Course: CS3031 Advanced Telecommunications
Title: Assignment 2 - Securing the Cloud
Name: Leong Kai Ler
Student Number: 15334636
Date: April 8, 2019

Introduction:

Purpose:

The aim of this project is to develop a secure cloud storage application for Dropbox, Box, Google Drive, Office365 etc. For example, the application should be able to 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.

A suitable key management system will be designed and implemented for the application that will allow files to be shared securely, and users to be added and removed from “Secure Cloud Storage Group”. The application can be set up on a desktop or mobile platform and make use of any open source cryptographic libraries.

Design

The implementation of this assignment can be divided into mainly two components: User Interface and a Cloud Storage Group. These components make use of different modules that consists of helper functions to execute their tasks.

The highlight of the design revolves around how to transmit symmetric keys to valid users in the group securely. The symmetric keys used for encryption when uploading files and decryption when downloading. One critical assumption that is took on during the development of the application is such that the activities of the cloud storage group is always being peeked on and the contents in it is always exposed to outsiders.

As a result, the contents in the cloud storage has to be encrypted for data protection and can only be decrypted by users who have access to the symmetric keys. To achieve this, a symmetric key has to shared securely among the users. The initiative taken here is to have the users send their public keys to the cloud storage group, so that it can be used to encrypt the symmetric key during transmission.

User Interface

Implementation

Code:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Thu Apr  4 09:11:25 2019

@author: Adamlkl
"""
import os
import sys
import random
import Encryptor
import KeySaver
from multiprocessing.connection import Client
from multiprocessing.connection import Listener
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive

folder_Id = '17oua44SP5sR6E_g_h3a9Ua5qjHqAFvFy'

'''
    - try to establish connection with CloudGroup
    - send over public key for symmetrical key encryption
    - get encrypted symmetrical key from CloudGroup
    - if user is not in the userlist, encrypted symmetrical key won't be sent
'''
def retrieve_symmetrical_key(username, address, port, group_address,
   group_listener, user_key):
    ser_key = KeySaver.serialize_key(user_key)
    conn = Client(address, authkey=b'secret password')
    conn.send([username, port, ser_key])
    conn.close()

    group_connection = group_listener.accept()

    encryption_key = b"error"
    try:
        encryption_key = group_connection.recv()
    except:
        pass
    group_connection.close()
    return KeySaver.generate_symmetric_key(user_key, encryption_key)

# download file from drive folder with key passed and save it to downloads
def retrieve_file(symmetric_key, filename, drive):
    file_list = drive.ListFile({'q': "'17oua44SP5sR6E_g_h3a9Ua5qjHqAFvFy' in
        parents and trashed=false"}).GetList()
    for file1 in file_list:
        # find file to be downloaded
        if file1["title"] == filename:
            encrypted_text = file1.GetContentString()
            decrypted_text = Encryptor.decrypt(encrypted_text.encode(),
                symmetric_key,)

            # printing downloaded text to check results
            print(decrypted_text)

            # save files in downloads
            with open (os.path.join("downloads",filename),'wb') as d_file:
                d_file.write(decrypted_text)
                d_file.close()

# encrypt file with key passed and upload it to drive folder
def upload_file(symmetric_key, filename, drive):
    u_file = drive.CreateFile({"parents": [{"kind": "drive#fileLink", "id": "
        17oua44SP5sR6E_g_h3a9Ua5qjHqAFvFy"}],'title':filename})
    with open (os.path.join("testfiles",filename),'rb') as uploadfile:
        plain_text = uploadfile.read()
        encrypted_text = Encryptor.encrypt(plain_text, symmetric_key)
        u_file.SetContentString(encrypted_text.decode())
        uploadfile.close()
    u_file.Upload()

# print usage of the user page
def usage():
    print "Command List: \n1. upload <file> \n2. download <file> \n3. quit\n"

# print list of files in drive folder
def print_fileList(drive):
    # Auto-iterate through all files that matches this query
    file_list = drive.ListFile({'q': "'17oua44SP5sR6E_g_h3a9Ua5qjHqAFvFy' in
        parents and trashed=false"}).GetList()
    for file1 in file_list:
        print('title: %s, id: %s' % (file1['title'], file1['id']))

def main():
    # get username
    username = raw_input("Enter username?\n")
```

```
'''
    - try to load user's key using username
    - if found, load the key from file
    - otherwise, create a new one and save it in files
'''
try:
    user_key = KeySaver.load_key(username)
except:
    user_key = Encryptor.generate_private_key()
    KeySaver.save_key(username, user_key)

'''
    - sets up local Google webserver to automatically receive
      authentication code from user and authorizes by itself.
'''
gauth = GoogleAuth()
# Creates local webserver and auto handles authentication.
gauth.LocalWebserverAuth()
# Create GoogleDrive instance with authenticated GoogleAuth instance.
drive = GoogleDrive(gauth)

'''
    - attempting to establish connection with CloudGroup to get symmetric
      key for encryption of files
'''
address = ('localhost', 6000)
port = random.randint(6001,7000)
group_address = ('localhost', port)      # family is deduced to be 'AF_INET
    '
group_listener = Listener(group_address, authkey=b'secret password')
sym_key = retrieve_symmetrical_key(username, address, port, group_address,
     group_listener, user_key)
running = True

while running:
    inputs = raw_input("How can I help you?\n")

