Side Channels in Cloud Services

Deduplication in Cloud Storage

Although deduplication is most effective when applied across multiple users, cross-user deduplication has serious privacy implications. Some simple mechanisms can enable cross-user deduplication while greatly reducing the risk of data leakage.

Danny Harnik
IBM Haifa Research Lab

Benny Pinkas
Bar Ilan University

Alexandra Shulman-Peleg
IBM Haifa Research Lab

As the volume of data increases, so does the demand for online storage services, 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 provided by the hosting company. *Cloud storage* refers to scalable and elastic storage capabilities delivered as a service using Internet technologies with elastic provisioning and use-based pricing that doesn't penalize users for changing their storage consumption without notice.[1,2]

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. As storage services transition 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 saves 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.[3] These savings also translate to lower fees for service users. Deduplication's effectiveness depends on such factors as the type of data, the retention period, and the number of users. The percentage of space reduction is calculated as 100 percent less the inverse of the space reduction ratio. So, even a deduplication ratio of 1:3 results in a 66 percent saving. Reported deduplication ratios in common business settings range from 1:10 to 1:500, resulting in disk and bandwidth savings of more than 90 percent.[4]

However, there's an inherent risk in entrusting data to the storage cloud. In doing so, the data owner releases control over the data. Yet, a wide range of users and applications are more than willing to hand over their data storage tasks to cloud providers. They put their trust in the cloud provider's integrity and in the security of its access control mechanisms.

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 serve as a side channel that reveals information about the contents of other users' files. Deduplication can also serve as a covert channel through which malicious software can communicate with a command-and-control center, regardless of any firewall settings at the attacked machine.

We analyze deduplication's security issues and propose a simple mechanism that allows cross-user deduplication while reducing the risk of data leakage. Our mechanism states rules by which deduplication can be artificially turned off. This simple practice gives clients a guarantee that adding their data to the cloud has a limited effect on what an adversary might learn about this data. Thus, we can essentially assure clients of the privacy of their data.

Data Deduplication Strategies

We categorize data deduplication strategies according to the basic data units they handle. In this respect, there are two main data deduplication strategies:

- File-level deduplication is a popular type of service in which only a single copy of each file is stored.[5,6]

Two or more files are considered identical if they have the same hash value.
- Block-level deduplication 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.[8,9]

The discussion in this article can be applied to both strategies.

The architecture of a deduplication solution might follow two basic approaches. In the *target-based* approach, the target data storage device or service handles deduplication, and the client is unaware of any deduplication that might occur. This technology improves storage utilization, but doesn't save bandwidth. *Source-based* deduplication, on the other hand, acts on the data at the client before it's transferred. Specifically, the client software communicates with the backup server (by sending hash signatures) to check for the existence of files or blocks. It replaces duplicates with pointers and never sends the actual duplicate data over the network. This approach improves both storage and bandwidth utilization.

Security issues

The attacks we describe can occur in deduplication performed at either the file or block level. Here, we'll assume that deduplication is performed at the file level. There are, however, two features of the deduplication service that are crucial for the attacks.

The first is *source-based deduplication*. That is, deduplication must be performed at the client side. As mentioned earlier, 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"). 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 the software provides this type of report.

The second feature that's 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 because 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 this to be true even for private customers: almost every common software manual or media file that we tried to back up using popular backup services was already available on the servers and was therefore deduped. Note that these are huge files, so deduplication offers great savings to service providers.)

We performed the following test to identify services that perform source-based cross-user deduplication (the test can be repeated by any reader, on the storage service of the reader's choice):

- We installed the service's client software on two different computers and created two different user accounts.
- Next, we used one account to upload a file (our tests used Sun's VirtualBox software, which is almost 73 Mbytes).
- We then used the second account to upload the same file, checking whether it was indeed uploaded.

