FogVault: a (partial) solution for transparent client-side cloud storage encryption

Ben Yuan, Megan O'Leary, Ivan Ferreira Antunes Filho

Department of Electrical Engineering and Computer Science

Massachusetts Institute of Technology

Introduction

Cloud storage solutions are widely adopted today by individuals and organizations in many practical use cases, like file synchronization, data backup, and file sharing. By delegating the responsibility of storage to a third party, users are less affected by hardware costs and disk failures, incur less administrative and technical overhead because of the reduced need for self-hosted storage solutions, and are less reliant on inefficient file transfer methods like email and physical media. For most users, a cloud-based solution appears at first glance to provide an ideal blend of convenience, reliability, and accessibility.

However, security-conscious entities will observe that transfer of responsibility for storage also establishes a new and concerning trust relationship. Since a cloud storage provider must provide files on demand to its users, it must by corollary be able to read the contents of files that users store. For many classes of users – enterprises with valuable trade secrets, research labs dealing with controlled information, individuals in politically turbulent regions – such a concession is unacceptable, as no guarantee can be provided that the storage provider will not act against the interests of their users. For instance, a rogue storage provider may choose to divulge stored information to domestic or foreign intelligence agencies, or may pilfer trade secrets for its own use or for sale to other parties. Data encryption is an accepted solution to this problem, but is often cumbersome to use, poorly integrated with the storage solution being used, or otherwise poorly implemented.

For a cloud-based data encryption system to be considered usable, it must accomplish its primary goal of isolating the cloud storage provider from the plaintext data while posing no unnecessary hindrances to the user. For such a system to itself be trustworthy, it must stand up to close examination from anyone who doubts its effectiveness.

We attempted to build a solution that accomplishes these goals, providing transparent encryption and filesystem synchronization in a single open-source solution. In particular, we were able to realize the following goals with our work:

- Open-source application using open-source libraries.
- Direct integration with the Dropbox API, to avoid dependency on the Dropbox client.
- (theoretically) Cross-platform support using Qt, a respectable open-source framework.

- Basic functionality implemented without any trusted third party.
- Runs in the ‘background’ with minimal configuration.

We were not able to accomplish many of our more ambitious goals, but we were able to at least validate the basic principles we were attempting to achieve.

What we built

FogVault v0.08.0 implements the following features:

- A cryptographic library that supports verifiable multi-user asymmetric-key file cryptography and provides the cryptographic processing and validation operations required to achieve our desired security guarantees.
    - User key pairs using elliptic-curve cryptography for signing and ECDH.
    - File and filename encryption/decryption and integrity checking, with unique per-file symmetric keys that are themselves protected per-user by ECDH.
- A filesystem synchronization capability that pushes local updates to Dropbox, processes remote updates from Dropbox, and maintains the data integrity properties required to achieve our desired security guarantees.
    - Uploading of created and modified files to Dropbox in encrypted form with filename encryption on files and directories.
    - Downloading of remotely created and modified files from Dropbox, validating changes, and populating the local directory.
- A user frontend application that allows a user to manage user keys and operates the filesystem synchronization capability, allowing the application to achieve its user goals.
    - Generate, import, and export user keys (to support setting up multiple machines).
    - Display status information about the program.
    - Run the filesystem synchronization in the background.

In its current state, FogVault is <u>not</u> suitable for production use. Further testing and refinement is most certainly needed.

The threat model

The core system design of FogVault v0.08.0 operates in the following threat model:

- There are two classes of adversary that FogVault attempts to address: Dropbox itself, and anyone capable of monitoring network traffic between Dropbox and the FogVault client. We do not attempt to address the local-access problem.
- We assume that Dropbox, when it chooses to be trustworthy, provides the following reliability guarantees:
    - If a file is uploaded to Dropbox, it is stored exactly as received. Dropbox does not lose files.
    - If a file is requested from Dropbox, it is served exactly as stored; Dropbox does not modify stored files.