    # requests for symmetrical key in case it is changed
    sym_key = retrieve_symmetrical_key(username, address, port,
       group_address, group_listener, user_key)

    # handles instructions from users
    argv = inputs.split(' ')
    if len(argv)>2:
        print "Usage: python ex.py "
        sys.exit(1)
    else:
        command = argv[0]

        if command == "upload":
            filename = argv[1]
            upload_file(sym_key, filename, drive)
        elif command == "download":
            filename = argv[1]
            retrieve_file(sym_key, filename, drive)
        elif command == "quit":
            running = False
            print "Goodbye"
```

```python
            else:
                usage()

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

Cloud Storage Group

Implementation

Code:

```python
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri Apr  5 13:19:19 2019

@author: Adamlkl
"""
import os
import pickle
import threading
import KeySaver
import DriveManager
import Encryptor
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from multiprocessing.connection import Client
from multiprocessing.connection import Listener


class CloudGroup(threading.Thread):

    def __init__(self, users, lock, listener, sym_key):
        super(CloudGroup,self).__init__()
        self.users  = users
        self.lock = lock
        self.listener = listener
        self.key = sym_key
        self.running = True

    # change symmetric key of the CloudGroup
    def change_key(self,new_key):
        self.key = new_key

    # shut down the CloudGroup
    def close(self):
        self.running = False

    '''
        - continously listens to users
        - get public key from valid users
        - encrypt symmetric key with corresponding public key
        - send encryted symmetric key back to the user
    '''
    def run(self):
        while True:
            connection = self.listener.accept()
            message = []
            try:
                message = connection.recv()
            except:
                pass
            # print(msg)
            if len(message)==3:
                self.lock.acquire()
                if message[0] in self.users:
                    client_address = ('localhost', message[1])
                    response = Client(client_address, authkey=b'secret
                        password')

                    public_key = KeySaver.load_public_key(message[2])
                    encrypted_key = Encryptor.encrypt_symmetric_key(public_key
                        , self.key)
                    response.send(encrypted_key)
                    response.close()
                else:
                    print('Unauthorised access by %s' % (message[0]))
                self.lock.release()

            connection.close()


# print commands in usage list
def usage():
    print "Command List: \n1. add <username> \n2. remove <username> \n4. list
        \n4. quit\n"

'''
    - decrypt all the files in drive folder using old symmetric key
    - create new symmteric key and save it
    - encrypt all the files in drive folder using new symmetric key
'''
def reset(key, drive, client_handler, sym_key_filename):
    folder = drive.ListFile({'q': "'17oua44SP5sR6E_g_h3a9Ua5qjHqAFvFy' in
        parents and trashed=false"}).GetList()
    DriveManager.decrypt_all_files(key, folder)
    new_key = Encryptor.generate_key()
    with open(os.path.join("SymmetricKey",sym_key_filename), 'wb') as key_file
        :
        key_file.write(new_key)
    client_handler.change_key(new_key)
    DriveManager.encrypt_all_files(new_key, folder)

def main():
    filename = 'UserList'
    # load users in CloudGroup
    try:
        infile = open(filename,'rb')
        user_list = pickle.load(infile)
        infile.close()
    except IOError:
        print "Could not read file:", filename
        user_list = set()

    # load Symmetric key
    sym_key_filename = 'Symmetric_Key.txt'
    try:
        with open(os.path.join("SymmetricKey",sym_key_filename), 'rb') as
            key_file:
            sym_key = key_file.read()
    except:
        print "Could find symmetric key file:", filename
        sym_key = Encryptor.generate_key()
        with open(os.path.join("SymmetricKey",sym_key_filename), 'wb') as
            key_file:
             key_file.write(sym_key)

    '''
        - sets up local Google webserver to automatically receive
          authentication code from user and authorizes by itself.
    '''
    gauth = GoogleAuth()
    # Creates local webserver and auto handles authentication.
    gauth.LocalWebserverAuth()
    # Create GoogleDrive instance with authenticated GoogleAuth instance.
    drive = GoogleDrive(gauth)

    '''
        - Create listener to receive requests from users
        - attempting to establish connection with users to send symmetric
          key for encryption of files
    '''
    address = ('localhost', 6000)     # family is deduced to be 'AF_INET'
    listener = Listener(address, authkey=b'secret password')
    lock = threading.Lock()
    client_handler = CloudGroup(user_list, lock, listener, sym_key)
    client_handler.start()
    running = True

    while running:
        command = raw_input('How may I help you?\n')
        argv = command.split(' ')
        # use lock to prevent anyone contacting CloudGroup at during
            management
        lock.acquire()

        # add user to list
        if argv[0] == 'add':
            user_list.add(argv[1])

        # remove user from list
        elif argv[0] == 'remove':
            user_list.remove(argv[1])
            reset(sym_key, drive, client_handler, sym_key_filename)

        # shut down CloudGroup
        elif argv[0] == 'quit':
            #save user files
            users = open('UserList','wb')
            pickle.dump(user_list,users)
            users.close()
            client_handler.close()
            running = False
            print 'Goodbye'

        # print valid users
        elif argv[0] == 'list':
            print 'Users:'
            for x in user_list:
                print(x)

        # print usage
        else:
            usage()
        lock.release()


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

Modules

Encryptor:

Drive Manager:

Key Manager: