Side channels in cloud services, the case of deduplication in cloud storage

Danny Harnik
IBM Haifa Research Lab
dannyh@il.ibm.com

Benny Pinkas
Bar Ilan University
benny@pinkas.net

Alexandra Shulman-Peleg
IBM Haifa Research Lab
shulmana@il.ibm.com

***Abstract*—Cloud storage services commonly use deduplication, which eliminates redundant data by storing only a single copy of each file or block. Deduplication reduces the space and bandwidth requirements of data storage services, and is most effective when applied across multiple users, a common practice by cloud storage offerings.**

We study the privacy implications of cross-user deduplication. We demonstrate how deduplication can be used as a side channel which reveals information about the contents of files of other users. In a different scenario, deduplication can be used as a covert channel by which malicious software can communicate with its control center, regardless of any firewall settings at the attacked machine.

Due to the high savings offered by cross-user deduplication, cloud storage providers are unlikely to stop using this technology. We therefore propose simple mechanisms that enable cross-user deduplication while greatly reducing the risk of data leakage.

Keywords: Cloud storage, deduplication, side channels, differential privacy.

1 Introduction

The fast growth of data volumes leads to an increased demand for online storage services, ranging from simple backup services to cloud storage infrastructures. Remote backup services give users an online system for collecting, compressing, encrypting, and transferring data to a backup server that is provided by the hosting company. *Cloud storage* refers to scalable and elastic storage capabilities that are delivered as a service using Internet technologies with elastic provisioning and use-based pricing that does not penalize users for changing their storage consumption without notice [9], [1].

The term *data deduplication* refers to techniques that store only a single copy of redundant data, and provide links to that copy instead of storing other actual copies of this data.[1] With the transition of services from tape to disk, data deduplication has become a key component in the backup process. By storing and transmitting only a single copy of duplicate data, deduplication offers savings of both disk space and network bandwidth. For vendors, it offers secondary cost savings in power and cooling achieved by reducing the number of disk spindles [8]. According to recent statistics, deduplication is considered to be the most-impactful storage technology and it is estimated to be applied to 75% of all backups in the next few years [8].

The work of Benny Pinkas was supported by the SFEROT project funded by the European Research Council (ERC).

[1]Note that this does not refer to planned storage redundancy, such as RAID, which is used for providing data durability, but rather to excess copies stored by the users.

1.1 Approaches to Deduplication

Data deduplication strategies can be categorized according to the basic data units they handle. In this respect there are two main data deduplication strategies: **(1) File-level deduplication**, in which only a single copy of each file is stored. Two or more files are identified as identical if they have the same hash value. This is a very popular type of service offered in multiple products [5], [2]; **(2) Block-level deduplication**, which segments files into blocks and stores only a single copy of each block. The system could either use fixed-sized blocks [7] or variable-sized chunks [6], [11]. The discussion in this paper may be applied to both strategies.

In terms of the architecture of the deduplication solution, there are two basic approaches. In the **target-based approach** deduplication is handled by the target data-storage device or service, while the client is unaware of any deduplication that might occur. This technology improves storage utilization, but does not save bandwidth. On the other hand, **source-based deduplication** acts on the data at the client before it is transferred. Specifically, the client software communicates with the backup server (by sending hash signatures) to check for the existence of files or blocks. Duplicates are replaced by pointers and the actual duplicate data is never sent over the network. The advantage of this approach is that it improves both storage and bandwidth utilization.

The effectiveness of deduplication: The effectiveness of deduplication depends on multiple factors such the type of data, the retention period and the number of users. The percentage of space reduction is calculated as 100% less the inverse of the space reduction ratio, whereas, e.g., even a deduplication ratio of 1:3 results in a 66% saving. Reported deduplication ratios in common business settings range from 1:10 to 1:500, resulting in disk and bandwidth savings of more 90% [3]. These savings translate into huge financial savings to the providers and users of cloud storage services.

