Dropbox Confidentiality Client

Michael Muggler (mxm12131), Nathan Ritchey (nar140730) and Stephen Balhoff (sjb091020)

# Introduction

The Dropbox Confidentiality Client (DCC) is a Python command line utility that provides end-to-end encryption and integrity verification when hosting sensitive files on Dropbox. This software provides a single user with confidentiality and integrity assurance from adversaries present outside the control of the user. By default Dropbox provides users with these assurances. User data is uploaded to Dropbox using SSL, encrypted using 256-bit keys and authenticated[[1]](#footnote-1). [1]

For an average user Dropbox is secure. Dropbox routinely tests its applications and infrastructure for security issues. [2] The motivation of this project is not to provide missing security rather it is to enhance security from the user perspective. Dropbox encrypts data with 256-bit keys created and stored by Dropbox. Dropbox considers all data uploaded on behalf of a user in a single security class with no finer security controls. Dropbox is a company with employees and legal obligations. Leaving them vulnerable to government agencies and either malicious or unwitting employees.

The attack model assumes that malicious parties either at or working with Dropbox (such as governments) can decrypt and modify user data. This is because the 256-bit encryption key is created and stored by Dropbox. In order to prevent this we create two random 256-bit keys and use AES in CBC mode. These keys are encrypted and stored on Dropbox using a user provided password that is expanded into a key (to decrypt the random keys) using PBFDK2. With this program Dropbox will not be able to modify or decrypt files or have access to the metadata associated with the user’s file system.

This paper will discuss three major steps of implementation. First we discuss developing a user-friendly method of creating high entropy password. Second we discuss a method to derive, secure and store high integrity keys. Third we discuss how we represent the user’s file system and the mechanisms for uploading and downloading files from Dropbox. This paper then provides a detailed explanation of how the software works.

# Major Steps of the Implementation

## User-friendly Method for High Entropy Passwords

In many applications, including ours, the password interface is the first and only authentication method for a user to gain access to a secure system. Consequently many application designers enforce stringent requirements for passwords in an effort to make it difficult to guess by an unauthorized party. However this only serves to make it harder for a user to recall. Many times it forces the user to circumnavigate the mechanism by storing the password in an insecure physical location.

An important design principle for security mechanisms is to ensure the psychological acceptability of that mechanism. But to balance this with the strength requirements for the application is tricky. In our application we assume the adversary has the encrypted data and can perform an offline brute force attack on the password. Taken from data science the strength of passwords can be determined by information entropy.

Typically the higher the entropy the more unpredictable a password is, more resistant to brute-force attacks and harder to remember. In a special report by NIST[[2]](#footnote-2) a method for choosing a high entropy password was developed. The requirements are: a minimum of eight characters from a 94-character set, one upper case and lower case letter, one number and one special character. This string should not be a recognizable permutation of personally identifiable information[[3]](#footnote-3).

Later research with actual user data showed that actual human-selected passwords and despite meeting the NIST requirement had both poor entropy and memorability by the user. Entropy is formally defined as the probability of being guessed on the first attempt over some time period. We can represent entropy in terms of bits by the formula. Where N is the number of symbols permitted in the password and L is the desired length.

The NIST method produces 30-bits of entropy. However later research into the NIST requirement demonstrates that this scheme does not provide a reasonable security in any human-oriented system. RFC 4086, “Randomness Requirements for Security,” states that at least 128 bits of entropy is required for cryptographic keys used for a long period of time. Thus because we do not meet entropy requirements and cannot achieve such entropy with a human oriented system we cannot use the NIST method. Meeting such high entropy means passwords would need to be excessively long and not memorable.

To meet the memorability and low length requirements for passwords we employ the Diceware method in which the user will enter a random phrase. This phrase is easier to remember than a strict set of rules producing a jumbles string. Further human brains respond much better to pneumonic than strings of letters, numbers and symbols. In Diceware we have a list of 7,776 phrases. Each word is represented by a five digit number. We use a PRNG to “roll a dice” five times to select a word. We repeat this process 8 times. Each word selected produces 12.9 bits of entropy, giving a total entropy of 103.4 bits. While not the 128 bits expected it is sufficient according to Reinhold which recommends six words (or 77 bits).

## Method to Derive, Secure and Store High Integrity Keys

For our application we use two randomly generated keys. The encryption and signing key are 256-bit keys. We use Python’s standard random implementation which is cryptographically secure. This implementation uses a verity of sources to ensure randomness including the state of the network stack. These keys are used to encrypt and sign all files the user uploads to Dropbox through our client. To ensure added randomness we seed Pythons random with the hash of random documents found on the users drive. We store these keys in a password file which will be uploaded to Dropbox.

This password file contains both keys. We prompt the user for the master password which contains eight random words. The first four words and last four words are passed to a PBKDF2 function and a random salt. This function produces a 32 byte (or 256 bit) key. We encrypt the randomly generated encryption and signing key with the respective 256 bit keys derived from the master password. This output along with the two random salts of 16 bytes make the key matter stored in the password file.

We use the signing key to create a signed hash using SHA256 of the key matter which is verified prior to decrypting the key matter. This signing process ensures the data has not been maliciously modified to encrypt new data into Dropbox. The password file is uploaded to Dropbox so the user can download and decrypt the data from any machine connected to Dropbox with only the need for the master password.

## Virtual File System and File Management

At all times it is assumed that Dropbox is a hostile place to store data. Two files are required to facilitate the communication between the user’s local machine and Dropbox. The first file contains the user’s Dropbox API Token and container name. The purpose of the container is to allow the user to store multiple sets of files with different master passwords. Including data sets they may not own but ensure no conflicts occur.

The API token and name of the container is stored in a hidden file in the directory in which the user wants mirrored on Dropbox. This is encrypted using the master password all together. The key is derived from the master password using PBKDF2. The reason for encryption is the API token is just as critical as the user’s password. While the token only permits reading and writing to a specific directory on the Dropbox it should be protected. The container name is a Base-64 encoded hash of some textual data and a five-digit random number. The random number decreases the probability of collisions for container names.

The user’s file system is represented in a Virtual File System file (or “VFS”). This file is encrypted and decrypted each time the user performs an operation using the client. The VFS file represents the file system as a recursive dictionary. This file is encrypted and integrity protected. When a file is uploaded to Dropbox it is encrypted and integrity protected. Encryption must occur first, because we want integrity verification to occur first. The file loses its metadata instead being assigned a UUID. The file’s respective metadata and location on the users file system is stored in the VFS file. The encrypted file is uploaded in the containers directory on Dropbox with a UUID.

# Difficulties Faced During Implementation & Future Work

[TODO]

# Division of Labor

[TODO]

# Usage of the Software with Screenshots

[TODO]

1. Dropbox supports multifactor authentication. Dropbox implements a time-based one-time password algorithm (TOTP) as defined in RFC6238. Dropbox supports the mobile applications Authy and Google Authenticator. [1] [↑](#footnote-ref-1)
2. NIST Special Publication 800-63 [↑](#footnote-ref-2)
3. Personally Identifiable Information (PII) includes your name, date of birth, mailing address, know aliases, usernames, phone numbers or any string that can reasonably identify you in a pool of other persons. [↑](#footnote-ref-3)