Description:

We are using C programming language. We gathered two different codes from Github for the AES-256 encryption/decryption in counter mode and HMAC-SHA256. The AES-256 and HMAC-SHA256 code did not have a randomly generated key, so we have to implement that for both codes. Any arguments that need random values when encrypting or decrypting on AES or SHA uses a random value generator. Within these random value generator, there is a constant string where key, IV and tag values randomly select one character within the constant string until their length are statfied. The process of selecting a value is done by seeding time them using the mod operator to get one number which will serve as the index in the constant string. Afterwards, it is a matter of string copying index by index. After constructing a string that satisfies the length for KEY, IV and TAG it is outputted to the screen for shell script input and written to a file. The user will not have to execute any of these programs, but only the main.sh script.

Within the main script the user is given 3 options 1: encrypt 2: decrypt 3:exit

**1:encrypt**

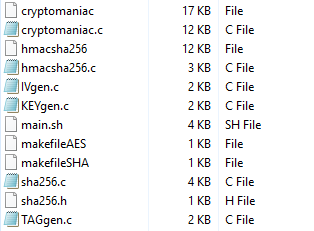
-When a user hits 1 the first thing they must do is have a text file locally and enter in its name.

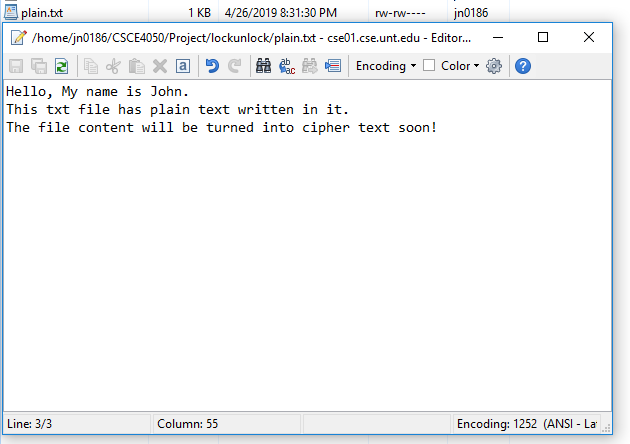
-Afterwards they are ask what they would like the cipher text file name to be. This file name will be the name that is uploaded and seen on google drive.

-Once the name is given, the script executes the 3 random value generator for KEY, IV and TAG setting the programs output to one of it’s own bash variable and spawning 3 text files with their message inside them

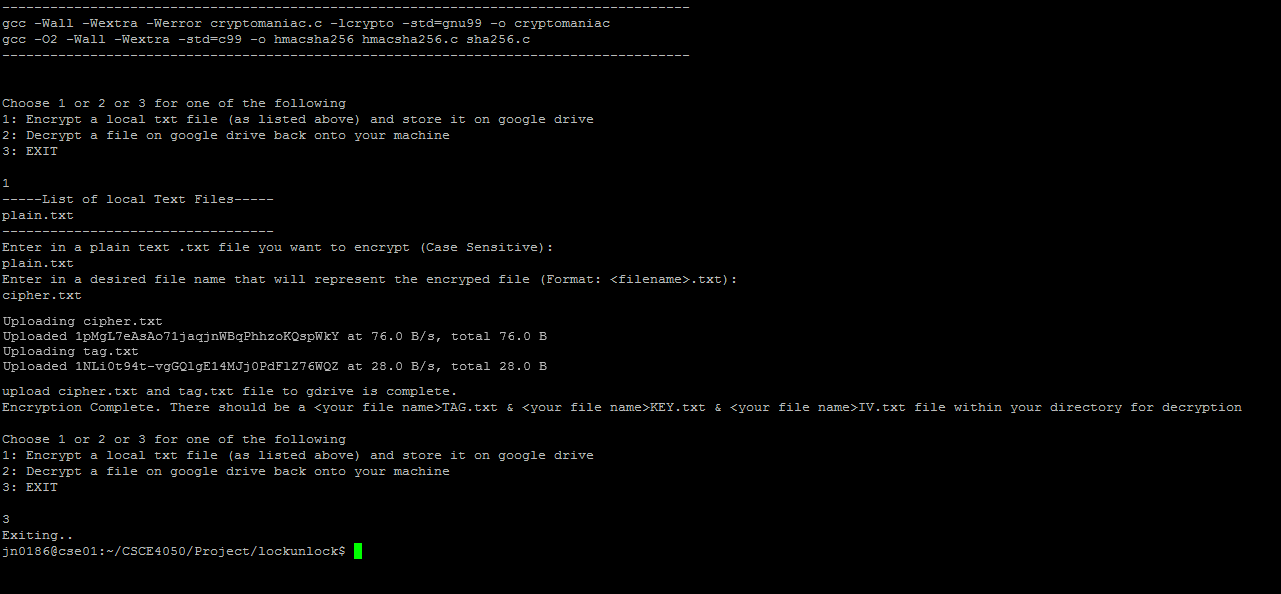
-Then the script runs the encryption where the input file is turned into the cipher text and the tag is generated.

