC to the gateway's internal disk), a gateway client (to upload data from the gateway disk to the cloud), and a bandwidth manager (to schedule the upload to the cloud and combine file data with other traffic for obfuscation purposes). The gateway disk, the gateway client and the bandwidth manager are presumed to be in a zone inaccessible to the user (Zone B); the user has access only to the gateway server (Zone A); see Figure 1.

Every file the user wishes to upload to the cloud is copied in its entirety to the gateway server, which saves the file to the internal storage, inaccessible to the user. The gateway server then delegates the upload task to the gateway client. The cloud storage service is queried by the gateway client for the existence of the file in the cloud. If the file is not present in the cloud, the file is added to the bandwidth manager queue. If the file is already present in the cloud, nothing is added to the bandwidth manager queue and the file is linked to the user's account by the cloud storage provider. The bandwidth manager will upload the file when needed, along with other traffic (legitimate traffic such as web browsing and VoIP traffic, as well as obfuscation traffic). After a possible delay, the file is deleted from the gateway disk.

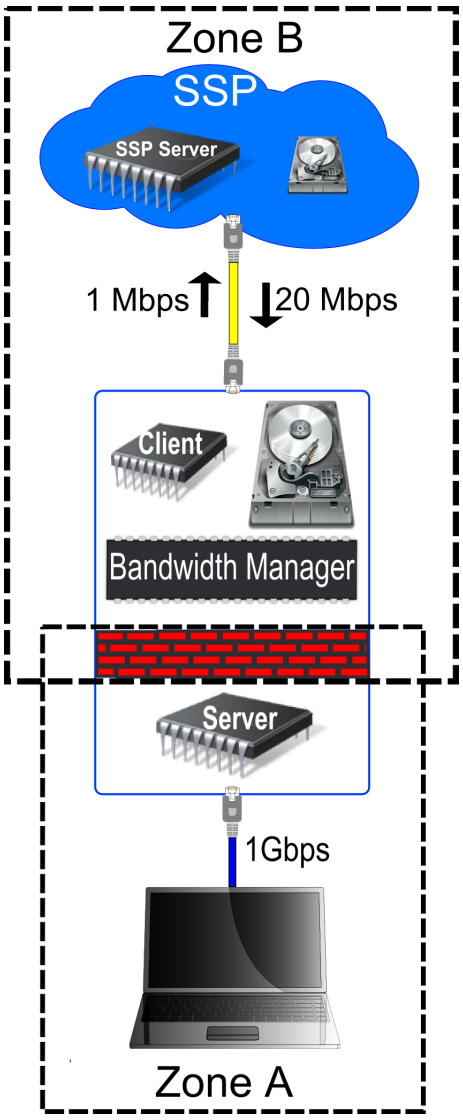

Figure 1: Gateway-based cloud storage system – adapted from [22]. SSP stands for Storage Service Provider

The bandwidth manager combines different types of traffic, catalogued according to their QoS requirements: D (real time support, e.g., VoIP), E (peer-to-peer traffic), F (no QoS requirement, e.g., browsing traffic). The aggregation of these traditional services is denoted by $T = \{D, E, F\}$ (traffic is inter-mixed). C represents cloud storage traffic and O stands for obfusca-

I_X	I_B	B_O	B_C
0	1	B_O	$B_{max} - (B_T + B_O)$
1	1	$\alpha \cdot (B_{max} - B_T)$	0

Table 3: Bandwidth allocation for cloud storage and obfuscation traffic – adapted from [22]

tion traffic. The cloud (B_C) and obfuscation (B_O) bandwidths are allocated as shown in Table 3.

In Table 3, I_X is a binary function indicating the presence ($I_X = 1$) or absence ($I_X = 0$) of file X in the cloud. I_B is a binary function indicating whether the bandwidth required by the T services is smaller or greater than B_{max} (maximum bandwidth available on the gateway uplink). $I_B = 1$ if $B_T < B_{max}$, and $I_B = 0$ otherwise. When $I_B = 0$, then $B_O = 0$ and $B_C = 0$ [22]. The amount of bandwidth that can be allocated for obfuscation traffic, $\alpha \in [0, 1]$ can be adjusted; $\alpha = 0$ offers maximum bandwidth savings, but least resistance against the detection of deduplication, and $\alpha = 1$ offers no bandwidth savings, but most resistance against the detection of deduplication. O can be stuffing traffic or other useful traffic. Zone B of the data path (see Fig. 1) is composed of the gateway client, bandwidth manager (implemented as software or hardware modules) and the authenticated link to the digital subscriber line access multiplexer (DSLAM). This zone is defined as secure and inaccessible to an attacker/user, and thus assumed to provide protection against the detection of deduplication.

7. Weaknesses of the Home Gateway-based solution

We discuss three weaknesses of the Gateway-based deduplication solution, which can lead to deduplication detection at the client side without violating assumptions as used by Heen et al. [22].

