Design of an encrypted Dropbox

by Shashank and Rohit Gupta
IIT Kanpur

September 01, 2019

1 Introduction

In this document we present a design architecture of a cryptographically authenticated and secure dropbox-like file storage system.

1.1 Goals Achieved

1. Confidentiality
2. Integrity

2 Part 1:A simple but secure client

Following methods are implemented:

2.1 InitUser(username string, password string)

Here, we will set user datastructure where we will put username,password,private key and information about shared and owned files. Username will be implemented using **HMAC** algorithm and its key will be password of user, and password will be stored using Argon2 and its salt will be username as the server will never know real username and password so we can use them as key.We Generate AES key for encryption and rsa public key is stored in keystore for Digital Signature.
This data will be stored using **AES** symmetric key with IV as password and rest can be used as AES key.Username is passed through a hash function then **uuid** is created which can be used as a path.

2.2 GetUser(username string, password string)

GetUser will also use the **User** datastructure and another datastructure called **Maccheck** in order to authenticate the user. **GetUser** will try to establish a path using username and if it does not find a path then it will throw an error but in case it finds it then it will decrypt the file using the same password as key and IV(**first block of hashed password**) to decrypt it. After verifying the username and password the user can access the data.

2.3 StoreFile(filename string, data []byte)

Here we need to store the file metadata so we will create a datastructure called **Filemetadata** which will be used to store information like **UUID**,**encryption key**,**MAC key**, and .Firstly it will initialize **Filemetadata** with uuid and file encryption key

using AES,Verifying Key using HMAC and owner name .After verifying the owner of the file, this data will be stored in the owned section of the user datastructure. File will be stored into blocks and each block will be encrypted using AES key and for entropy iv of first block will be random bytes and after that iv of next blocks will be last bytes of encrypted previous block(size of iv will be userlib.blocksize).iv will be stored in front of each data block. Each block will be stored with its MAC for integrity on a path provided by inode
Inode Generate path by generating random bytes every 32 blocks for indirect pointer and generating random bytes for each direct pointer also.iv for future block will be stored in file metadata
Inode file strucure is used to store the file on the server.For that a datastructure called **Inode** will be created which will store information like **Size**(which will be stored in terms of number of blocks), **fileID** and a **double pointer** structure. 32 **double indirect** pointers and 32 **single indirect** pointers is used.Therefore

$$No.of blocks = 32 * 32 = 1024 blocks$$

2.4 AppendFile(filename string, data [])byte

For performing this action you should be an owner or a contributor. And if you are one of the previous two then you already have the file metadata which is the user datastructure from which you can get the path to the file and encryption key.So we will first call **Inode** structure and from there we will get number of blocks already present then from the next block we will follow the same scheme for each block as described in store file.

2.5 LoadFile(filename, blockoffset integer)

similar to the **AppendFile(...)** method to load a file just get the file metadata from its user datastructure and if not present then user is not owner or contributor.Inode strucure will be called after this from where we will get the path to block that is needed, From the path we will first check for integrity then we will decrypt the block using **AES** key and iv present in front of block and send to user.

3 Part 2: Sharing and Revocation

3.1 Sharing

For sharing we have **owned** and **share** section in user data structure(these are maps of the file shared with or owned by user). So we can store metadata of files owned by user in owned and files that are shared with user in share.**Sharing data** will be stored in a seperate datastructure. When owner wants to share the file with user it sends files metadata to the the receiver and the receiver will receive the file and store it with a name which is suitable to him(File path is present in uuid(File Metadata)).So file names could be different for sharer and in this way he could do anything with the file except revokation.Data to be sent is encrypted using public key of receiver.

3.2 Revokation

For the **revokation** of the file we need to check for the owner of the file. If the user is the owner of the file he will try to make a copy of the original file by calling the

Loadfile function on loop. And when he gets the whole file he will remove all the metadata of the original file and also modify the **owned** and **shared** section of the **User** datastructure.After this he will call the **Storefile** function and store the file with a same name but with new UUID and new keys. So other users will get an illusion that they have been revoked by the owner of the file.