If the file wasn't retransmitted over the network, we concluded that the backup service performed source-based, cross-user deduplication. (In fact, when checking popular storage services, we don't need to use two accounts because any popular file found on the Web will likely exist on the servers. Therefore, the test can consist of downloading a popular file from the Web, uploading it to the service, and checking for deduplication.) We identified the following services that perform cross-user, source-based deduplication:

- DropBox, a popular file-sharing and backup service that crossed the 4 million user milestone in 2010 (see http://blog.dropbox.com/?p=339);
- MozyHome, which provides online backup for more than 1 million customers and 50,000 business users, storing more than 25 Pbytes (see http://mozy.com/news/releases/comcast-launches-secure-backup-share-online-storage-solution-for-its-internet-customers); and
- Memopal, which was ranked by Backup Review as the best online backup service in Europe, with almost 1,000 new subscribers per day (see www.memopal.com/en/pressrelease/memopal-online-backup-ranked-as-the-best-online-backup-serv.htm).

Most vendors don't try to hide the fact that they use deduplication. Our tests easily detected this fact by checking the history or log file; reading the upload status message, which differs between uploaded and deduplicated files; checking the upload speed to see if a file upload completes within a time that's much shorter than the time required by the client machine's upload bandwidth; or monitoring network traffic and measuring the amount of transmitted data (this is the most generic deduplication detection method). Most services have additional client-server communication traffic, but it's negligible compared to the large volumes of data transmitted when uploading large files.

A storage service that supports source-based, client-side, deduplication essentially serves as an oracle, answering the query, "Did any user previously upload a copy of this file?" An attacker can answer

this query by asking to upload a copy of a file and observing whether deduplication occurs. This is a rather limited query. First, the answer must be yes or no, which doesn't detail who has previously uploaded the file or when. Moreover, in the attack's basic form, the attacker can only ask this query once, because the attacker must send the query by uploading the file, which is then stored at the upload service. Consequently, the answer to any future query will always be positive.

If there's no deduplication, and a full upload begins, the attacker shuts down the communication channel and terminates the upload.

The latter shortcoming 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 there's no deduplication, and a full upload begins, the attacker shuts down the communication channel and terminates the upload. Consequently, the attacker's copy of the file isn't stored at the server. The attacker can thus repeat the same experiment later. Furthermore, by applying this procedure at regular intervals, the attacker can find the time window in which the file is uploaded.

We describe three attacks on online storage services. The first two let an attacker learn about the contents of other users' files, and the third attack describes a new covert channel.

Attack I: Identifying Files

Assume an attacker Alice wants to learn information about Bob, a cloud storage service user. Obviously, if Alice suspects that Bob has some specific sensitive file X that's unlikely to be in the possession of any other user, she can use deduplication to check whether this conjecture is true. All Alice needs to do is try to back up a copy of X and check whether deduplication occurs.

More specifically, assume there's a file proving some illegal activity (for example, a recording of a violent event, a file with stolen sensitive information, or material related to child pornography). Once law enforcement authorities get a copy of this file, they can upload it to different cloud storage providers and identify the storage services storing copies of the file. They can then request a court order that will require the service provider to reveal the identities of users who uploaded the file. (If the file is considered too sensitive to be uploaded for the purpose of identifying the users who possess it, the authorities can terminate the upload process immediately after identifying whether deduplication is applied to this file.)

Attack II: Learning the Contents of Files

Attack I only lets the attacker 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 file contents. Assume, for example, that Alice and Bob work in the same company, which uses a cloud backup service to back up all of its employees' machines. Once a year, all employees receive a new copy of a standard contract containing their updated salary. Alice wants to know Bob's new salary, which is probably some multiple of $500 in the $50,000 to $200,000 range. All 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 (a total of 301 files). She then runs a backup to the company backup service that she and Bob use. The single file for which deduplication occurs is the one with Bob's actual salary.

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

An online banking service sends its customers a document containing their login name and PIN, which is a four-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. Alice can apply the same attack to arbitrary passwords if they're taken from a moderately sized domain. Unlike online dictionary attacks, the attacked banking service doesn't notice that someone is trying all potential passwords of a certain user.

Second, suppose Bob stores a file detailing the results of some medical test of his on his computer. Alice can use this attack to find the test result, which usually comes from a small domain (for example, a yes/no answer for the occurrence of a genetic disease or the result of a pregnancy test, or within a range of, say, 100 likely values for a cholesterol test). Alice might know the referring physician's name and the date of the referral, or they're likely to come from a small domain. Alice might even guess the test's serial number, if such a number exists, if she has an example of a result of a test taken on a similar date.

Finally, suppose both Alice and Bob participate in an auction that requires bidders to submit their bids on some standard form containing their name and bid (this is common practice in many auctions and procurement

processes). If Alice can speculate on, say, Bob's 10,000 most likely bids, she can use the same attack to find Bob's actual bid and then set her bid accordingly.

Attack III: A Covert Channel

Suppose Alice installed some malicious software on Bob's machine. Bob, however, runs a firewall that prevents unauthorized programs from connecting to the outside world. Even if such a firewall isn't running, Alice might wish to hide the communication between the malicious software and its command-and-control server.

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

We 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," it saves the file X_0; otherwise, it saves the file X_1. The files must be sufficiently random so that it's 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—that is, she learns what message the software sent.

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

If the software can examine the log files of backups and observe when deduplication occurs, Alice can use the same technique to send messages in the opposite direction—namely, to the malicious software.

Solutions

Deduplication offers great savings to the providers of cloud storage services, and these savings are translated to lower fees charged to the users of these services. On the other hand, deduplication introduces new security risks. A simple solution that addresses the security risks of deduplication is limiting the number of uploads a user is permitted per time window. This approach might damage the experience of regular users, who have limited network connectivity, and yet it doesn't prevent malicious users from writing scripts that repeat their attacks between the specified time windows. Developing more advanced solutions that try to model user behavior to identify suspicious uploads is also not straightforward. For example, in the deduplication attacks we've described, the user doesn't have to fully upload the file and can abort the transmission in the middle, simulating a network failure. Below, we describe several practical solutions to deduplication's security risks. The first solution essentially prevents deduplication from taking place, so it might be unacceptable. The latter solutions allow deduplication while limiting the security risks' scope.

The cost of applying any solution might be substantial. Assume, for example, that a basic deduplication approach saves 95 percent of the communication costs, and that deploying a certain solution reduces the saving to 93 percent. Consequently, the communication costs are increased by 40 percent (from 5 to 7 percent of the raw communication overhead).

Using Encryption to Stop Deduplication

A simple solution to the risks we've described is, of course, to stop using cross-user deduplication. Clients of cloud storage services, regardless of the service they use, can stop using deduplication by encrypting their data before the storage service's client software operates on their files. (Users must perform this encryption using their personal key, rather than a global key, which all system users share. Otherwise, deduplication is still possible. After all, 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 testing the encryption options of the examined service providers, we made several interesting observations. All services encrypt the client-server communication channel, typically using SSL. Still, this encryption does not affect deduplication because all information can be decrypted at the server side. As for data encryption (rather than channel encryption, which is done using SSL), the default configuration lets the service, and not the user, generate the encryption keys. Deduplication of copies of the same file uploaded by different users occurs even when data encryption is used under this default configuration. This fact 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 let users encrypt their data with their own personal keys. The option of using personal encryption keys is costly to service providers, not only because it prevents them from using deduplication, but also because key-management issues introduce potential complications (for example, supporting users who lose their keys). Among the three services we examined, only MozyHome currently supports

this option. The MozyHome service offers two methods of data encryption: 448-bit Blowfish encryption using a Mozy key and 256-bit AES encryption with the user's own personal key. The latter option doesn't enable cross-user deduplication, so our analysis in this article refers only to encryption using a Mozy-supplied key. Users who choose to encrypt their data with their own keys receive several warning messages before the system applies this choice. Furthermore, users who choose to use a Mozy key during initial setup don't have the option of changing the setting later to usage of a personal key.

Encryption with a personal key might prevent deduplication but it's also susceptible to offline dictionary attacks, which might reveal the personal key. This type of attack is based on the observation (which holds for the MozyHome service's personal key option) that if two users choose the same key, deduplication is applied between identical files stored by these two users. Suppose now that Alice knows that Bob's key comes from a relatively small domain and that Bob, and Bob alone, uploaded a certain file. 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 that's hard to identify by the attacked service. Thus, the use of personal keys for encrypting data that might be deduplicated is sensitive to the use of weak keys.

In addition, suppose that Alice somehow knows the personal key used by Bob (perhaps Bob, like many users, uses the same key or password for many different services, and Alice got hold of the key through one of these services). Then, by using the deduplication-based attacks we've described, Alice can identify whether Bob uploaded a particular file, rather than whether some user uploaded the file. This is done by Alice using the same personal key as Bob, and uploading a copy of the file.

Earlier research on deduplication of encrypted files resulted in the notion of *convergent encryption*, which produces identical ciphertexts from identical plaintext files, even if the files are encrypted using different keys.[6] (Each file is essentially encrypted using a key that's a hash of the file contents.) That work aimed to enable deduplication of encrypted files. It was later followed by work allowing deduplication of encrypted data chunks.[10] In the context of our examination, the use of convergent encryption doesn't solve the security risks because users can still identify the occurrence of deduplication.

Performing Deduplication at the Servers

A cloud storage provider can 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). However, this approach eliminates all of deduplication's bandwidth savings, and the service provider and/or the users must pay for transferring the raw amount of data. To estimate this solution's cost, we compared the pricing of Amazon's S3 service for data storage and transfer. As of the end of June 2010, the cost of transferring 1 Gbyte of data was between 157 and 216 percent of the cost of storing this data for a month (we performed this comparison for each service tier offered by this service). This pricing suggests that the cost of transferring data is about the same as the cost of storing it for 1.5 to 2 months.

MozyHome introduced an interesting trade-off between bandwidth and privacy after we notified them about the attacks described here. In MozyHome's current version (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. First, 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. Second, uploading large files consumes most of the bandwidth, and therefore the cost of uploading all copies of small files is tolerable.

A Randomized Solution

The security risks of deduplication stem from the fact that it occurs if, and only if, the file was previously uploaded to the storage service. Weakening the correlation between deduplication and the existence of files in the storage service can reduce the risks. We can weaken this correlation 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 detail, we 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 have been uploaded. In this case, Alice's uploading of a single copy of the file doesn't reveal whether Bob previously uploaded this file. However, Alice can upload many copies of the file (even using multiple user identities) and check whether deduplication occurs after she uploads t or $t - 1$ copies of it. The latter case indicates that a different user uploaded a copy. (We can safely assume that Alice knows the threshold t because she can conduct simple experiments to reveal the value of t.)

In our randomized solution, 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 that might be public (for example, assume $d = 20$). No one but the server should be able to compute t_X, even

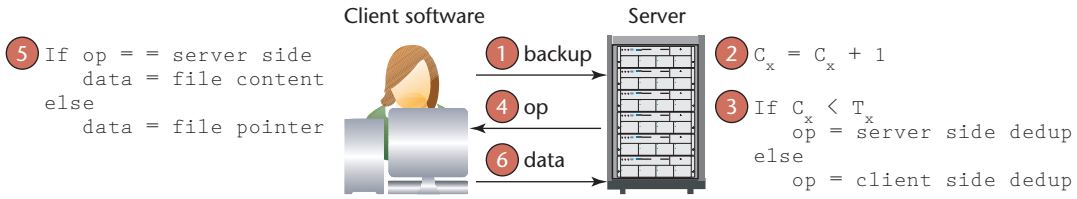

Figure 1. A randomized solution. The figure details the operations performed when a client makes a backup request. The server keeps a random threshold (t_X) for every file. If the number of existing copies of the document (c_X) is at least as high as the threshold, the system performs client-side deduplication and doesn't send the file content over the network. Otherwise, deduplication is performed only at the server side.

if the contents of X are known. One way to achieve this property is to have the server choose t_X at random and store this value privately. Alternatively, the server could use a secret key s and compute the threshold as a function of the file contents, or of the file's hash value, and of the secret key s—namely, computing $t_X = F(X, s)$. In this case, there's no need to explicitly store the threshold of X because the server can easily recompute it.

Now, for every file X, the server keeps a counter c_X of the number of clients that have previously uploaded copies of X. When a new copy of the file is uploaded, it's deduped at the client side if $c_X \geq t_X$ or if the copy is uploaded by a client that has previously uploaded X. Otherwise, no deduplication occurs. Note that the minimum number of copies for which deduplication can occur is 2, because it's 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 because 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, the system can perform deduplication. Thus, the overall disk space savings offered by this solution are the same as in the basic deduplication scheme. The only penalty is that the bandwidth utilization is smaller because $(t_X - 1) \cdot X$ more copies of the file are uploaded, compared to a plain deduplication solution. Figure 1 illustrates the proposed solution and its data flow.

Handling deletions. 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 occurs after only one of the two copies is uploaded, she knows that some other user must have just uploaded a copy of X. Similarly, if deduplication occurs after she uploads three, rather than two, copies of X, another user must have removed a copy of the file.

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

The following amended policy offers a simple solution to the attack. 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 the system must perform deduplication even after all copies of the file are deleted. Thus, once the number of copies reaches the threshold, the server must keep a copy of the file even if it's deleted. In practice, this policy's implications might not be too costly, because services already retain copies of deleted files for long periods of time, and because the policy is applied to relatively popular files that have been uploaded at least t_X times, and therefore it's less likely that all copies of these files will be deleted.

Security. The randomized solution still lets an attacker distinguish between the case in which a file X was uploaded by d users, and the case in which no user uploaded X. Yet, this sort of information tells about the file's popularity but is unlikely to tell anything about any specific user, and therefore it's probably less important to protect against this information's release.

To show that this solution doesn't reveal too much information about the inclusion of any file X in the data stored by the server, we compare the attacker's views in two instances. In the first, another user has already uploaded the file X; in the second, no copy of X has previously been uploaded. Distinguishing between these two cases seems to be most relevant for breaching a single user's privacy. If we ensure that distinguishing between these two cases is difficult, each user can be assured that uploading its copy of the

file doesn't substantially affect the attacker's view, and so can't be detected by it. Indeed, *differential privacy*, which considers privacy issues in statistical queries to large databases, uses such a measure.[11]

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

In the first, the attacker uploads a single copy of X and finds that deduplication occurs. It therefore immediately learns that $t_X = 2$ and that a copy of X was previously uploaded by another user. In the second type of event, the attacker must upload d copies of X under different identities before deduplication occurs. It therefore knows that $t_X = d$ and no copy of X was previously uploaded.

Third, if deduplication occurs after the attacker uploads $2 < t < d$ copies of X, one of two events could have occurred: either a copy of X was previously uploaded, and the threshold is $t_X = t + 1$, or 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. It's now easy to show 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 doesn't add any information about whether X was uploaded to the server.

Observe that under the condition that X was previously uploaded, the first event, which leaks information, occurs with probability $1/(d - 1)$, whereas the third event, which doesn't leak any information, occurs with probability $1 - 1/(d - 1)$. If X wasn't previously uploaded, the second event occurs with probability $1/(d - 1)$, whereas the third event occurs with probability $1 - 1/(d - 1)$. Because for any file only one of the two conditions holds, we get the following theorem:

Theorem 1. For a fraction of $1 - 1/(d - 1)$ of the files, the solution we've described leaks no information that lets an attacker distinguish between the case in which a single copy of a file was previously uploaded and the case in which the file wasn't previously uploaded.

We can apply a similar analysis 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 value of $1 \le c < c' \le d$. Security is better when $c' - c$ is small.

This solution has the following cost. As long as fewer than t_X copies of the file are uploaded, source-based deduplication doesn't occur. Namely, there are $t_X - 1$ unnecessary uploads of the file. Given statistics on the distribution of the number of copies, we could compute the solution's expected cost. We assume that the threshold t_X is chosen in the range $[2, d]$. Therefore, if popular files have many more than d copies, the expected cost is small compared to the benefit of using deduplication. A larger d value results in a higher cost, but it also provides better security because the probability of determining if a copy of X is present is at most $1/(d - 1)$. Different systems could choose the threshold according to different distributions (other than the uniform distribution). The goal is to minimize cost while maximizing security.