1.2 Privacy Risks in Cloud Storage and Deduplication

There is an inherent risk in entrusting data in the storage cloud, since the data owner is basically releasing control over your data. Yet, in reality, a wide range of users and applications are more than willing to hand over their data storage tasks to a

cloud provider. They put their trust in the integrity of the cloud provider and in the security of the access control mechanisms that it uses.

Setting these issues aside, we point out an additional threat – the privacy implications of cross-user deduplication. We demonstrate how deduplication in cloud storage services can be used as a side channel which reveals information about the contents of files of other users. In a different scenario, deduplication can be used as a covert channel by which malicious software can communicate with its command and control center, regardless of any firewall settings at the attacked machine.

We analyze these threats and propose a simple mechanism that enables cross-user deduplication while greatly reducing the risk of data leakage. More specifically, the proposed method is a mechanism stating rules by which deduplication is sometimes artificially turned off. We quantify the guarantees of this simple practice. This gives clients a guarantee that adding their data to the cloud has a very limited effect on what an adversary may learn about this data. Thus, it is possible to essentially ensure clients of the privacy of their data.

2 Security Issues

The attacks we describe can be applied to deduplication that is performed either at the file level or at the block level (to be concrete, we assume from now on that deduplication is performed at the file level). There are, however, two features of the deduplication service that are crucial for the attacks:

- **Source-based deduplication.** That is, deduplication must be performed at the client side. As mentioned above, this version of deduplication saves bandwidth and is therefore commonly used. The result of applying this approach is that the client can observe whether a certain file or block was deduplicated (or "deduped" in short). This can be done by either examining the amount of data transferred over the network, or by observing the log of the storage software, if that software provides this type of report.
- The second feature which is crucial for the attack is **cross-user deduplication**. That is, each file or block is compared to the data of *other* users, and is deduped if an identical copy is already available at the server. This approach is popular since it saves storage and bandwidth not only when a single user has multiple copies of the same data, but also when different users store copies of the data. (Enterprise clients often store multiple copies of identical, or similar, data. We found out that this is true even for private customers: almost every common software manual or media file that we tried to backup using popular backup services was found to be already available on the servers and was therefore deduped. Note that these are huge files and therefore deduplication offers great savings to the service providers.)

Identifying storage providers susceptible to the attack: We performed the following test to identify services that perform source-based and cross-user deduplication (the test can be repeated by any reader, on the storage service of his of her choice): (1) We installed the client software of the service on two different computers and created two different user accounts; (2) We used one account to upload a file (in our tests this file was Sun's VirtualBox software of size almost 73M); (3) We used the second account to upload the same file again, checking whether it is indeed uploaded. When the file was not re-transmitted over the network we concluded that the backup service performed source-based, cross-user deduplication. (In fact, when checking popular storage services there is no need to use two accounts, since, as described above, any popular file found on the web is likely to exist on the servers, as it was previously uploaded by other users. Therefore the test can consist of downloading a popular file from the web, uploading it to the service and checking whether deduplication occurs.)

We identified services of three leading backup and file synchronization providers that perform cross-user, source-based deduplication. These services were (1) **DropBox**, a popular file sharing and backup service which crossed the 3 million user milestone;[2] 2) **Mozy**, which is a leading provider of online backup for consumers and businesses, providing backup to over one million customers and 50,000 business users, and storing more than 25 petabytes;[3] and (3) **Memopal** which was ranked by Backup Review as the best online backup service in Europe, with almost 1000 new subscribers per day[4] Notably, most vendors do not try to hide the fact that deduplication occurred, and in our tests this fact was easily detected in several simple ways: (1) Checking the history or the log file (this approach works with MozyHome[5]); (2) According to the upload status message, which differs between uploaded and deduplicated files (this approach works with Memopal[6]); (3) According to the upload speed, checking if an upload of a file is finished within a time which is much shorter than the time required by the upload bandwidth of the client machine (this is the case with DropBox[7]); (4) Finally, the most generic deduplication detection method, which is applicable to all the services regardless of their interface, is to monitor network traffic and measure the amount of transmitted data. We note that most services have additional client-server communication traffic, but it is negligible compared to the large volumes of data transmitted when uploading large files.

When the requirements listed above are met, the storage service essentially serves as an "oracle", which provides an answer to the following query: "Did any user previously upload a copy of this file?" This query is answered by the

[2] http://www.geek.com/articles/news/dropbox-reaches-3-million-user-milestone-20091126.

[3] Mozy press release, February 18, 2010, http://mozy.com/news/releases/ comcast-launches-secure-backup-share-online-storage-solution-for-its-internet-customers.

[4] http://www.memopal.com/en/pressrelease/memopal-online-backup-ranked-as-the-best-online-backup-serv.htm

[5] MozyHome, version 1.16.4.0 reports in the history tab that the file is "already on Mozy servers".

[6] Memopal, version 2.0.0, build 1326 highlights deduplicated files by showing a message "TurboUploaded" at the Status field at the Memopal Control Panel.

[7] We ran these tests with DropBox version 0.7.110.

attacker asking to upload a copy of the file, and observing whether deduplication occurs. Note that this is a rather limited query: First, the answer is a yes/no answer which does not detail who performed the upload of the file, or at what time. Moreover, in the basic form of the attack the attacker can only ask this query once – the query is asked by doing an upload of the file; afterwards the file is stored at the upload service and therefore the answer to the query will always be positive.

The latter limitation can be overcome by the following strategy, suggested to us by Adi Shamir: The attacker begins uploading a file, and observes whether deduplication occurs. If deduplication does not happen, and a full upload begins, then the attacker shuts down the communication channel and terminates the upload. As a result, the copy of the file owned by the attacker is not stored at the server. This enables the attacker to repeat the same experiment at a later time, and check again whether the file was uploaded. Furthermore, by applying this procedure at regular intervals, the attacker can find the time window in which the file is uploaded.

In the following sections we describe three attacks on online storage services. The first two enable an attacker to learn about the contents of files of other users, whereas the third attack describes a new covert channel.

2.1 Attack I: Identifying Files

This first attack allows identifying whether a specific file, known to the attacker, was previously uploaded to the storage service.

Assume that there is an attacker, Alice, who wants to learn information about Bob, a user of a cloud storage service. Then obviously, if Alice suspects that Bob has some specific sensitive file X which is unlikely to be at the possession of any other user, she can use deduplication to check whether this conjecture is true. All that Alice should do is try to backup a copy of X and check whether deduplication occurs.

As a specific example, assume that there is a file proving some illegal activity (e.g. a recording of a violent event, or a file with some stolen sensitive information, or material related to child pornography). Law enforcement authorities, once getting hold of a copy of this file, can upload the file to different cloud storage providers, and identify the storage services which store copies of the file. They can then ask for a court order that will require the service provider to reveal the identities of users who uploaded the file. (If the file is considered to be too sensitive to be uploaded for the purpose of identifying the users who posses it, then, as described above, the process of uploading the file by the authorities can be terminated at its beginning, immediately after identifying whether deduplication is applied to this file.)

2.2 Attack II: Learning the Contents of Files

The attack described above only allows to check whether a specific file is stored in the cloud storage service. However, the attacker might apply this attack to multiple versions of the same file, essentially performing a brute force attack over all possible values of the content of the file. Assume for example that Alice and Bob work in the same company, which uses a cloud backup service to backup the machines of all its employees. Once a year, all employees receive a new copy of a standard contract which contains their updated salary. Alice is curious to find Bob's new salary, which is most probably some multiple of $\$500$ in the range $\$50,000 - \$200,000$. All that Alice has to do is generate a template of Bob's contract, with Bob's name and the date of the new contract, and then generate a copy of the contract for each possible salary of Bob (a total of 301 files). She then runs a backup to the company backup service which she and Bob use. The single file for which deduplication occurs is the one containing Bob's actual salary.

