

CS3031 Project 2

Securing the Cloud

Author: Vahe Sasunts

Student ID: 16316570.

Task

The aim of this project is to develop a secure cloud storage application for Dropbox, Box, Google Drive, Office365 etc. For example, your application will secure all files that are uploaded to the cloud, such that only people that are part of your “Secure Cloud Storage Group” will be able to decrypt your uploaded files. To all other users the files will be encrypted. You are required to design and implement a suitable key management system for your application that will allow you to share files securely, and add or remove users from your group. You are free to implement your application for a desktop or mobile platform and make use of any open source cryptographic libraries.



Table of Contents

CS3031 Project 2	0
Securing the Cloud	0
Task.....	0
Introduction	1
My interpretation of the task.....	1
Research	1
Project Plan	2
Approach	2
Solution	3
Main.py.....	3
Main().....	3
Users.py	3
Save_list(obj, name).....	3
load_list(name)	3
newUser().....	3
Login()	3
deleteUser()	4
GoogleDriveAPI.py.....	4
MenuTypes.py	4
StandardMenu().....	4
Encryption.py.....	5
KeyGen().....	5
KeyRead()	5
Encrypt(filename, Key).....	5
Decrypt(filename, key).....	5
Auth.py	5
Strengths	6
Weaknesses	6
Testing.....	6
Debugging.....	6
Results	6
Things I would change	6
Conclusion.....	7

What I learned	7
What I achieved	7
Appendix	8
main.py	8
users.py	8
googleDriveAPI.py	10
menuTypes.py	11
encryption.py	13
auth.py	14
References	16

Introduction

My interpretation of the task

From reading the brief it was clear that I was given the freedom to use whatever technologies I wanted, I could either create a desktop or mobile program and can use any library or API I wish.

The main goal of the project is to create an application which connects to an online cloud storage platform and can upload files while encrypting them and as you download the file it should decrypt the encrypted file.

The program should be able to handle multiple users to login and download these secured files and also the key should be shared among them

Research

I decided to use python to create the program and therefore based most of my research on python libraries and API's.

Initially I had to decide which cloud storage platform I was going to use. There was little research needed here as I am an avid user of cloud storage platforms and have defaulted to using Google Drive. I then had to look into the API's that Google provides for drive. Luckily, they have an extensive QuickStart Guideⁱ which provided me with how to authenticate the user connecting to the cloud service. The guide did not end there as there was a guide for uploading filesⁱⁱ, downloading filesⁱⁱⁱ, and searching for files^{iv}. There were more guides however these were enough for me to create the required program.

I then researched cryptography for python and found a popular library to do just that. The library I'm using for the cryptography part of the assignment is named cryptography and it has great documentation^v. The encryption I will use is called Fernet^{vi} it specifically uses AES in CBC mode with 128-bit key for encryption and HMAC using SHA256 for authentication. It initialises vectors using the computers salt^{vii}.

Next, I checked common ways to store credentials such as username and password and found that pickle^{viii} is a tool which allows you to store python objects and re-load these objects back into the program after exiting. This meant that I could use any type of username and password system. One common solution is to use a key-value pair (dictionary or hash map) for this.

Project Plan

Approach

After doing the last assignment (project 1 – web proxy server), I realised that I shouldn't write all of my code in one main file. This created a lot of confusing and made it difficult to debug. Any changes made would affect the whole program and this was disastrous in my case as I had to make many changes on the day the assignment was to be demoed.

For this reason, I decided to keep the code separate and import relevant classes as needed for each class. The classes I made are:

- main.py
- users.py
- googleDriveAPI.py
- menuTypes.py
- encryption.py
- auth.py

Main.py will be the main line of the program. This is what will run to create the front end for the user. It is small and compact and invokes the user class to allow for login as well as the menuType class to generate the menu.

Users.py as the name suggests is where user names and passwords are created, deleted and stored.

GoogleDriveAPI.py is the interface between my program and Google drive and will allow me to carry out the required functions to download and upload files.

MenuTypes.py will be the class which creates the menus for a privileged (admin) user and standard user. It invokes most classes to carry out required functions.

Encryption.py again as the name suggests is the class that carries out the encryption and decryption functions as well as generating keys.

Auth.py is used to authenticate the Google drive user when calling the APIs.

Solution

Note all of my code is available in the appendix.

Main.py

Main()

When the program starts the very first thing it does is look to see if a key exists, if no key exists then one is generated automatically. Next it checks to see if there are any previous users registered in the program. If there are no users, then it prompts the user to create an admin account which will be given privileged rights to create new accounts or generate new keys.

