A note on Dropbox security

Here, we explain some tools and ideas from crypto that, in our opinion, will give an online file storing users better security but will still maintain the deduplication functionality (which is one of the main selling points of Dropbox).

Scheme

The scheme is as follows: The first time the user sign up with dropbox, a local dropbox client asks for a username and password to authenticate the user to the dropbox server. Note that this is completely independent of the keys/passwords needed for the crypto that follows. This is only to "register" or authenticate the user to Dropbox (for things like billing, keeping an account of the user's usage etc.)

Once the user sets up a username and password, the client locally creates a strong password using a Password Based Key Derivation Function (PBKDF2). Let us call this key, $\texttt{key}_{\text{user}}$. This key is *not stored in the cloud.*

Also declare two hash functions $h_1(x)$ and $h_2(x)$ that are public and a good approximation of an ideal hash function (they're referred to as *random oracles* in crypto literature). In practice, $h_1(x) \equiv \texttt{SHA1}(1 \mid\mid x)$ and $h_2(x) \equiv \texttt{SHA1}(2 \mid\mid x)$. Informally, $h_1(\cdot)$ is used to construct a deterministic encryption scheme where the key depends on the message. And $h_2(\cdot)$ is used in a manner largely identical to how presently Dropbox uses a hash function to quickly look up key-value pairs in a large database. As you will see later, it is crucial that we use different hash functions.

Encrypting files

Let $f_1, \ldots, f_n$ be the list of files that the client is going to upload to dropbox. He computes $\texttt{Enc}(h_1(f_i), f_i)$ to be the encryption of the i^{th} file under the key derived from the file using the hash $h_1(\cdot)$. Let his local table comprise of the following pairs:

$$T := \{\langle h_1(f_i), h_2(f_i)\rangle\}_{i=1}^{n}.$$

The first corresponds to the decryption key and the second corresponds to the lookup value.

Now, he encrypts the table T under his *own* key. In other words, the user computes $\texttt{Enc}(\texttt{key}_{\text{user}}\ , T)$. Note that $\texttt{key}_{\text{user}}$ is derived only from the password and independent of Dropbox or the files the user uploads to Dropbox.

We note the following: Every file is encrypted identically if it hashes to the same value. This ensures that the dedup functionality is preserved. Also, only someone who knows $\texttt{key}_{\text{user}}$ can reconstruct the table T offline (i.e., when the user is not online and actively in the process of uploading his files).

Uploading and syncing files

The way to interact with dropbox now is fairly straightforward. Upload $\texttt{Enc}(h_1(f_i), f_i)$ using the hash $h_2(f_i)$ as is done normally with files today. If f_i were already uploaded by a different users, Dropbox already has $\texttt{Enc}(h_1(f_i), f_i)$ from the user and can check to see if $h_2(f_i)$ is in some master database that Dropbox maintains. Dropbox, for dedup does not need to store *which* user has uploaded $h_2(f_i)$. The user also uploads $\texttt{Enc}(\texttt{key}_{\text{user}}, T)$ that is associated with the username-password to access the account (again this password could be completely independent of the password used to derive $\texttt{key}_{\text{user}}$).

When the user wants to sync across multiple devices, he retrieves $\texttt{Enc}(\texttt{key}_{\text{user}}, T)$ by authenticating himself to Dropbox. He can then recover $h_1(f), h_2(f)$ for all his files f. By locally computing $h_2(f)$ he knows which files he does not have locally and has to retrieve from Dropbox. This is done as follows: Send across $h_2(f)$. Dropbox retrieves from its master database $\langle \texttt{Enc}(h_1(f), f\rangle$ and sends it back. Now, the user knows $h_1(f)$ and can decrypt and restore his file f. Syncing is thus complete.