This attack can be applied whenever the number of possible versions of the target file is moderate. It seems very relevant for a corporate environment where often files are small variations of standard templates. Consider for example the following three examples:

- An online banking service sends its customers a document containing their login name and their PIN, which is a 4 digit number. Alice can therefore generate $10,000$ documents with the login name "Bob" and all possible values of the PIN, and check which of these files has already been stored. This document corresponds to Bob's actual PIN. The same attack can be applied to arbitrary passwords if they are taken from a moderately sized domain. Note that unlike online dictionary attacks, the attacked banking service does not notice that someone is trying all potential passwords of a certain user.
- Suppose that a file detailing the results of some medical test of Bob is stored on his computer. Alice can use this attack to find the result of the test, which usually comes from a small domain (e.g., it is a yes/no answer for the occurrence of a genetic disease or for the result of a pregnancy test, or comes from a range of, say, a hundred likely values for a cholesterol test). The name of the referring physician, and the date of the referral, might be known to Alice or are likely to come from a small domain. Even the serial number of the test, if such a number exists, might be guessed by Alice if she has an example of a result of a test taken on a similar date.
- Suppose that both Alice and Bob participate in an auction, which requires bidders to submit their bids on some standard form containing their names and their bid (this is actually a common practice in many auctions and procurement processes). If Alice can speculate on, say, the $10,000$ most likely bid values of Bob, she can use the same attack to find Bob's actual bid and then set her bid accordingly.

2.3 Attack III: A Covert Channel

Suppose that Alice managed to install some malicious software on Bob's machine. Bob, however, runs a firewall which prevents unauthorized programs from connecting to the outside world. Even if such a firewall is not running, Alice

might want to hide the communication between the malicious software and its command and control server).

If Bob is using an online storage service which uses cross-user deduplication, then Alice can use the deduplication attack to establish a covert channel from the malicious software to a remote control center run by her. (The existence of a covert channel might be a second-order attack, and there might also be other ways of establishing a covert channel. Still, it is interesting to examine how a covert channel can be established by exploiting cross-user deduplication.)

Let us first describe how a single bit can be transferred: The software generates one of two versions of a file, X_0 or X_1, and saves it on Bob's machine. If it wants to transfer the message "0" then it saves the file X_0; otherwise it saves the file X_1. The files must be sufficiently random so that it is unlikely that any other user generates identical files. At some point in time (say, daily) Bob runs a backup and stores the file on the online storage service. Alice then performs a backup with the same service as Bob, and learns which of the files, X_0 or X_1, was previously stored, i.e., she learns what message was sent by the software.

The covert channel can be used to transfer arbitrarily long messages by having the software save more than a single file, and using more than two options for the contents of each file. A detailed performance analysis of this method is beyond the scope of this article.

We described here how the malicious software can send messages to its command and control center. The same technique can be used for sending messages in the opposite direction, if it is possible for the malicious software to examine the log files of backups and observe when deduplication takes place.

3 SOLUTIONS

Deduplication offers great savings to the providers of cloud storage services, and these savings translate to lower fees to the users of these services. On the other hand, deduplication introduces new security risks. Considering simple solutions that try to address these risks, one may try to limit the number of uploads permitted for a user per time window. While this approach may damage the experience of regular users, which have limited network connectivity, it does not prevent malicious users from writing scripts that repeat their attacks between the specified time windows. Developing more advanced solutions that try to model the user behavior identifying suspicious uploads is also not straightforward. Especially, since in the above described deduplication attacks the user does not have to fully upload the file and can abort the transmission in the middle, simulating network failures.

Below, we describe several practical solutions to the security risks of deduplication. The first solutions essentially prevent deduplication from taking place, and might therefore be unacceptable. The latter ones enable the usage of deduplication while limiting the scope of the security risks. The cost of applying any solution might be substantial. Assume, for example, that a basic deduplication approach saves 95% of the communication costs, and that deploying a certain solution reduces the saving to 93%. As a result, the communication costs are increased by 40% (from 5% to 7% of the raw communication overhead).