Otherwise it would present the user to login. If you log in as admin then you will have a menu specific to that admin user with added functionality. Otherwise you get presented a standard menu. This is running in a loop to allow users to log in and out as they wish.

Users.py

Save_list(obj, name)

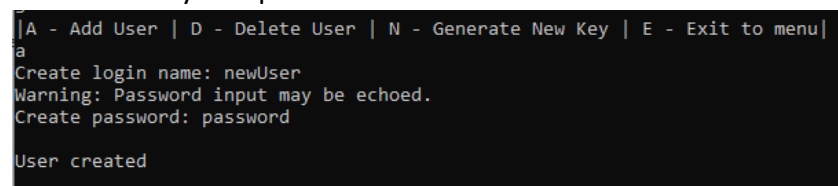
This function is used to pickle (save) the list of users that are part of the program. I use a dictionary to store the user name and password as a key value pair. It is called every time a new user is created.

load_list(name)

This function is used to load the object where the usernames and passwords are store. It is called when the user class is invoked to ensure the usernames are up to date.

newUser()

This function is used to create a new user. It is simple as it asks the user to enter a user name, it warns the user if this user name exists and then asks for a password and stores it in our dictionary and pickles it.



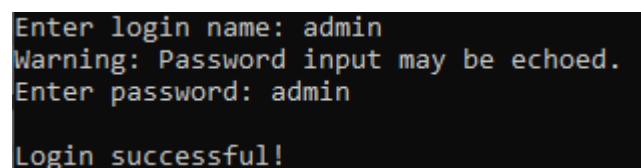
```
| A - Add User | D - Delete User | N - Generate New Key | E - Exit to menu |
a
Create login name: newUser
Warning: Password input may be echoed.
Create password: password

User created
```

Figure 1: User Creation

Login()

This function is called in the main line, it provides the user with a login screen and checks if the user logging in is the admin, a standard user or if they fail to login in 3 attempts it will terminate the session.



```
Enter login name: admin
Warning: Password input may be echoed.
Enter password: admin

Login successful!
```

Figure 2: Login Screen *Warning generated as password is echoed in my IDE

deleteUser()

This is called to delete a user by the admin. It prevents from the admin account from being deleted and also if the user does not exist it will notify the user. Once a certain user is deleted the username dictionary is once again pickled and stored safely.

```
|A - Add User | D - Delete User | N - Generate New Key | E - Exit to menu|
d
Enter username you wish to delete: newUser
User deleted.
|A - Add User | D - Delete User | N - Generate New Key | E - Exit to menu|
```

Figure 3: User deletion

[GoogleDriveAPI.py](#)

Here there are all relevant functions required to carry out uploading downloading and file searching. The functions include:

- uploadFile(fileName)
- downloadFile(file_id, fileName)
- searchFile(query)
- fileID(query)

[MenuTypes.py](#)

PrivilegeMenu()

This generates a menu for the admin which has more features such as adding and deleting users and also generating a new key. This is where the majority of actions are performed such as uploading a file, it will encrypt then upload and when downloading from the drive it will download and decrypt the file.

```
|U - Upload file | D - Download file | S - User settings | L - Log out | E - Exit|
```

Figure 4: Privilege menu

StandardMenu()

This menu is nearly identical to the above menu however the settings option is removed to prevent the user from performing privileged actions.

Encryption.py

KeyGen()

KeyGen as the name suggests is used to generate a new key and store it on the computer.

KeyRead()

This is used to read the key from the computer, if no key exists it will automatically generate a new one by calling keyGen().

Encrypt(filename, Key)

This is used to encrypt the file, it is using Fernet to perform the required encryption.

Decrypt(filename, key)

This is used to decrypt the file that had been encrypted.

Auth.py

This is used to authenticate with Google Drive. This is taken from the Google Drive API QuickStart guide.

Strengths

The key strengths of my solution include:

- Multiple users can be created and deleted.
- Files are encrypted as they upload and decrypted after download.
- The UI is easy to navigate.
- Files cannot be decrypted unless you know the key.

Weaknesses

Weaknesses I found include:

- The user interface is built around a console terminal. It would be nice to build a GUI around the program.

Testing

Debugging

From the last project I decided to have a better environment for me to debug. This is why I used Visual Studio as my IDE for this project. I would have break points to easily see where my code would run into issues and when writing all the other functions such as encrypting, decrypting, upload file, download etc. I would have a test file where I would run the code on its own to ensure the code works and later try and integrate it to the overall solution. This helped me when testing each function as I did not have to navigate the menus etc.