Security

Now, let's look at the security of the above scheme. What does Dropbox have: A database with key-value pairs $\langle h_2(f), \texttt{Enc}(h_1(f), f)\rangle$. Dropbox also has $\texttt{Enc}(\texttt{key}_{\text{user}}, T)$ corresponding to each user. Since the user is the only person who has $\texttt{key}_{\text{user}}$, and $\texttt{key}_{\text{user}}$ is never stored remotely, this does not leak any information about *what* files the user has uploaded remotely to dropbox.

It is secure against an adversary trawling the website for specific files because the adversary can only conclude if *someone* has uploaded the file, among the millions of Dropbox users. It is also secure against subpoenas because without involving the user himself, one cannot retrieve $\texttt{key}_{\text{user}}$. Therefore, unless a subpoena is issued to an individual forcing him to reveal his secret key, subpoenaing dropbox gives no additional information.

Finally, for non-dedup documents, although it requires some work, one can show that $\texttt{Enc}(h_1(x), x)$ does not leak any information about x to someone who doesn't know x. Therefore, tax documents and other sensitive information are secure against adversaries that target the dropbox database directly.

Caveats and Questions

There are several caveats. Firstly, this assumes that Dropbox does not behave maliciously as a company. This is either through the client or the service. The client could be rigged to leak $\texttt{key}_{\text{user}}$ but this largely falls back to old questions raised in the famous essay "Reflections on Trusting Trust", by Ken Thompson. Dropbox could log all accesses and effectively reconstruct the value of T. However, this logging must be explicitly mentioned in the ToS and the users have the option to demand that Dropbox stop logging this information as it is not required for the current functionality. If Dropbox insists on logging for system or database performance issues, this situation is analogous to Google logging IP addresses for queries. Appropriate actions must be taken to de-anonymize and store the logs for as short a period as required (could be a few days to a few months time).

Secondly, this does not protect against an *online* adversary who colludes with Dropbox. An agent who requires that Dropbox reveals the ownership of a particular file (to check for copyright infringement, among other reasons) can wait and monitor users logging in and files requested online. This allows the adversary to test whether or not a particular file belongs to a particular user's account (assuming the user performs some action relating to this file—either uploading or syncing to a different device).

Thirdly, it is not yet known with this scheme how to ensure that the client does not misbehave in reporting to Dropbox the file sizes and therefore the current usage of the user's quota. In particular, deletions pose a problem and places the onus of honestly reporting the changes to the Dropbox server.

Fourthly, Dropbox's master database must have a `refcount` to ensure that files that have been deleted by all users can be safely deleted from the database. This `refcount` will probably be a distribution where a small fraction of popular files have `refcounts` in the thousands or millions but most files will have a very low `refcount` of perhaps 1 or 2 (i.e., there is a very long tail). It is yet unclear whether an adversary who gets snapshots of the `refcounts` from the database can conclude some information about the files and the users accessing the files.

And finally, there requires additional work using so called *proofs of retrievability* of digital files to ensure that applications like DropShip do not allow users to maliciously claim ownership of files (which can therefore lead to a new method of P2P file sharing). This requires the user to prove to Dropbox (interactively) that he owns the file by replying to the server's challenges for hashes on random segments of the file. This does not require Dropbox to get access to the file f itself, because it suffices for the user to provide a proof of retrievability of $\texttt{Enc}(h_1(f), f)$. Distributing the encryption of the file is as cumbersome as distributing the file itself, so this does not provide DropShip-like applications any leverage.

There are also several questions outside the scope of cryptography. If Dropbox decides to go about implementing these features, what are the engineering challenges required to smoothly transition from the current scheme to the new one. Are there ways to go about this in a way that makes the new infrastructure insecure (along the lines of how a man-in-the-middle attack during session setup can compromise the entire SSL session).

Acknowledgements

Most of these ideas have been floating around but I thought it would be useful to pen them down in one place. Credit to Jonathan Mayer and Mike Hamburg for introducing the problem. Many thanks to Sriram Keelvedhi, Vimal Jeyakumar, Dan Boneh, and the folks frequenting Hacker News for discussing the problems and the schemes presented here.

(This is a working draft and I welcome suggestions and comments)