3.1 Using Encryption to Stop Deduplication

A simple solution to the risks described above is, of course, to stop using cross-user deduplication. Clients of cloud storage services can do so, regardless of the service they use, by encrypting their data before the local client software of the storage service operates on their files. (We stress that this encryption must be done with a personal key of the user, rather than by using a global key which is shared by all users of the system. Otherwise deduplication will still be possible, since, if the encryption function is deterministic, as is typically the case, the encrypted file is a deterministic function of the original file and of the encryption key. Therefore identical files result in identical encrypted versions of the file, which can still be deduplicated.)

When we tested the encryption options of the service providers that we examined, we identified several interesting observations. First, the default configuration lets the service itself, rather than the user, generate the encryption keys. When this option is used, it enables deduplication of copies of the same file uploaded by different users. This suggests that either the data is decrypted when it arrives at the servers, or that all files are encrypted with the same key. An alternative solution is to allow users to encrypt their data with their own private keys. Among the three services that we examined, the only service that currently supports this option is MozyHome[8]. The option of using personal encryption keys is costly to the service providers, not only because it prevents the usage of deduplication, but also due to potential complications caused by key management issues (e.g., supporting users who lose their keys).

It is interesting to note that the usage of a personal key is susceptible to offline dictionary attacks against that key. The attack is based on the observation (which holds for the personal key option of the MozyHome service) that if two users choose the same key then deduplication is applied between identical files stored by these two users. Suppose now that Alice knows that (1) Bob's key comes from some relatively small domain, and (2) a certain file was definitely uploaded by Bob, and by him alone. Then Alice can try to upload copies of this file encrypted with each possible value of Bob's key, and identify the right value of the key as the one for which deduplication occurs. This is a powerful dictionary attack which is very hard to identify by the attacked service.

[8]The Mozy service offers two methods of data encryption: 448-bit Blowfish encryption using a Mozy key, 256-bit AES encryption using the user's own private key. The latter option does not enable cross-user deduplication and therefore our analysis refers only to encryption with a key supplied by Mozy. This is the default option, and users who choose to use their own keys are shown several warning messages before this choice is applied. Furthermore, there is no option to allow a user who chose during initial setup to use a Mozy key, to change the setting later to usage of a private key.

The attack described above demonstrates that the usage of personal keys for encrypting data which might be deduplicated, is very sensitive to the usage of weak keys. Another related vulnerability is the following: Suppose that Alice somehow knows the personal key used by Bob (perhaps Bob, as many typical users, uses the same key or password for many different services, and Alice managed to get hold of the key through one of these services). Then by using the deduplication based attacks described above Alice can now identify whether a particular file was uploaded *by Bob*, rather than whether the file was uploaded by some user. This is done by Alice choosing to use the same personal key as Bob, and uploading a copy of the file.

It is also worth mentioning earlier research on deduplication of encrypted files that resulted in the notion of "convergent encryption", which produces identical ciphertexts from identical plaintext files, even if the files are encrypted using different keys [2]. (Each file is essentially encrypted using a key which is a hash of the contents of the file.) The goal of that work was to enable deduplication of encrypted files and it was later followed up by a work allowing deduplication of encrypted data chunks [10]. In the context of our examination the usage of convergent encryption does not solve the security risks since users are still able to identify the occurrence of deduplication.

3.2 Performing Deduplication at the Servers

It is possible to prevent clients from identifying the occurrence of deduplication by changing the backup software so that files are always uploaded, and the deduplication process is run at the servers (this is the target-based approach to deduplication). A major drawback of this approach is that it eliminates all bandwidth savings of deduplication, and that the service provider and/or the users must pay for transferring the raw amount of data. In order to estimate the cost of this solution we compared the pricing of the Amazon S3 service for data storage and transfer. As of the end of June 2010, the cost of transferring 1GB of data was between 157% to 216% of the cost of storing this data for a month (this comparison was done for each service tier offered by this service). This pricing suggests that cost of transferring data is about the same as the cost of storing it for 1.5-2 months.

