CS3031 Assignment 2

"Securing The Cloud"

David Scolard
Student ID : 16322277

Introduction

The Task:

To develop an application capable of encrypting files and uploading them to the cloud, as well as downloading them and subsequently decrypting them. This application must only be accessible to individuals who are a part of my *"Secure Cloud Storage Group"*.

Google Drive API:

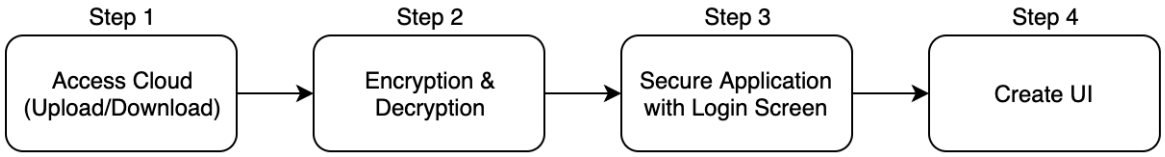
I previously have had little to no experience working with cloud applications or cryptography. I began searching for ways to do this, first thing to cross my radar was the Google Drive API. There are many tutorials online showing how to upload, download, create folders etc. The Google Drive API seemed extremely intuitive and easy to implement, I decided this was the cloud service I was going to use.

PyCrypto:

I decided to complete this project in python. This allowed me to use the Python Cryptography Toolkit (PyCrypto). PyCrypto is a collection of both secure hash functions(such as SHA256), and various encryption algorithms(AES,DES,RSA, etc.). This crypto library would be perfect for the task at hand.

Planned Layout

There are many facets to this project, careful planning was necessary to ensure it all came to getter as one functioning application. I broke it down into the following steps:



I tackled each of these steps individually in four files:

- gDrive.py
- encryption.py
- admin.py
- menus.py

Implementation

Google Drive Access:

The first step was to get a simple program working which was able to upload and download files from the cloud. In order to do this I first had to create my own google developer account. This would let me create the credentials necessary for accessing the Google Drive API. Once I had my key to give access to the API I stored it in a file called client_secret.json so it could be accessed. I set the scope to ['https://www.googleapis.com/auth/drive'] which gives full access to the drive for uploading, downloading etc. The google developers website provides an abundance of sample code for getting started which helped a lot.

My auth.py file deals with the credentials side of things, an instance of the auth class is created whenever I run gDrive.py giving me full access to my drive. The gDrive.py file is where I've written the functions responsible for uploading, downloading and browsing files in my drive. These functions were mostly taken again from the google developers website: https://developers.google.com/drive/api/v3/manage-uploads

My application currently works within the test_files directory for uploading files. Downloaded files are placed directly into my projects downloads folder. Once I had this part of the project working I now had to figure out how to encrypt/decrypt my files before uploading and downloading them respectively.

Encryption:

As I said at the beginning I decided to use the PyCrypto library. I could then encrypt my files with AES. The AES function expects your key to of length 16, 32 or 64 bytes long, for AES-128, AES-196 and AES-256 respectively. The longer the key here the stronger the encryption. PyCrypto also allows you to pass in an IV(Initialisation vector) into the AES. I've learnt this a vital part of block encryption algorithms that work in chain modes like CBC, if we encrypt the same plaintext with two different IV's then the resulting ciphertext sequences will look completely unrelated. We generate a random IV for each encryption. Once we have the file encrypted we write it back into the test_files directory from where it is uploaded to the drive. Once uploading is complete the encrypted file in the directory is deleted as there is no use for it there.
Decryption is fairly straight forward with the key. The encrypted file has already been downloaded into the downloads folder when the decrypt function is called. The decrypt function takes in the filename and the key. Retrieves the file and decrypts it using AES again. The decrypted file is written back into downloads and the encrypted file is deleted.

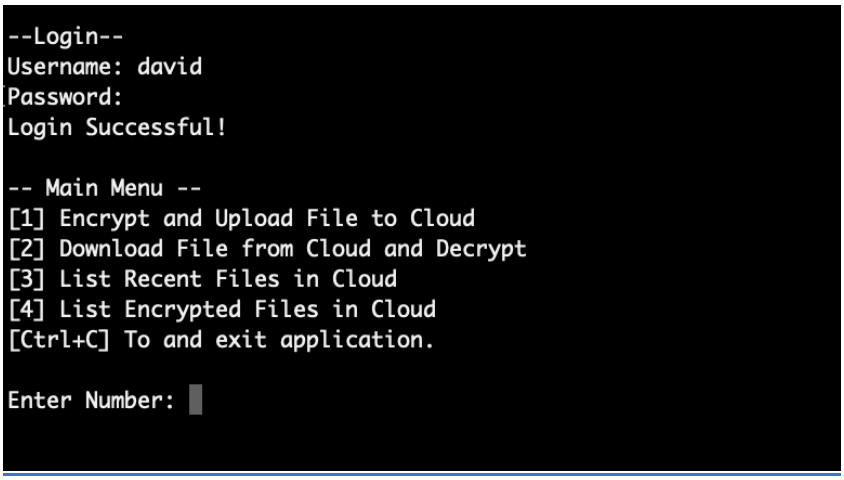
Secure Application:
Now my application had access to the cloud and was able to carry out the necessary encryption, the next step was to create a secure group. The obvious way to do this in my eyes was a login screen from which users could gain entry to the application. I also wanted to be able to dynamically add and remove users from the cloud storage group. The admin.py file takes care of creating a new user, deleting a user and logging users in. When a user is created they simple need to enter their desired username and password which can be used henceforth to access the app. To delete a user you just need the name of the user to delete. Obviously I didn't want everyone to have the ability to add and delete any users they want so I created an admin account from the start, this admin account has its own menu which allows it control over users when it logs in. Usernames and passwords are stored in a pickle file for added security, when a login is attempted the program simply checks if the username and password match in the pickle file.

Interface:
My interface is nothing fancy, simply using the terminal window with plain text and prompting the user for inputs such as file name. When a user logs in they are brought to the main menu where they can choose between:

1. Encrypting a file and uploading it to the cloud
2. Decrypting a file and downloading it
3. List recent files in cloud storage
4. List Encrypted Files in cloud
5. Quit program

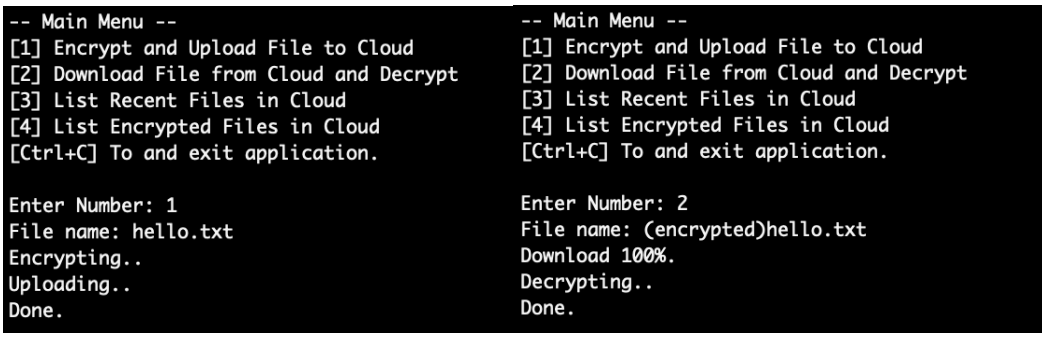
To access one of these the user simple needs to enter the number corresponding to the action. If they choose to upload or download a file they will be prompted to enter the name of the file and the program will take it from there. The user might not know the name of a file in the cloud so I have the list files function which list recent files, and also a list encrypted files function which will list all encrypted files in the cloud. This list recent files function will prompt the user to enter the number of files they want to list. If they enter 10 for example the program will then list the 10 most recently used files in the cloud.

Login and Menu:

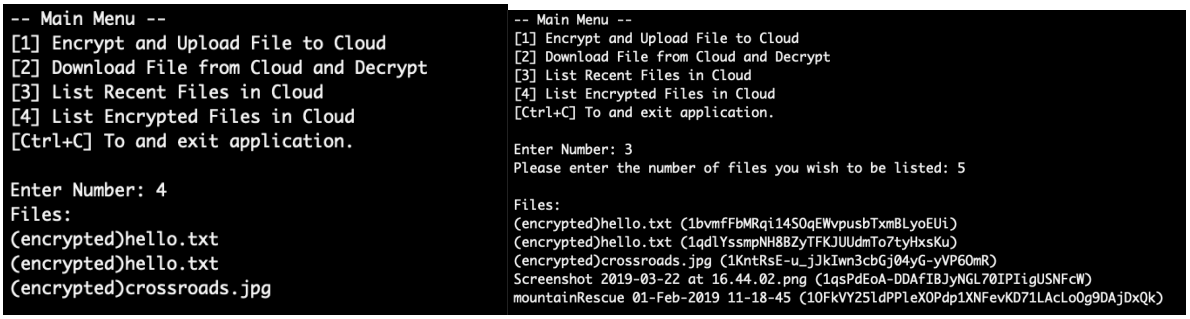


**Note: Prompt the user for a password without echoing. This is done using the getPass() function.*

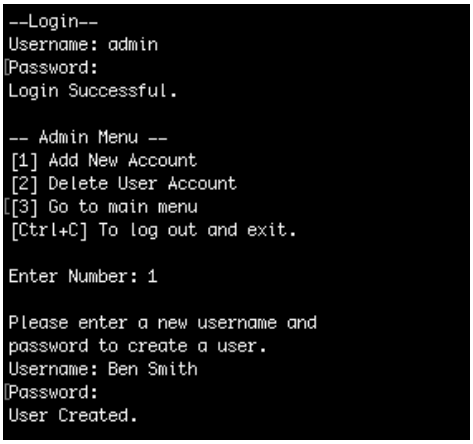
Download/Upload:



List Encrypted Files/Recent Files:



Admin Menu and New User:



User Delete: