

Cryptbox – Encrypted File System

Nolan Eastin, Louis Sobel, Stephanie Wang

December 13, 2013

Problem

This project was inspired over the current debate over the issue of trusting third parties with private information. For example, storing private files on Dropbox. Users have to trust Dropbox to have strong enough security that no malicious person will be able to access and read your files, and that Dropbox themselves won't be malicious and read your files and use your private information. Our solution, Cryptbox, is an encrypted file system inside Dropbox that uses passwords and provides an HTML wrapper so that the files can be accessed from the web app or locally on any computer given that the user has the correct key files or that the user knows a file password.

Encrypted File System Structure

We used fusepy to create a file system in user space. This file system is mounted on `/path_to_mount_point/` and will contain an `__enc__` subdirectory in which all of our encrypted files will be kept.

As mentioned, we root our file system with an `__enc__` prefix. The the files stored on disk under `__enc__` will be encrypted automatically by Cryptbox. We provide Dropbox with a symlink to `__enc__`, which means all they can access is the encrypted files. That means that although Dropbox has access to the files on disk, they are encrypted and they can't decrypt the information. This `__enc__` root is transparent to the user and users merely have to go to `~/Dropbox/cryptbox/foo` and Cryptbox will decrypt the files for the user.



Figure 1: Representation of sym link on Dropbox to our filesystem.

Figure 1 shows how the filesystem is represented on disk. If a user tried to read `/__enc__/cryptbox/foo`, they would see encrypted text. However, if they try to access it on `~/Dropbox/cryptbox/foo`, Cryptbox will handle the decryption and return the file to the user.

Data Structures and Methods

We use two primary data structures to handle the encrypted file system, these represent how Cryptbox manages encrypted files and system calls to the files under our system.

DecryptedFileManager

Its primary function is to manage access to the cryptbox files. Important functionality is open and close, which take care of encrypting and decrypting the files for the user.

open()

Opens the encrypted file. It then decrypts the file (given that you have the right credentials), and writes to a temporary file, say tmp-decrypted. Cryptbox will have a handle to that tmp-decrypted file and will release that when user calls close() on the encrypted file.

close()

On close, Cryptbox encrypts the tmp-decrypted file and writes the ciphertext to the encrypted file.

OpenDecryptedFile

This represents a handle to an open decrypted file.

decrypt()

given the correct password, it will decrypt the ciphertext and return plaintext.

encrypt()

encrypts a file using AES.

File System Encryption

Files that live under the Cryptbox system will be encrypted when placed there or when created there. Every file that Cryptbox handles will first be decrypted, Cryptbox will then create a temporary file which will hold the plaintext and be given to the user. The user can make changes to that temporary file. Whenever FUSE calls write() on the encrypted file, we encrypt the plaintext in the temporary file and write it to the encrypted file. Specifically, the user when flush() or close() are called on this tmp-decrypted file, FUSE will encrypt the file and write to the real file.

In the case that a file is created, only encrypt will be called to ensure that the encrypted file exists.

We have some very basic synchronization support which relies on using a global lock whenever certain operations are called, this deals with multiple users having a handle to the file.

Encryption

We use a combination of RSA and AES encryption. Each file will be encrypted with a password. That password can be a default one or can be specified (generally this is useful if you want to share the file, you pick a different password and somehow share this password). We don't currently support infrastructure to distribute the password safely. Our design relies on PGP to actually distribute each user's public keys.

Cryptbox has two main files that specify how file encryption works. We have a key file, stored

somewhere in the user's filesystem, that has the public and private RSA keys for a user. Additionally, we have a config file that specifies a default password and a password for specific files. It would be wise to store the key file and the config file outside of Dropbox to ensure that they don't have access to your keys.

```
{
  default_password: password
  files: {
    path: {
      password: {
        hash(key1) : enc_key1_password, hash(key2): enc_key2_password
        public_keys: [ key1, key2,...]
      }
    }
  }
}
```

Figure 2: Example Config file for Cryptbox

Figure 2 represents each user's config file. The config is stored in a JSON format. Given a file, a user merely needs to decrypt the password (which is encrypted using RSA) with their own private key and then use the password to decrypt the file (which is encrypted using AES).

We use key lengths of 256 bits. AES encryption is used to actually encrypt and decrypt each file. Given a password, we have a function to generate our AES encryption key. We use MD5 to generate the 256 bit key and then use that to encrypt the file.

HTML wrapper

We created an HTML wrapper for each file. Each file will have 3 key components:

1. Ciphertext
2. MIME Type
3. A password json object.

The json object is similar to the one in the config file. It specifies a list of encrypted passwords and the public keys used to encrypt the password. If the file is viewed on the web, it will present a form the user, which should prompt for the file password. If the user knows the password for this file, then he can enter this password, which will then be used to decrypt the file and reveal the plaintext to the user.

Future Work

One of the future goals for the project would be to provide some stronger security guarantees. Some ways in which this could be achieved is by replacing the md5 function we use to generate our AES keys to something stronger. Additionally, we currently have the MIME type exposed on every file. We could easily have included it as part of the encryption process.

Something that is lacking is that we assume that Dropbox will not modify the files in such a way that they can steal out passwords, and we also don't sign files to guarantee identity.

We currently lack directory support, which is one of the bigger issues that need to be addressed if we are going to support Dropbox sharing.

Conclusion

Cryptbox allows users to keep their files encrypted within Dropbox. These files can be accessed easily if one has access to the public/private keys or knows the password for a file. This helps usability in that users can access their files on the web with a password and access the files locally without changing the user experience (since Cryptbox is transparent to the user if the config file is set up correctly).