Theorem 1 gives a result with an "all or nothing" flavor. It shows that for all but $1/(d - 1)$ 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 find whether a copy of the file has been uploaded.

Security against Attack III (covert channel to and from malicious software) is a bit trickier. The malicious software can place d copies of a file rather than one, thus ensuring 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, it considers the counter c_X to be the number of users that hold file X. Deduplication within a single user always occurs, while across users it occurs only if $c_X \ge t_X$. Under such rules, the analysis of the covert channel case is similar to that of the other attacks.

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 minimally influencing deduplication efficiency. Furthermore, our observations motivate an evaluation of the risks induced by other deduplication technologies, and of cross-user technologies in general. The goal must be to assure clients that their data remains private by showing that uploading their data to the cloud has a limited effect on what an adversary can learn about them. □

Acknowledgments

The European Research Council supported Benny Pinkas's work as part of the ERC project SFEROT. We thank Adi Shamir and Dalit Naor for their useful suggestions. Benny Pinkas performed most of his work reported here while at the University of Haifa.

References

1. A.W.C. Stanley Zaffos, "Cloud Storage: Benefits, Risks and Cost Considerations," Gartner, Apr. 2009.
2. M. Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," tech. report UCB/EECS-2009-28, Electrical Eng. and Computer Science Dept.,

Univ. Calif., Berkeley, 2009; www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.
3. D. Russell, "Data Deduplication Will Be Even Bigger in 2010," Gartner, Feb. 2010.
4. M. Dutch and L. Freeman, "Understanding Data Deduplication Ratios," data management forum report, Storage Networking Industry Assoc. (SNIA), 2009; www.snia.org/forums/dmf/news/articles/SNIA_DeDupe_Ratio_Feb09.pdf.
5. H.S. Gunawi et al., "Deconstructing Commodity Storage Clusters," *Proc. 32nd Ann. Int'l Symp. Computer Architecture* (ISCA 05), IEEE CS Press, 2005, pp. 60–71.
6. J.R. Douceur et al., "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," *Proc. Int'l Conf. Distributed Computing Systems*, IEEE CS Press, 2002, pp. 617–630.
7. S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Storage," *Proc. 1st Usenix Conf. File and Storage Technologies* (FAST '02), Usenix Assoc., 2002, pp. 89–101.
8. A. Muthitacharoen, B. Chen, and D. Mazieres, "A Low-Bandwidth Network File System," *Proc. Symp. Operating Systems Principles* (SOSP 01), ACM Press, 2001, pp. 174–187.
9. L.L. You, K.T. Pollack, and D.D.E. Long, "Deep Store: An Archival Storage System Architecture," *Proc. 21st Int'l Conf. Data Eng.* (ICDE 05), IEEE Press, 2005, pp. 804–815.
10. M.W. Storer et al., "Secure Data Deduplication," *Proc. 4th ACM Int'l Workshop Storage Security and Survivability* (StorageSS), ACM Press, 2008, pp. 1–10.
11. C. Dwork, "Differential Privacy: A Survey of Results," *Proc. 5th Int'l Conf. Theory and Applications of Models of Computation* (TAMC 08), Springer-Verlag, 2008, pp. 1–19.

Danny Harnik *is a research staff member at the IBM Haifa Research Lab. His research interests include storage systems, cryptography, and security. Harnik has a PhD in computer science from the Weizmann Institute, Israel. Contact him at dannyh@il.ibm.com.*

Benny Pinkas *is an associate professor in the Department of Computer Science at Bar Ilan University. His research interests include cryptography, computer security, and privacy. Pinkas has a PhD in computer science from the Weizmann Institute, Israel. He is a member of the International Association for Cryptologic Research, IEEE, and the ACM. Contact him at benny@pinkas.net.*

Alexandra Shulman-Peleg *is a research staff member at the IBM Haifa Research Lab. Her research interests include storage systems and performance management and currently focus on cloud storage solutions. Shulman-Peleg has a PhD in computer science from Tel Aviv University. She is a member of the Storage Networking Industry Association. Contact her at shulmana@il.ibm.com.*