An interesting tradeoff between bandwidth and privacy was introduced in MozyHome, after we notified them about the attacks described in this paper. In the current version of that system (as of June 2010), files of relatively small size are always uploaded, and source-based deduplication is only applied to larger files. This solution would be useful whenever the following two properties hold: (1) sensitive data is typically stored in small files (for which source-based deduplication is not applied), while there is almost no sensitive data related to large files; and (2) most of the bandwidth is consumed by uploading large files, and therefore the cost of uploading all copies of small files is tolerable.

Fig. 1. **A Randomized solution.** The figure details the operations performed upon the client backup request. For every file the server keeps a random threshold (T_x). If the number of existing copies of the document (C_x) is at least as high as the threshold, then client side deduplication is performed and the file content are not sent over the network. Otherwise, deduplication is performed only at the server side.

3.3 A Randomized Solution

The security risks of deduplication stem from the fact that deduplication of a file occurs if, and only of, this file was previously uploaded to the storage service. The risks can be reduced by weakening the correlation between deduplication and the existence of files in the storage service. This is done by assigning a random threshold for every file, and performing deduplication only if the number of copies of the file exceeds this threshold.

Before examining this solution in more detail, it is instructive to examine a similar approach, which is *insecure*: Here, the server sets a global threshold t (say, $t = 10$), and performs deduplication of a file only if at least t copies of the file were uploaded. In this case, indeed, uploading a single copy of the file by Alice does not reveal whether Bob has previously uploaded this file. However, Alice can upload many copies of the file (even using multiple user identities), and check whether deduplication occurs after t, or $t - 1$, copies of the file are uploaded by her. The latter case indicates that a copy of the file was previously uploaded by a different user. (We can safely assume that Alice knows the threshold t, since she can conduct simple experiments to reveal the value of t.)

The solution: Let us now describe the randomized solution in more detail. For every file X, the storage server assigns a threshold t_X chosen uniformly at random in a range $[2, d]$, where d is a parameter which might be public. (For example, assume that $d = 20$.) It is important that no one except for the server can compute t_X, even if the contents of X are known. One way of achieving this property is by the server choosing t_X at random and storing this value privately. Another method is the server using a secret key s, and computing the threshold as a function of the contents of the file, or of its hash value, and of the secret key s. Namely, computing $t_X = F(X, s)$. In this case there is no need to explicitly store the threshold of X since the server can easily recompute it.

Now, for every file X the server keeps a counter c_X of the number of *clients* which have previously uploaded copies of X. When a new copy of the file is uploaded, it is deduped at the client side if one of the following two conditions hold: (1) $c_X \geq t_X$, or (2) the copy is uploaded by a client that has previously uploaded X. Otherwise no deduplication occurs. Note that the minimal number of copies for which deduplication can occur is 2, since it is impossible to perform

deduplication when the first copy of the file is uploaded.

This solution hides the occurrence of deduplication from users during the first $t_X - 1$ times that X is uploaded, since the file is uploaded as if no copy of it is available at the server. However, once the data is transferred to the server side deduplication can be preformed. Thus, the overall disk space savings offered by this solution are exactly as in the basic deduplication scheme. The only penalty paid is that the bandwidth utilization is smaller, since $(t_X - 1) \cdot X$ more copies of the file are uploaded, compared to a plain deduplication solution. Figure1 illustrates the proposed solution and its data flow.

Handling deletions: File deletions must also be addressed. When a deletion occurs it seems natural to decrement the counter c_X of the number of copies of the file. This setting, however, enables the following attack: Alice uploads copies of the file X and notices that deduplication occurs after t copies are uploaded. She then repeatedly removes two copies of X from the online storage, and again uploads these two copies. If in one of these tests she notices that deduplication happens after only one of the two copies is uploaded, then it must be the case that some other user has just uploaded a copy of X. Similarly, if deduplication happens after uploading three, rather than two, copies of X, then another user must have removed a copy of this file.

This attack is not very practical, since online storage services typically retain copies of deleted files for some period of time after their deletion. For example, the policy of most services, including Mozy, DropBox, and Memopal, is to retain files for at least 30 days after their removal. Therefore, each iteration of the attack would take at least 30 days to execute.

A simple solution to the attack is the following amended policy: After the counter c_X reaches the threshold t_X, deduplication is always performed, regardless of whether deletions have occurred. The drawback of this solution is that deduplication must be performed even after all copies of the file are deleted. This means that once the number of copies reaches the threshold the server must keep a copy of the file even if is deleted. The implication of this policy in practice might not be too costly, since (1) copies of deleted files are already being retained for long periods of time, and (2) the policy is applied to relatively popular files, which have been uploaded at least t_X times, and therefore it is less likely that all copies of these files will be deleted.

Security: The randomized solution described above still enables an attacker to distinguish between the case that the file X has been uploaded by d users, and the case that no user uploaded X. Yet this sort of information tells about the popularity of the file but is unlikely to tell anything about any specific user, and therefore it is probably less important to protect against the release of this information.

We would like to show that the solution does not reveal too much information about the inclusion of any file X in the data stored by the server. In order to analyze security we compare the views of the attacker in two instances: one where the file X was already uploaded by another user, and one where no copy of X has previously been uploaded. Distinguishing between these two cases seems to be the most relevant for breaching the privacy of any single user. If we ensure that it is hard to distinguish between these two cases, then each user can be assured that uploading its copy of the file does not make a substantial difference to the view of the attacker, and therefore cannot be detected by it. Indeed, such a measure is used in the notion of *differential privacy* (see [4] and references within), which considers privacy issues in statistical queries to large databases.

To analyze privacy, we consider three types of events in the setting where the attacker wants to identify whether a (single) copy of a document was uploaded.

1) If the attacker uploads a single copy of X and finds out that deduplication occurs, then it immediately learns that $t_X = 2$ and that a copy of X was previously uploaded by another user.
2) If, on the other hand, the attacker must upload d copies of X, under different identities, before deduplication occurs, then it must be the case that $t_X = d$ and that no copy of X was previously uploaded.
3) If deduplication happens after the attacker uploads $2 < t < d$ copies of X, then this could be due to one of two events: (a) A copy of X was previously uploaded, and the threshold is $t_X = t + 1$. (b) No copy of X was previously uploaded, and the threshold is $t_X = t$. In any case, the probability that t_X was set to either t or $t+1$ is exactly $1/(d-1)$ and is independent of whether X was uploaded or not. It is easy to show now that the probability that X was previously uploaded, given the event that deduplication occurred after uploading t copies (where $2 < t < d$), is equal to the a-priori probability that X was previously uploaded. In other words, the occurrence of deduplication does not add any information about whether X was uploaded to the server or not.

Observe that if X was previously uploaded then the first event, which leaks information, happens with probability $1/(d-1)$ whereas the third event, which does not leak any information, happens with probability $1 - \frac{1}{d-1}$. If X was not previously uploaded then the second event happens with probability $1/(d-1)$ whereas the third event happens with probability $1 - \frac{1}{d-1}$. Since for any file only one of the two conditions holds, we get the following theorem:

Theorem 1: For a fraction of $1 - \frac{1}{d-1}$ of the files, the solution described above leaks no information which enables an attacker to distinguish between the case the a single copy of a file was previously uploaded, and the case that the file was not previously uploaded.

It is worthwhile to mention some enhancements of this basic result, as well as some notes about its performance:

- A similar analysis can be applied to find an upper bound on the probability with which the attacker can distinguish between the occurrence of c copies, and c' copies, of X, for any values of $1 \leq c < c' \leq d$. Security is better when

$c' - c$ is smaller.