7.1. The Attacker's Capabilities

We assume that the attacker is able to perform several actions, which are consistent with the threat model used by Heen et al. An attacker has access to zone A of the network model (see Fig. 1); i.e., the attacker has full control over the traffic that reaches the gateway device and is able to measure and monitor this traffic. The attacker also has physical access to his own gateway device (e.g., he can unplug or reset the device). We also assume that the attacker has a means of measuring the amount of traffic

that leaves the gateway device. This measurement can be performed with dedicated network equipment or inferred from side channels, such as energy consumption [8] or heat release [35] of the gateway device, or by observing the network activity LED (or connecting it to a meter). Most router/modem devices also offer real-time upload/download statistics on the device's web interface. ISPs also offer ways of querying a subscriber's traffic stats. The attacker is not required to inspect traffic that leaves the gateway device.

7.2. Zero Traffic Attack

According to Heen et al. [22], the gateway device makes it impossible for an attacker to learn if a file is already present in the cloud by simply observing the network activity. The attacker cannot distinguish between C (traffic generated from a file being uploaded to the cloud) and O (obfuscation traffic between the gateway device and the DSLAM). This assumption may be bypassed as follows.

An attacker in possession of a gateway device (as any legitimate user of the service will have to be), can control traffic T. (As depicted in Fig. 1: Zone A, where the traffic T is generated, is easily controlled by an attacker). In particular, the attacker can purposefully generate no T traffic at all. In this case ($T = 0$), it follows that for $I_X = 0$ and $I_B = 1$, we have $B_C = B_{max} - B_O, B_O = B_O$ (see Table 3). So the total traffic that leaves the gateway device will be $T_{Total} = B_{max} - B_O + B_O = B_{max}$. Whereas for $I_X = 1, I_B = 1$, we have $B_O = \alpha \cdot B_{max}, B_C = 0$, and thus the total traffic will be $T_{Total} = \alpha \cdot B_{max}$.

Therefore, if $\alpha \neq 1$ the traffic transmitted for $I_X = 0$ (file is not present) will always be greater than the traffic transmitted when the file is present ($I_X = 1$). By simply observing the amount of traffic transmitted (as indicated in Section 7.1), an attacker can know if a file has been deduplicated or not, without the need to inspect the traffic. B_{max} is known, or can be easily measured. In a scenario where the attacker generates zero traffic T, a bandwidth consumption of less than B_{max} will reveal that data is being deduplicated.

Measuring α. We show that α has to be randomly generated for each file; otherwise its value can be easily deduced and used to fine-tune the zero-traffic attack. We consider the following cases: when α is fixed across different file upload sessions, and when α is generated randomly for each session. If α is not randomized before each transfer, its value can be easily deduced in the following way (again, assuming $T = 0$):

1. The attacker generates a set of X_N files, filled with junk data (e.g., from `/dev/urandom`). For ease of measurement, the files can have the same size. As the files are generated with pseudo-random data, they will not be in the cloud storage.
2. The attacker uploads one file at a time, measuring the transmission time for each file. According to the bandwidth manager algorithm (see Table 3), the upload time for each file can be calculated as follows:

$$t_1[i] = \frac{filesize(X_i)}{B_{max}}, \text{ for } i \in \{1 \dots N\}.$$

3. Next, the attacker uploads the same files again, one at a time. At this point the files will already be in the cloud and the transmission time for each file can be calculated as follows:

$$t_2[i] = \frac{filesize(X_i)}{\alpha \cdot B_{max}}, \text{ for } i \in \{1 \dots N\}.$$

4. By observing the time difference, an attacker can calculate the values α_i as follows:

$$\alpha_i = \frac{t_1[i]}{t_2[i]}, \text{ for } i \in \{1 \dots N\}.$$

If α is the same for all uploads, all values of α_i will be equal or slightly fluctuate around the same value. If α is randomly generated before each transmission, the attacker can gain insight into the random number generator by observing α_i.

Regardless of α being random or not, by simply measuring the amount of traffic that leaves the gateway, the attacker can deduce if a file is being deduplicated or not. This attack assumes that $\alpha \neq 1$ and that the attacker is able to set $T = 0$. An obvious solution would be to always saturate the uplink by generating an amount of traffic $B_O = B_{max} - B_T$. This corresponds to the case of $\alpha = 1$ - maximum resistance against deduplication attacks but no bandwidth savings and is equivalent to server-side deduplication.

7.3. File Deletion Attack

The user's home network transfers files to the gateway disk at a significantly higher rate (e.g., 1Gbps) than the rate at which files are uploaded from the disk to the cloud (e.g., 1Mbps). This difference in transfer speed can be exploited to infer when deduplication takes place. As shown below,

randomly delaying, pausing or throttling the transfer of files from the user machine to the gateway disk does not mitigate this attack. Delaying the deletion from the gateway disk of a file that has been transferred or linked to the user account doesn't help either.

The last step of the process that takes place when a user wishes to upload a file is to delete the file from the disk:

> "(9) At last, and with a possible internal delay, the file X is deleted from the gateway disk." [22] (section III-A, step 9).

The second attack can be launched just by knowing the size of the gateway's internal drive. If the size is not present in the device's specifications, it can be easily obtained, e.g., by physically extracting the drive, since all gateways of the same model will have the same disk capacity. We suppose, as assumed in the paper, that the drive can hold a maximum of 500MB.

Step 1 of the process that takes place when a user uploads a file X to the storage service provider (SSP) can be used to infer if deduplication takes place or not. In the Heen et al. scenarios, the SSP (storage service provider) and NSP (network service provider) are the same entity. We quote step 1 for completeness [22]:

> "The user client uploads the file X to the gateway server. This upload is fast because it is performed at the home network LAN speed (typically 8 seconds for a 1 GByte file on a 1 Gb/s home network). The gateway server always accepts the file X from the user, i.e., the gateway server does not apply deduplication. Once the file uploaded, the user client is not anymore required in our scenario: the end user is free to leave the home network or to start another task."

Assume that an attacker would like to check if $File_A$ exists in the cloud; also assume the arbitrary size of 20MB for $File_A$. The attacker generates $File_B$ of pseudo-random content, with the file size equal to the total disk capacity of the gateway's internal drive (in this example 500MB). The attacker then proceeds to upload $File_A$ to the cloud. As soon as $File_A$ has been transferred to the gateway's disk (which can be observed by monitoring the traffic between the attacker's machine and the gateway), the attacker initiates the upload of $File_B$ to the cloud. However, $File_B$ cannot be stored in its entirety on the gateway's drive before $File_A$ is deleted. $File_A$ in turn

cannot be deleted from the drive before it exists in the cloud, either by being copied from the gateway's disk or because it was previously uploaded to the cloud by a different user. If $File_A$ does not exist in the cloud and must be uploaded, the transfer of $File_B$ from the user's machine to the gateway's disk will be delayed by the time it takes to upload and then delete $File_A$. This scenario of uploading in sequence 2 or more files with a combined size exceeding the gateway's disk capacity, remains unaddressed in the Heen et al. proposal.

7.3.1. Simple Counter-Measures and Their Shortcomings

Below we briefly discuss four simple approaches to address the file deletion attack; we also analyze shortcomings of these proposals.

(a) Delayed upload. In the first case, the gateway server might delay the upload of $File_B$ until the drive has enough free space to hold the entire file. However, the attacker can use this initial delay as an indication that $File_A$ is not being deduplicated.

(b) Simultaneous upload. Another approach would be to start the upload of $File_B$ to the gateway's drive immediately, storing it in the available space, i.e., total capacity − sizeof ($File_A$). In our current example, the available space is: $(500 - 20)$MB = 480MB. In this case, after copying a certain percentage of $File_B$ to the gateway drive, the drive will have no more space available and the upload must be paused until $File_A$ is deleted from the drive. This pause can be used by an attacker to infer that $File_A$ is being copied to the cloud.

It is possible that the upload and deletion of $File_A$ completes before the drive runs out of space. The conditions for this event to occur can be calculated as follows. In the case of Heen et al.'s 1Gbps home network example, if $File_A$ is 20MB and $File_B$ is 500MB, the drive will fill up in 3.75 seconds. Considering that the stated ADSL uplink speed is 1Mbps, only 0.46875 MB of $File_A$ will be uploaded to the cloud before the drive fills up. For the given parameters (network speed and disk size), any $File_A$ larger than 500KB (the exact threshold is 499.507KB) will not be uploaded quickly enough to free up the space for $File_B$.

The 500KB value is obtained as follows. Assume a file of size X MB that has been copied from the user's machine to the 500MB gateway disk. It takes $8 \cdot X$ seconds (the uplink speed is 1 Mbit/s, and the filesize is given in MBytes) to upload the file from the gateway device to the cloud over a

1Mbps uplink. It takes $(500 - X)$ MB to fill up the drive. The time to copy $(500 - X)$ MB from the user's device to the gateway disk is $\frac{8 \cdot (500-X)}{1024}$ sec over the assumed 1Gbps connection. Therefore, in order for the file to be uploaded to the cloud before the drive fills up, its size X must satisfy the following relation: $\frac{500-X}{128} > 8X$, or $X < \frac{500}{1025} = 0.48780$ MB $\backsimeq$ 500 KB. The time it takes to delete $File_A$ from the drive is assumed to be zero. Thus, for any file larger than 500KB, an attacker can determine if it is being duplicated or not.

Note that, the minimum size for $File_A$ can be significantly increased by using a larger gateway disk (e.g., to 9.99MB for a disk size of 10GB, or 99.90MB for a disk size of 100GB). The size of $File_B$ could be limited by the user's cloud storage quota, if this value is known to the gateway server. In such a scenario, the gateway server could disrupt or refuse the upload of a file whose size (or amount of bytes already transferred to the gateway disk) exceeds the user's cloud quota. This hurdle can be easily overcome by the attacker, by simply generating a series of pseudo-random files, $File_{Bi}, 0 \leq i \leq n$, so that the size of $File_{Bi}$ is less than the user's cloud quota (minus the size of $File_A$ if the available quota is updated before the upload completes). A disrupted or delayed upload of any file $File_{Bi}$ in the series, indicates that $File_A$ is not being deduplicated.

(c) Throttled upload speed. The gateway server can also throttle the upload of $File_B$ to the gateway disk so as to provide enough time for $File_A$ to be uploaded to the cloud. However, this change in upload speed can be easily detected by an attacker.

(d) Delayed file deletion. To obfuscate if $File_A$ has been deduplicated, the deletion time must be delayed with a period equal to or greater than the time it would take to perform the actual upload to the cloud. This is feasible, since the gateway client knows the size of the file to be deduplicated as well as the uplink speed. However, even in this scenario, the attacker can still learn if a file is indeed uploaded to the cloud or just kept on the gateway disk for a time equal to the upload time. By generating other types of upload traffic (e.g., D, E, F) the attacker can saturate the upload link to various degrees and further delay the deletion of $File_A$. If the file is indeed uploaded from the gateway's disk to the cloud, the upload time will be delayed in accordance to the other upload traffic that the attacker generates. These delays will be reflected in the time it takes to upload $File_B$ from the user's device to the the gateway disk. If instead the file has been deduplicated and its deletion is

delayed from the gateway disk with a pre-established value (calculated by the gateway client from the filesize and uplink speed), the upload traffic that the attacker generates will have no effect on the time it takes to upload $File_B$.

7.4. Forced Power-Cycle Attack

An attacker may also detect file deduplication by interrupting the traffic between the gateway device and the CSP. Since the attacker knows the size of the file she is about to upload, as well as the uplink speed, she can calculate the minimum amount of time that a full upload would take (e.g., a 50MB file will take 6.6 minutes over a 1Mbps uplink). The attacker can unplug or power-cycle the gateway device before a full upload would complete (e.g., after four minutes for a 50MB file).

If deduplication occurred, then the file would be linked to the attacker's account even though the gateway was disconnected. On the other hand, if the file was not already present in the cloud, then the actual upload process was terminated prematurely, and either no file or only a portion of the file will be present in the attacker's account. Even if the content of a user's account cannot be viewed from a device that does not connect through the gateway device (e.g., a web interface accessed through a different Internet connection), it is still possible to infer deduplication as follows. After the gateway device is reconnected, the attacker observes the amount of traffic that leaves the gateway device (the attacker purposely generates zero T traffic; see Section 7.2). If the file was deduplicated, no traffic will be sent after the gateway device authenticates with the DSLAM. Traffic that would match the remaining data to be uploaded indicates that a full upload was interrupted and is now resumed. This assumes that the gateway device will try to re-upload the file or resume a previously failed upload. If the upload is not re-initiated or continued after an unexpected power cycle or disconnect, the attacker can wait until the file is deleted (or perhaps trigger the deletion by uploading a file of pseudo-random content whose size equals the disk capacity). Once the file is deleted from the gateway disk, the attacker verifies if the file appears in her CSP account. If the file is present in the attacker's CSP account, it means the file was deduplicated.

The attacks described above are made possible by weaknesses of the deduplication algorithm that are exploitable due to the amount of control that users have over their home gateway devices. The attacks can be mitigated by either rethinking the deduplication algorithm, or by positioning the gateway device outside of the user's control. For instance, the gateway device could

be placed at the ISP level for home users, or a the DMZ/perimeter level for corporate scenarios. In both cases, attackers would no longer be able to control the amount of traffic reaching the gateway device, measure the amount of traffic leaving the gateway device, or power cycle the device. Given the higher throughput and lower cost of subscriber-to-ISP and internal-network-to-perimeter links compared to WAN links, the approach could still provide implementation incentives. Of course, the gateway device would have to be re-sized (and potentially redesigned) accordingly.

8. Storage Gateway-based Deduplication

Shin et al. [38] propose a storage gateway-based deduplication solution that relies on a local, disk-attached network server, which resides at the customer's site, and acts as a broker between multiple users and the CSP. The approach is targeted towards scenarios involving multiple users uploading their data through the same local network storage server, such as a corporate setup (see Figure 2). By providing differential privacy guarantees, the proposed deduplication algorithm is shown to mitigate side channel attacks, including independent file and related files attacks (see Section 2.2).

The concept of differential privacy was introduced by Dwork et al. [16] for the purpose of disclosure and inference control in statistical database scenarios (i.e., releasing statistical data without compromising the privacy of individual respondents). Intuitively, if the probability of any result of a randomized function is almost independent of whether an individual item is present or not in the input, the randomized function is said to satisfy differential privacy. Formally, a randomized function R gives ϵ-differential privacy, if for all data sets D_1 and D_2 that differ in a single item, and all $S \subseteq Range(R)$, $Pr\{R(D1) \in S\} \leq e^{\epsilon} \cdot Pr\{R(D_2) \in S\}$.

Shin et al. observe that deduplication can be defined mathematically as a function $R_S(F)$, where a file F is the input of the function R and the amount of network traffic represents its output based on whether a storage S contains the file F or not. By assuming S_1 and S_2 to be two storages that differ only in the file F, $S_2 = S_1 \cup \{F\}$, a randomized function $R_S(F)$ satisfies ϵ-differential privacy, if for all storages S_1 and S_2 and all $0 \leq \tau \leq filesize(F)$, $Pr\{R_{S_2}(F) = \tau\} \leq e^{\epsilon} \cdot Pr\{R_{S_1}(F) = \tau\}$.

Side-channel deduplication attacks can be modeled as an attacker trying to identify whether the storage S is S_1 or S_2. Therefore, a deduplication protocol that provides ϵ-differential privacy is not susceptible to side channel attacks. To also be resilient against related-files attacks, a deduplication

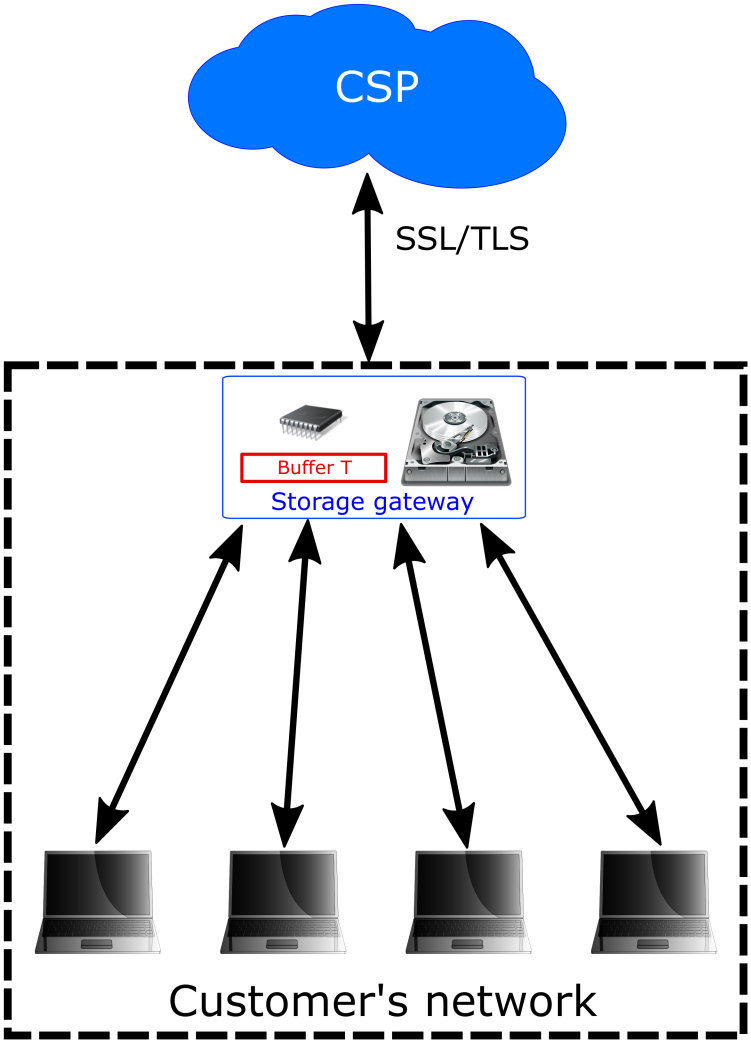

Figure 2: Storage gateway-based cloud storage system – adapted from [38]

.

protocol has to prevent information leakage when an attacker uploads the related files $\{F_1, F_2, ...F_n\}, n > 1$, meaning the probability of an attacker distinguishing S_1 from $S_2 = S_1 \cup \{F_1, F_2, ..., F_n\}$ is the same as the probability of distinguishing S_1 from $S_2 = S_1 \cup \{F\}$.

The deduplication protocol is implemented on the storage gateway device, which determines the amount of traffic sent to the CSP by running a differentially private algorithm. When a user uploads a file to the CSP, the entire file is first transferred to the gateway's local storage. The gateway device queries the CSP for the existence of the file by uploading the file's hash and receiving a true/false response. The amount of data to be transferred from the gateway device to the CSP, $\tau = \lceil \alpha \cdot filesize(F) \rceil$ is calculated by using a Poisson-distributed random number with the mean $\alpha \in [0, 1]$.

If F does not exist in the cloud, $\alpha \xleftarrow{R} Poi(\frac{1-\epsilon}{2})$ and a subset of $\tau = \lceil \alpha \cdot filesize(F) \rceil$ bytes of F, $\{b_1, b_2, ..., b_\tau\}$ is transferred to the cloud. The remaining bytes of F are en-queued into a buffer T on the gateway device. If F already exists in the cloud, $\alpha \xleftarrow{R} Poi(\frac{1+\epsilon}{2})$, and $\tau = \lceil \alpha \cdot filesize(F) \rceil$ bytes of data $\{c_1, c_2, ..., c_\tau\}$ are de-queued from the buffer T and then transferred to the CSP.

If T does not hold enough data, junk data (pseudo-random bytes) are sent instead. Using a buffer T, which holds complementary segments of already uploaded file portions, the network overhead is reduced; data transmitted to the CSP when uploading an already existing file will mostly consist of file chunks that need to be uploaded in order to complete the transfer of previous files. Shin et al. show that the proposed protocol provides ϵ-differential privacy for both individual files and related-files scenarios.

The bandwidth penalty is smaller than in the Harnik et al. and Lee-Choi proposals, since dummy data is uploaded only when the buffer T holds less data than needs to be transmitted. However, unlike for Harnik et al. and Lee-Choi, an overall average bandwidth penalty cannot be calculated. Each time a file is deduplicated, the amount of artificial traffic sent to the CSP will be given by the difference between the amount of data stored in the buffer T (a chunk of a previous file that needs to be uploaded), and the amount of data that must be uploaded in order to satisfy ϵ-differential privacy. This amount is dependent on the size of specific files and the order in which they are uploaded. Empirically, running 3TB of data through the proposed protocol configured for the highest security guarantee ($\epsilon = 0$) resulted in 48GB of artificial traffic (1.56%).

The zero traffic, file deletion and forced power-cycle attacks introduced in Section 7 are not applicable since (i) the attacker has no longer control over the gateway device; (ii) he is unaware of the disk capacity (which for a storage gateway is likely to be larger than for a workstation); and (iii)

he does not have the advantage of a significant difference between the LAN speed and the gateway uplink speed.

Both the home- and storage-gateway solutions always perform a full upload from the client to the gateway device, thereby mitigating hash-only attacks. Since the solutions do not use any form of encryption, server side-channel attacks are still feasible.

The effectiveness of gateway-based deduplication approaches is limited in scope, since it only addresses attacks launched from behind the gateway device. Files stored by large CSPs are accessible from all over the Internet and side-channel attacks can be launched from networks whose traffic does not pass through the proposed device, thereby circumventing its security controls.

9. Comparison

Table 4 summarizes our comparison of existing solutions. We evaluate these solutions in terms of security (i.e., prevention of known attacks), deployment cost (e.g., latency, throughput, bandwidth, CPU) and target environment. We also list special requirements and parameters for each solution, if applicable. By design, server-side deduplication solutions will always perform a full upload from the client to the server and are therefore not susceptible to hash-only or client side-channel attacks. The downside of server-side deduplication proposals is that they provide no bandwidth savings. Client-side deduplication proposals on the other hand do not address all user attacks.

10. Conclusions

Cloud storage services are heavily used by both home and enterprise users, cf. [14, 11]. Most such services strongly rely on (cross-account) deduplication. Thus, deduplication is likely to remain in use for the foreseeable future, even though existing deduplication mechanisms pose a significant threat to user privacy. In this paper, we provide a systematic and comprehensive survey of deduplication-related privacy attacks, perform a security analysis of several leading mitigation proposals, evaluate their efficiency and deployability, and present three attacks on a home gateway-based deduplication solution. While most proposals address one specific exploitation method (e.g., POW schemes or probabilistic uploads), none of the existing proposals attempt to address all threat categories explored in this paper and at the same time take advantage of both bandwidth and storage savings that deduplication can offer.

	Deployment			Attacks mitigated			Additional remarks
	Dedup side	Target environment	Cost	Hash only	Side channel (User)	Side channel (CSP)	
Encryption							
Convergent [13] - server-side	Server	Unrestricted		✓	✓	✗	
Convergent [13] - client-side	Client	Unrestricted		✗	✗	✗	
DupLESS [3]	Server	Limited to key server scope	Network latency (14-17%)	✓	✓	✓	
P2P DupLESS [15]	Server	Limited to P2P scope	Throughput drop (4-30%)	✓	✓	✓	
Leakage-resilient [50]	Client	Unrestricted		✓	✗	✗*	
Liu et al. [27] - original	Server	Unrestricted		✓	✓	✓	Previous clients need to be online when the protocol is run
Liu et al. [27] - extension	Client	Unrestricted	Bandwidth penalty $d/2$	✗*	✓	✓	Previous clients need to be online when the protocol is run. Total info. leakage prob. is $\frac{2}{d-1}$
Probabilistic upload							
Harnik et al. [21]	Client	Unrestricted	Bandwidth penalty $d/2$	✗*	✓	✗	Total info. leakage prob. is $\frac{2}{d-1}$
Lee & Choi [26]	Client	Unrestricted	Bandwidth penalty $d/2$	✗*	✓	✗	Total info. leakage prob. is $(\frac{1}{3})^u + (\frac{1}{3})^d$
PoW							
Halevi et al. [19]	Client	Unrestricted	Client CPU overhead (e.g., 53ms/1MB, 1.5s/1GB)	✓	✗	✗	
Di Pietro et al. [12]	Client	Unrestricted	CPU overhead (e.g., client CPU cycles plateau under 10^{11})	✓	✗	✗	
Gateway-based							
Heen et al. [22]	Client	CSP, NSP are the same entity		✓	✗	✗	Requires special home gateway devices
Shin et al. [36]	Client	Limited to storage gateway scope		✓	✓	✗	Offers ϵ-differential privacy, Requires a storage gateway device and enterprise setup

Table 4: Comparison of existing privacy-friendly deduplication proposals. Server-side deduplication solutions are not susceptible to hash-only and user side-channel attacks, but provide no bandwidth savings. A star (*) indicates that the attack might be feasible is certain cases: when the CSP plays the role of a client (Leakage-resilient proposal) or when the file is deduplicated at the client side (Harnik et al. and Lee-Choi proposals). d is a system parameter.