Results

After debugging I had a finished product. It does exactly what the brief asks it to do. It encrypts and uploads files, download and decrypts them, it can handle multiple users and the key is stored safely on the computer.

Things I would change

As stated in the weaknesses, It would be nice to have a GUI to make it more intuitive for the user, however this was not specified in the brief. Alternatively, this could have been made as a chrome extension where you log in via the extension and as you upload a file it would encrypt it and then upload and again decrypt and download.

Conclusion

What I learned

From this project I learned how to use and interface with the Google Drive API. I also learned how to encrypt and decrypt files and manage a user database.

What I achieved

At the end of the project I managed to get a lot of the required tasks complete. My solution was able to create and delete users, encrypt files, decrypt files, upload them to my Google drive and also download them from my google drive.

Appendix

main.py

```
import googleDriveAPI
import users
import menuTypes
import encryption

def main():
    #Checking if key exists
    e = encryption
    key = e.keyRead()
    try:
        while True:
            u = users
            m = menuTypes
            #Checking if any users exists, if they don't then a new user will have to
            be created
            if(u.accounts == 0):
                print("\nYou must create an admin account. Please set username as
                'admin'\n")
                u.newUser()
            else:
                #Log in screen
                u.login()

                #Admin has their own menu and standard user has their own specific
                menu

                if u.privilege:
                    m.privilegeMenu()
                else:
                    m.standardMenu()

        except Exception:
            pass

if __name__ == "__main__":
    main()
```

users.py

```
import pickle
import sys
import getpass

#Function which saves the users dictionary object
def save_list(obj, name ):
    with open('Users/' + name + '.pkl', 'wb') as f:
        pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)

#Function which loads the user dictionary object, if none exists an empty dictionary
is returned
def load_list(name):
    try:
        with open('Users/' + name + '.pkl', 'rb') as f:
            return pickle.load(f)
    except FileNotFoundError:
        return {}
```

```

#users dictionary (Hash map)
users = load_list("list")
#Bool which indicates if admin is logged in or not
privilege = False

#Checks to see if dictionary is empty or not
if not users:
    accounts = 0
else:
    accounts = 1

#Function to create a new user
def newUser():
    while True:
        createLogin = input("Create login name: ")
        if createLogin in users:
            print("\nLogin name already exist! Try again\n")
        else:
            createPassword = getpass.getpass("Create password: ")
            users[createLogin] = createPassword
            #Takes user inputs stores in dictionary then saves it using the save_list
            save_list(users, "list")
            print("\nUser created\n")
            break

#Function which allows user to login
def login():
    global privilege
    count = 0
    while True:
        login = input("Enter login name: ")
        password = getpass.getpass("Enter password: ")

        #First case if the user is logging in as admin
        if login in users and users[login] == password and login == "admin":
            privilege = True
            print("\nLogin successful!\n")
            break

        #Next case is if standard user logs in
        elif login in users and users[login] == password:
            print("\nLogin successful!\n")
            break

        #Exceeding failed log in attempts
        elif count == 3:
            print("\nToo many wrong tries. Goodbye!")
            sys.exit()
        else:
            #Error logging in
            count = count+1
            print("\nUser doesn't exist or wrong password! Try again.\n")

#Function to delete a user
def deleteUser():
    userName = input("\nEnter username you wish to delete: ")
    #Cannot delete admin for obvious reasons
    if userName == "admin":
        print("Cannot delete the admin account.\n")
    elif userName in users:
        #Deleting user from dictionary

```

```

        del users[userName]
        save_list(users, "list")
        print("User deleted.\n")
    else:

        print("User name does not exist in network.\n")

googleDriveAPI.py
from __future__ import print_function
import httpplib2
import os

from apiclient import discovery
from oauth2client import client
from oauth2client import tools
from oauth2client.file import Storage

try:
    import argparse
    flags = argparse.ArgumentParser(parents=[tools.argparser]).parse_args()
except ImportError:
    flags = None
import io
import auth
from apiclient.http import MediaFileUpload, MediaIoBaseDownload

# If modifying these scopes, delete your previously saved credentials
# at ~/.credentials/drive-python-quickstart.json
SCOPES = 'https://www.googleapis.com/auth/drive'
CLIENT_SECRET_FILE = 'client_secret.json'
APPLICATION_NAME = 'Drive API Python Quickstart'

authIntsance = auth.auth(SCOPES,CLIENT_SECRET_FILE,APPLICATION_NAME)
credentials =authIntsance.get_credentials()
http = credentials.authorize(httpplib2.Http())
drive_service = discovery.build('drive', 'v3', http=http)

#Google drive API's taken from Google Drive API Quickstart guide modified slightly to
my spec

def uploadFile(fileName):
    try:
        file_metadata = {'name': fileName}
        filePath = "Files/"+fileName
        media = MediaFileUpload(filePath,
                                mimetype="text/plain")
        file = drive_service.files().create(body=file_metadata,
                                            media_body=media,
                                            fields='id').execute()

        fileID = file.get('id')
        print('File ID: ' + fileID )
    except FileNotFoundError:
        print("\nFile does not exists please double check\n")

def downloadFile(file_id, fileName):
    try:
        request = drive_service.files().get_media(fileId=file_id)
        fh = io.BytesIO()
        downloader = MediaIoBaseDownload(fh, request)
        done = False
        while done is False:

```

```

        status, done = downloader.next_chunk()
        print("Download %d%%." % int(status.progress() * 100))
    with io.open("Downloads/" + fileName, 'wb') as f:
        fh.seek(0)
        f.write(fh.read())
except FileNotFoundError:
    print("\nFile does not exists please double check\n")
except Exception:
    pass

def searchFile(query):
    results = drive_service.files().list(
        pageSize=100,fields="nextPageToken, files(id, name)", q="name contains '" + query
+ "'"").execute()
    items = results.get('files', [])
    if not items:
        print('No files found.')
    else:
        for item in items:
            print(item['name'])

def fileID(query):
    results = drive_service.files().list(
        pageSize=100,fields="nextPageToken, files(id, name)", q="name contains '" + query
+ "'"").execute()
    items = results.get('files', [])
    if not items:
        print('No files found.')
    else:
        for item in items:
            return item['id']

```

menuTypes.py

```

import users
import sys
import encryption
import googleDriveAPI

u= users
e= encryption
g= googleDriveAPI

#Function to generate menu for privileged (admin) user
def privilegeMenu():
    try:
        while True:
            #Menu system used to navigate the management console
            user_In = input("|U - Upload file | D - Download file | S - User settings
| L - Log out | E - Exit|\n").lower()
            if(user_In == "s"):
                while True:
                    #Cases for creating or deleting users and creating a new key
                    user_In = input("|A - Add User | D - Delete User | N - Generate
New Key | E - Exit to menu|\n").lower()
                    if (user_In == "a"):
                        u.newUser()

                    elif(user_In == "d"):
                        u.deleteUser()

                    elif(user_In == "N"):

```

```

        e.keyGen()
        break
    elif(user_In == "e"):
        break
    else:
        print("Wrong input try again\n")

elif(user_In == "u"):
    while True:
        #Upload file screen
        print("\nEnsure that files you wish to upload are in the 'Files'
folder.\nEnter m to return to main menu.\n")
        user_In = input("Enter file name: ")
        if(user_In == "m"):
            break
        else:
            e.encrypt(user_In, e.keyRead())
            g.uploadFile(user_In)

elif(user_In == "e"):
    print("Exiting.")
    sys.exit()

elif(user_In == "l"):
    #Logging out of admin account and setting admin bool to false
    u.privilege = False
    print("Logging out.")
    break

elif(user_In == "d"):
    #Download file screen
    while True:
        user_In = input("\n|S - Search for file | D - Download File | E -
Exit to menu|\n").lower()
        if(user_In == "s"):
            #Incase you forgot the file name you can double check here
            user_In = input("Enter file name: ")
            g.searchFile(user_In)
        elif(user_In == "d"):
            #download file via file name
            user_In = input("Enter file name: ")
            fileID= g.fileID(user_In)
            g.downloadFile(fileID, user_In)
            e.decrypt(user_In, e.keyRead())
        elif(user_In == "e"):
            break
        else:
            print("Wrong input try again\n")
    else:
        print("Wrong input try again\n")

except Exception:
    pass

#Function which generates menu for standard non-privileged users
def standardMenu():
    try:
        while True:
            #Same as above minus the settings option