    - Dropbox serves the latest versions of any files requested, provided that the requesting user has authorization in the Dropbox permission model to read these files.
- We recognize that Dropbox must necessarily have access to the contents of any files uploaded, as uploaded.
- We recognize that Dropbox may choose to violate its trust guarantees, for instance in any or all of the following ways:
    - Dropbox may alter, by either choice or accident, files stored on its servers, or cause them to be altered.
    - Dropbox may delete, by either choice or accident, files stored on its servers, or cause them to be deleted.
    - Dropbox may disclose, by either choice or accident, the existence, filenames, and contents of files stored on its servers.
    - Dropbox may serve, by either choice or accident, incorrect data for a file.

Security evaluation (theoretical)

We believe that the core system design of FogVault v0.08.0 achieves the following security properties:

- In a situation where the official Dropbox client is not installed on the machine of any FogVault user with access to a file F, it is difficult for Dropbox (or any other observer without the user private key corresponding to some key table entry of F) to determine the contents of F.
    - F is only ever stored on Dropbox in encrypted form.
    - Decrypting F requires knowledge of the file encryption key (Kf) used to decrypt F. This is difficult to retrieve.
- It is difficult, in the same scenario, for Dropbox to induce an undetected file modification.
    - F is encrypted using libsodium's AEAD (authenticated encryption with additional data) primitive, which detects unauthorized changes in the encrypted data, block authenticators, and metadata. Attempting to modify the file, substitute file data, or modify the file metadata, will most likely cause AEAD validation to fail, causing the file to be rejected.
    - Attempting to inject a 'martian file', i.e. a file generated by Dropbox itself, could be detected and rejected by the client under certain assumptions. (This is not currently done.)
- It is difficult for Dropbox to induce an undetected file deletion, and thereby destroy a user's files.
- It is difficult for a network attacker to do anything useful.

The security guarantees can be summarized as such:

- Entities, other than users granted access, cannot read a file's contents.
- Data modification or corruption induced by entities other than authorized users cannot cause an authorized user's plaintext copy of a file to be modified or deleted.

Known bugs and unfinished tasks

There are some ambitions we had for the FogVault project that we were unfortunately unable to realise. We discuss these below.

As this project is ultimately only a proof of concept in its current form, there exist several known bugs in the implementation that could not be addressed in time for this milestone. If the prototype submitted for grading runs in a single session and is not restarted, and the remote Dropbox folder starts empty, we believe that the prototype operates in a satisfactory way. Beyond these testing conditions, the prototype breaks in some unfortunate ways; among other symptoms, file and folder duplication and/or relocation may occur. With future work we hope to clean up the implementation and eventually release a proper alpha build.

Sharing of files between users in a reasonable, user-friendly way was a desired goal for the FogVault project. Unfortunately, the limitations of the Dropbox API in particular made realisation of this goal nearly impossible in any respectable manner. The underlying cryptographic framework supports multi-user access perfectly well; files can be made decryptable by multiple users, and the cryptographic code supports (in principle) adding and removing of user access. Additionally, the necessary logic that allowing multiple users introduces, while tangled with edge cases, would ultimately not be too difficult. However, there is one API that we would need from Dropbox's end that is not provided in the current Dropbox core API, namely the ability to designate certain files and folders as 'shared' with other users; without this, the user experience flow for file sharing becomes too prohibitively cumbersome to be useful. Additionally, there are issues with establishing trust relationships between people and keys that remain inadequately addressed by the FogVault system design; in particular, it remains impossible for a user A to trust the provenance of an incoming file from a user B about whom A has no prior knowledge.

We were hoping to ship a more user-friendly UI than the one delivered - in particular, a more user-friendly first user experience - and further development would certainly need to produce one before a viable product could be released to the world. For various reasons, we were limited to producing the demonstration UI shown, which provides direct access to the relevant FogVault features but is not by any means tuned for usability.