Because deduplication can be abused in many ways, a secure end-to-end solution has to be resilient against all types of known attacks on deduplication and ideally offer both bandwidth and storage savings. Such a solution can be achieved through a novel design or by combining existing solutions that each address specific attacks. Ways in which different deduplication solutions can be combined (e.g., combining PoW with probabilistic solutions) and how efficient the various combinations would be have not been analyzed yet and constitute a topic for future research. Our goal here is to highlight the missing pieces and limitations of existing privacy-friendly deduplication proposals to help attract more research in this area.

References

[1] P. Anderson and L. Zhang. Fast and secure laptop backups with encrypted de-duplication. In *USENIX Conference on Large Installation System Administration (LISA'10)*, San Jose, CA, USA, Nov. 2010.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *ACM Conference on Computer and Communications Security (CCS'07)*, pages 598–609, Alexandria, VA, USA, 2007.

[3] M. Bellare, S. Keelveedhi, and T. Ristenpart. DupLESS: server-aided encryption for deduplicated storage. In *USENIX Security Symposium*, Washington, DC, USA, Aug. 2013.

[4] R. Bhadauria and S. Sanyal. Survey on security issues in cloud computing and associated mitigation techniques. *International Journal of Computer Applications*, 47(18):47–66, June 2012.

[5] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: Theory and implementation. In *ACM Cloud Computing Security Workshop (CCSW'09)*, pages 43–54, Chicago, IL, USA, Nov. 2009.

[6] Boxcryptor.com. Boxcryptor: encryption for your cloud storage. Multiplatform solution for major cloud storage provider including Dropbox, Google Drive, and Microsoft SkyDrive. `https://www.boxcryptor.com`.

[7] C. Cachin and M. Schunter. A cloud you can trust. *IEEE Spectrum*, 48(12):28–51, Dec. 2011.

[8] A. O. Calchand, P. Branch, and J. But. Analysis of power consumption in consumer ADSL modems. Technical Report 100125A, Swinburne University of Technology, Jan. 2010. `http://researchbank.swinburne.edu.au/vital/access/manager/Repository/swin:19993`.

[9] A. Chakra, S. P. O'Doherty, J. Rice, and B. K. Yap. Embedding a unque serial number into the content of an email for tracking information dispersion. US Patent 20090187629 (publication date: July 23, 2009).

[10] D. Chaum. Blind signatures for untraceable payments. In *Crypto'82*, pages 199–203. Springer, 1982.

[11] CNET.com. Dropbox clears 1 billion file uploads per day. News article (Feb. 27, 2013). `http://reviews.cnet.com/8301-13970_7-57571513-78/dropbox-clears-1-billion-file-uploads-per-day/`.

[12] R. Di Pietro and A. Sorniotti. Boosting efficiency and security in proof of ownership for deduplication. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'12)*, Seoul, Korea, May 2012.

[13] 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 *IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, pages 617–624, Vienna, Austria, July 2002.

[14] I. Drago, M. Mellia, M. M. Munafò, A. Sperotto, R. Sadre, and A. Pras. Inside Dropbox: understanding personal cloud storage services. In *Internet Measurement Conference (IMC'12)*, Boston, MA, USA, Nov. 2012.

[15] Y. Duan. Distributed key generation for encrypted deduplication: Achieving the strongest privacy. In *ACM Cloud Computing Security Workshop (CCSW'14)*, pages 57–68, Scottsdale, AZ, USA, Nov. 2014.

[16] C. Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation (TAMC'08)*, pages 1–19. Springer, Xi'an, China, Apr. 2008.

[17] A. Fairless. Why SpiderOak doesn't de-duplicate data across users. Online article (Aug. 28, 2010). `https://spideroak.com/articles/why-spideroak-doesnt-deduplicate-data-across-users-and-why-it-should-worry-you-if-we-did`.

[18] C.-I. Fan, S.-Y. Huang, and W.-C. Hsu. Hybrid data deduplication in cloud environment. In *Information Security and Intelligence Control (ISIC'12)*, pages 174–177, Taiwan, Aug. 2012.

[19] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of ownership in remote storage systems. In *ACM Conference on Computer and Communications Security (CCS'11)*, Chicago, IL, USA, Oct. 2011.

[20] D. Harnik, O. Margalit, D. Naor, D. Sotnikov, and G. Vernik. Estimation of deduplication ratios in large data sets. In *IEEE Symposium on Mass Storage Systems and Technologies (MSST'12)*, Pacific Grove, CA, USA, Apr. 2012.

[21] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security and Privacy*, 8(6):40–47, Nov–Dec 2010.

[22] O. Heen, C. Neumann, L. Montalvo, and S. Defrance. Improving the resistance to side-channel attacks on cloud storage services. In *International Conference on New Technologies, Mobility and Security (NTMS'12)*, pages 1–5, Istanbul, Turkey, May 2012.

[23] HowToGeek.com. How to encrypt your cloud-based drive with Boxcryptor. Online article (June 28, 2011). `http://www.howtogeek.com/67074/how-to-encrypt-your-cloud-based-drive-with-boxcryptor/`.

[24] A. Juels. Method and system for preventing de-duplication side-channel attacks in cloud storage systems, Sept. 3 2013. US Patent 8,528,085.

[25] A. Juels and B. S. Kaliski Jr. PORs: Proofs of retrievability for large files. In *ACM Conference on Computer and Communications Security (CCS'07)*, pages 584–597, Alexandria, VA, USA, 2007.

[26] S. Lee and D. Choi. Privacy-preserving cross-user source-based data deduplication in cloud storage. In *IEEE International Conference on ICT Convergence (ICTC'12)*, pages 329–330, Jeju Island, Korea, Oct. 2012.

[27] J. Liu, N. Asokan, and B. Pinkas. Secure deduplication of encrypted data without additional independent servers. In *ACM Conference on Computer and Communications Security (CCS'15)*, Denver, CO, USA, Oct. 2015.

[28] R. C. Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, Stanford University, June 1979.

[29] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. In *USENIX Conference of File and Storage Technologies (FAST'11)*, San Jose, CA, USA, Feb. 2011.

[30] 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 *USENIX Security Symposium*, San Francisco, CA, USA, 2011.

[31] E. Rozier. The perils of cross-silo deduplication: Trading user security for provider storage efficiency. In *IEEE Workshop on Information Forensics and Security (WIFS'13)*, pages 85–90, Guangzhou, China, Nov. 2013.

[32] Safeboxapp.com. Safebox: Secure your life. Multi-platform encryption tool for Dropbox. `http://www.safeboxapp.com`.

[33] H. Shacham and B. Waters. Compact proofs of retrievability. In *ASIACRYPT'08*, pages 90–107. Springer, Melbourne, Australia, Dec. 2008.

[34] Z. Sheng, Z. Ma, L. Gu, and A. Li. A privacy-protecting file system on public cloud storage. In *Conference on Cloud and Service Computing (CSC'11)*, pages 141–149, Hong Kong, China, Dec. 2011. IEEE.

[35] I. J. Sherlock. Method for achieving adaptive thermal management of dsl modems, July 6 2001. US Patent App. 09/900,394.

[36] J. Shin, Y. Kim, W. Park, and C. Park. DFCloud: A TPM-based secure data access control method of cloud storage in mobile devices. In *IEEE Cloud Computing Technology and Science (CloudCom'12)*, pages 551–556, Taipei, Taiwan, Dec. 2012.

[37] Y. Shin, J. Hur, and K. Kim. Security weakness in the proof of storage with deduplication. IACR Cryptology ePrint Archive, Report 2012/554, 2012. `http://eprint.iacr.org/2012/554.pdf`.

[38] Y. Shin and K. Kim. Differentially private client-side data deduplication protocol for cloud storage services. *Security and Communication Networks*, 8(12):2114–2123, Aug. 2015.

[39] V. Shoup. Practical threshold signatures. In *Eurocrypt'00*, pages 207–220, Bruges, Belgium, May 2000. Springer.

[40] C. Soghoian. How Dropbox sacrifices user privacy for cost savings. Online article (Apr. 12, 2011). `http://paranoia.dubfire.net/2011/04/how-dropbox-sacrifices-user-privacy-for.html`.

[41] Sophos.com. Many Amazon S3 cloud storage users are exposing sensitive company secrets, claims report. Online article (Mar. 29, 2013). `http://nakedsecurity.sophos.com/2013/03/29/amazon-s3-cloud-storage-data-leak/`.

[42] M. Szydlo. Recent improvements in the efficient use of Merkle trees. Online article (Mar. 10, 2004). `http://www.emc.com/emc-plus/rsa-labs/historical/recent-improvements-efficient-use-merkle-trees.htm`.

[43] Tarsnap.com. Online backups for the truly paranoid. http://www.tarsnap.com.

[44] W. van der Laan. Dropship. Open source project (Apr. 2011). https://github.com/driverdan/dropship.

[45] Viivo.com. Cloud file encryption. Multi-platform encryption tool for Dropbox, Box, Drive and SkyDrive. http://www.viivo.com.

[46] C. Wang, Z.-g. Qin, J. Peng, and J. Wang. A novel encryption scheme for data deduplication system. In *International Conference on Communications, Circuits and Systems (ICCCAS'10)*, pages 265–269, Chengdu, China, July 2010. IEEE.

[47] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(5):847–859, 2011.

[48] D. Wilson and G. Ateniese. "To share or not to share" in client-side encrypted clouds. In *Information Security Conference (ISC'14)*, Hong Kong, China, Oct. 2014.

[49] Wired.com. New DRM will change the words in your e-book. News article (June 17, 2013). http://www.wired.com/gadgetlab/2013/06/new-ebook-drm/. Project site: https://www.sit.fraunhofer.de/de/angebote/projekte/sidim/.

[50] J. Xu, E.-C. Chang, and J. Zhou. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'13)*, Hangzhou, China, May 2013.

[51] ZDNet.com. Epsilon data breach: What's the value of an email address. Blog post (Apr. 5, 2011). http://www.zdnet.com/blog/btl/epsilon-data-breach-whats-the-value-of-an-email-address/46915.

[52] Q. Zheng and S. Xu. Secure and efficient proof of storage with deduplication. In *ACM conference on Data and Application Security and Privacy (CODASPY'12)*, pages 1–12, San Antonio, TX, USA, Feb. 2012.