```

```

        user_In = input("|U - Upload file | D - Download file | L - Log Out | E -
Exit|\n").lower()
        if(user_In == "u"):
            while True:
                print("\nEnsure that files you wish to upload are in the 'Files'
folder.\nEnter m to return to main menu.\n")
                user_In = input("Enter file name: ")
                if(user_In == "m"):
                    break
                else:
                    e.encrypt(user_In, e.keyRead())
                    g.uploadFile(user_In)

            elif(user_In == "e"):
                print("Exiting.")
                sys.exit()

            elif(user_In == "l"):
                print("Logging out.")
                break

            elif(user_In == "d"):
                while True:
                    user_In = input("\n|S - Search for file | D - Download File | E -
Exit to menu|\n").lower()
                    if(user_In == "s"):
                        user_In = input("Enter file name: ")
                        g.searchFile(user_In)
                    elif(user_In == "d"):
                        user_In = input("Enter file name: ")
                        fileID= g.fileID(user_In)
                        g.downloadFile(fileID, user_In)
                        e.decrypt(user_In, e.keyRead())
                    elif(user_In == "e"):
                        break
                    else:
                        print("Wrong input try again\n")
            else:
                print("Wrong input try again\n")

```

```

except Exception:

```

```

    pass

```

encryption.py

```

from cryptography.fernet import Fernet
import io

```

#Function which generates our key (AES)

```

def keyGen():
    key = Fernet.generate_key()
    file = open('Keys/key.key', 'wb')
    file.write(key)
    file.close()

```

#Function which reads the key

```

def keyRead():
    try:
        file = open('Keys/key.key', 'rb')
        key = file.read()

```



```

        file.close()
        return key
    except FileNotFoundError:
        #If key does not exists a new one is created.
        print("No Key exists, a new one has just been created.")
        keyGen()
        keyRead()

#Function to encrypt files using the key
def encrypt(fileName, Key):
    with open("Files/"+ fileName, "rb") as f:
        data = f.read()
    fernet = Fernet(Key)
    encrypted = fernet.encrypt(data)
    with open("Files/"+ fileName, "wb") as f:
        f.write(encrypted)

#Function to decrypt files using the key
def decrypt(fileName, Key):
    with open("Downloads/"+ fileName, "rb") as f:
        data = f.read()
    fernet = Fernet(Key)
    decrypted = fernet.decrypt(data)
    with open("Downloads/"+ fileName, "wb") as f:
        f.write(decrypted)

auth.py
from __future__ import print_function
import httpplib2
import os

from apiclient import discovery
from oauth2client import client
from oauth2client import tools
from oauth2client.file import Storage

try:
    import argparse
    flags = argparse.ArgumentParser(parents=[tools.argparser]).parse_args()
except ImportError:
    flags = None

#Google drive Authentication taken from Google Drive API Quickstart Guide

class auth:
    def __init__(self, SCOPES,CLIENT_SECRET_FILE,APPLICATION_NAME):
        self.SCOPEES = SCOPES
        self.CLIENT_SECRET_FILE = CLIENT_SECRET_FILE
        self.APPLICATION_NAME = APPLICATION_NAME

    def get_credentials(self):
        """Gets valid user credentials from storage.
        If nothing has been stored, or if the stored credentials are invalid,
        the OAuth2 flow is completed to obtain the new credentials.
        Returns:
            Credentials, the obtained credential.
        """
        cwd_dir = os.getcwd()
        credential_dir = os.path.join(cwd_dir, '.credentials')

```

```

if not os.path.exists(credential_dir):
    os.makedirs(credential_dir)
credential_path = os.path.join(credential_dir,
                                'google-drive-credentials.json')

store = Storage(credential_path)
credentials = store.get()
if not credentials or credentials.invalid:
    flow = client.flow_from_clientsecrets(self.CLIENT_SECRET_FILE,
self.SCOPES)
    flow.user_agent = self.APPLICATION_NAME
    if flags:
        credentials = tools.run_flow(flow, store, flags)
    else: # Needed only for compatibility with Python 2.6
        credentials = tools.run(flow, store)
    print('Storing credentials to ' + credential_path)
return credentials

```

References

- ⁱ <https://developers.google.com/drive/api/v3/quickstart/python>
- ⁱⁱ <https://developers.google.com/drive/api/v3/manage-uploads>
- ⁱⁱⁱ <https://developers.google.com/drive/api/v3/manage-downloads>
- ^{iv} <https://developers.google.com/drive/api/v3/search-files>
- ^v <https://cryptography.io/en/latest/>
- ^{vi} <https://cryptography.io/en/latest/fernet/>
- ^{vii} <https://github.com/fernet/spec/blob/master/Spec.md>
- ^{viii} <https://docs.python.org/3/library/pickle.html>