-then both of these files are transfer to google drive via gdrive.

[Starting Files needed]



[Making my plain text file that I want to encrypt]



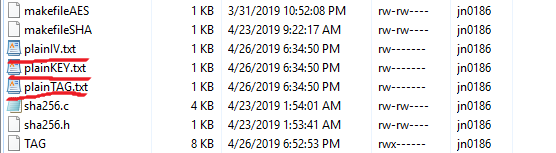
[The whole encryption process above:

Entering the plain text

cipher text name

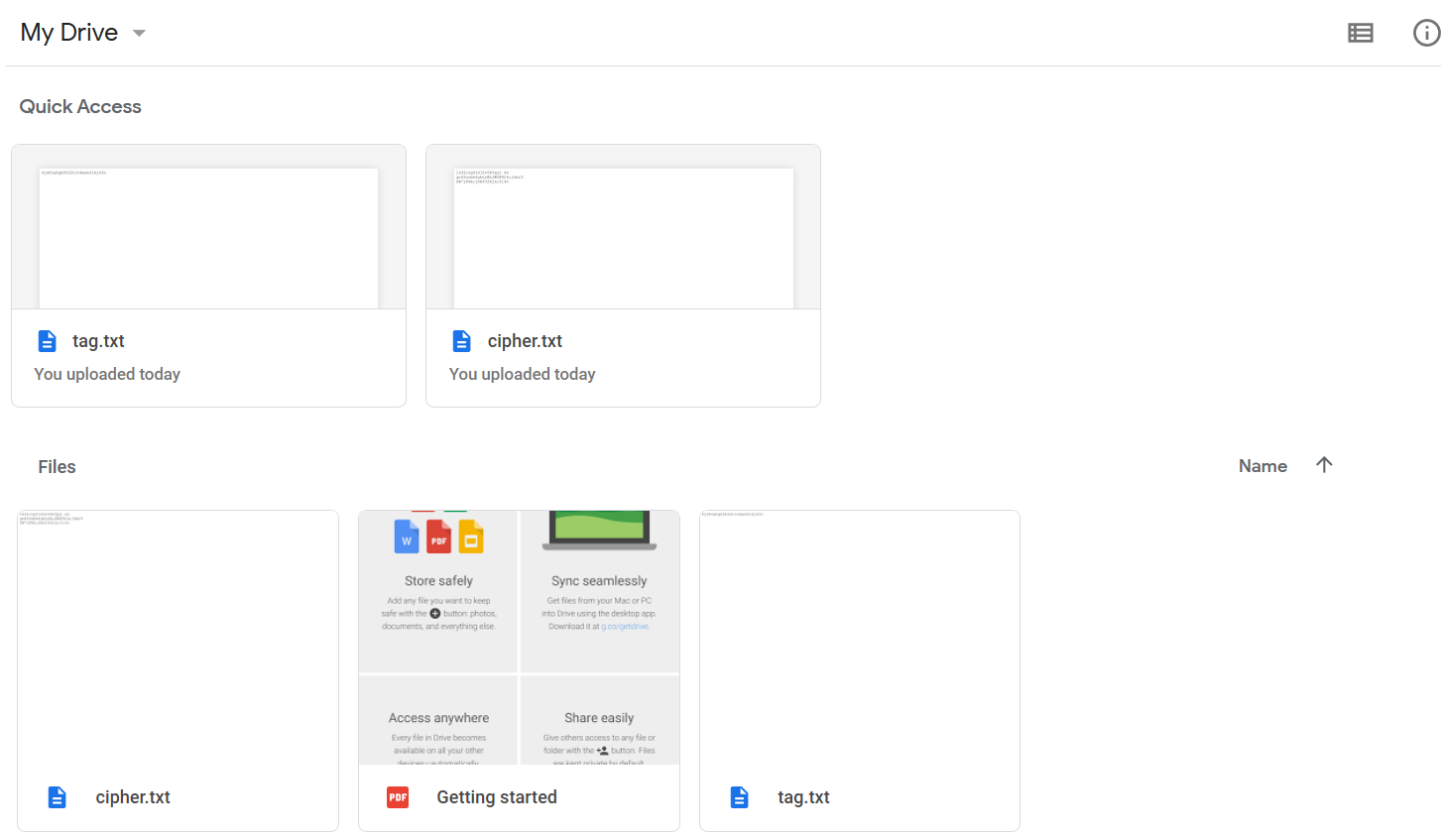
creating 3 text files with keys IV and tag

and uploading the cipher text and tag]