- The cost of the solution is the following: As long as less than t_X copies of the files are uploaded, source-based deduplication does not occur. Namely, there are $t_X - 1$ unnecessary uploads of the file. Given statistics on the distribution of the number of copies of files, it is possible to compute the expected cost of the solution.
  The threshold t_X is chosen in the range $[2, d]$. Therefore if popular files have many more than d copies, then the expected cost is small compared to the benefit of using deduplication. A larger value of d results in a higher cost, but also provides better security since the probability of distinguishing whether a copy of X is present is at most $1/(d-1)$.
- Theorem 1 gives a result with an "all or nothing" flavor: it shows that for all but $\frac{1}{d-1}$ of the files *no* information is leaked, whereas for the remaining files, where the threshold is set to either $t_X = 2$ or $t_X = d$, the attacker can distinguish whether a copy of the file has been uploaded.
- It is possible to choose the threshold according to *different distributions* (other than the uniform distribution). The goal is to minimize the cost while maximizing security.
- Security against Attack III (covert channel to/from a malicious software) is a bit trickier. The reason is that the malicious software can place d copies of a file rather than one, and thus ensure that deduplication will always take place for this file. The randomized solution overcomes this issue by counting all copies of a file uploaded by a *single* user as one. In other words, the counter c_X is taken to be the number of users that hold file X. Dedup within a single user always occurs, while across users it occurs only if $c_X \geq t_X$. Under such rules, the analysis of the covert channel case is similar to that of the other attacks.

4 Summary and Conclusions

In this article we pointed out the potential risks of cross-user source based-deduplication. We described how such deduplication can be used as a side channel to reveal information about the contents of files of other users, and as a covert channel by which malicious software can communicate with the outside world, regardless the firewall settings of the attacked machine.

Since deduplication offers substantial savings in both disk capacity and network bandwidth, we suggested and analyzed a mechanism that provides higher privacy guarantees while slightly reducing bandwidth savings.

Future work includes a more rigorous analysis of the privacy guarantees provided by our mechanism and a study of alternative solutions that maximize privacy while having minimal influence on deduplication efficiency. Furthermore, our observations give motivation to an evaluation of the risks induced by other deduplication technologies, and of cross-user technologies in general. The goal must be to ensure clients that their data remains private, by showing that uploading their data to the cloud has a limited effect on what an adversary may learn about them.

5 Acknowledgments

The work of the second author was supported by the European Research Council (ERC) as part of the ERC project SFEROT. The authors would like to thank Adi Shamir and Dalit Naor for their useful suggestions.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, February 2009, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

[2] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," *Distributed Computing Systems, International Conference on*, vol. 0, p. 617, 2002.

[3] M. Dutch and L. Freeman, *Understanding data de-duplication ratios*, SNIA, February 2009, http://www.snia.org/forums/dmf/news/articles/SNIA_DeDupe_Ratio_Feb09.pdf.

[4] C. Dwork, "Differential privacy: a survey of results," in *TAMC'08: Proceedings of the 5th international conference on Theory and applications of models of computation*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–19.

[5] H. S. Gunawi, N. Agrawal, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and J. Schindler, "Deconstructing commodity storage clusters," in *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 60–71.

[6] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *Symposium on Operating Systems Principles*, 2001, pp. 174–187. [Online]. Available: http://citeseer.ist.psu.edu/muthitacharoen01lowbandwidth.html

[7] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage," in *First USENIX conference on File and Storage Technologies*, Monterey,CA, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.8085

[8] D. Russell, *Data Deduplication Will Be Even Bigger in 2010*, Gartner, February 2010.

[9] A. W. C. Stanley Zaffos, "Cloud storage: Benefits, risks and cost considerations," *Gartner, April 2009*.

[10] M. W. Storer, K. M. Greenan, D. D. E. Long, and E. L. Miller, "Secure data deduplication," in *StorageSS*, 2008, pp. 1–10.

[11] L. L. You, K. T. Pollack, and D. D. E. Long, "Deep store: An archival storage system architecture," in *In Proceedings of the 21st International Conference on Data Engineering (ICDE 05*. IEEE, 2005, pp. 804–815.