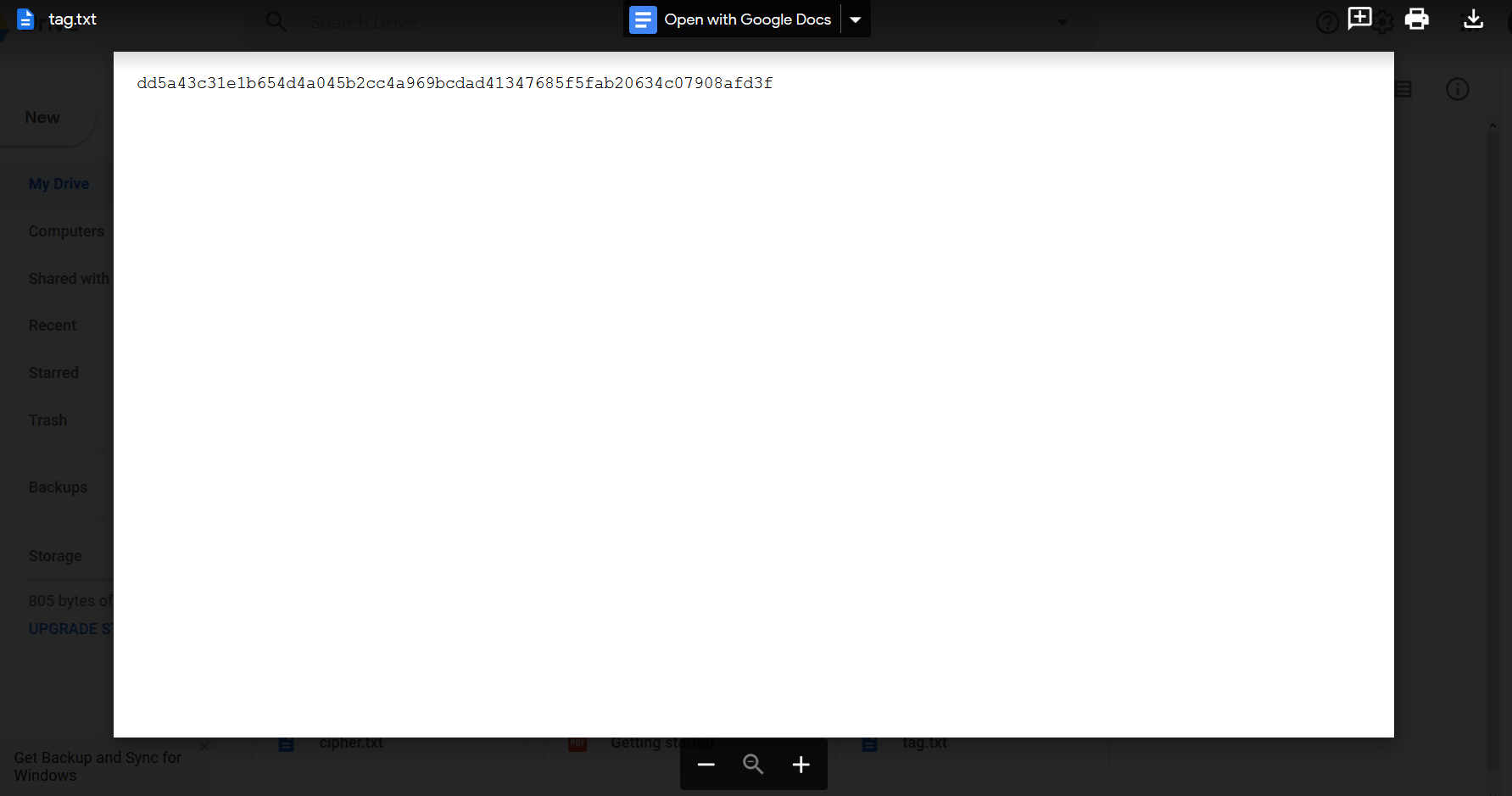


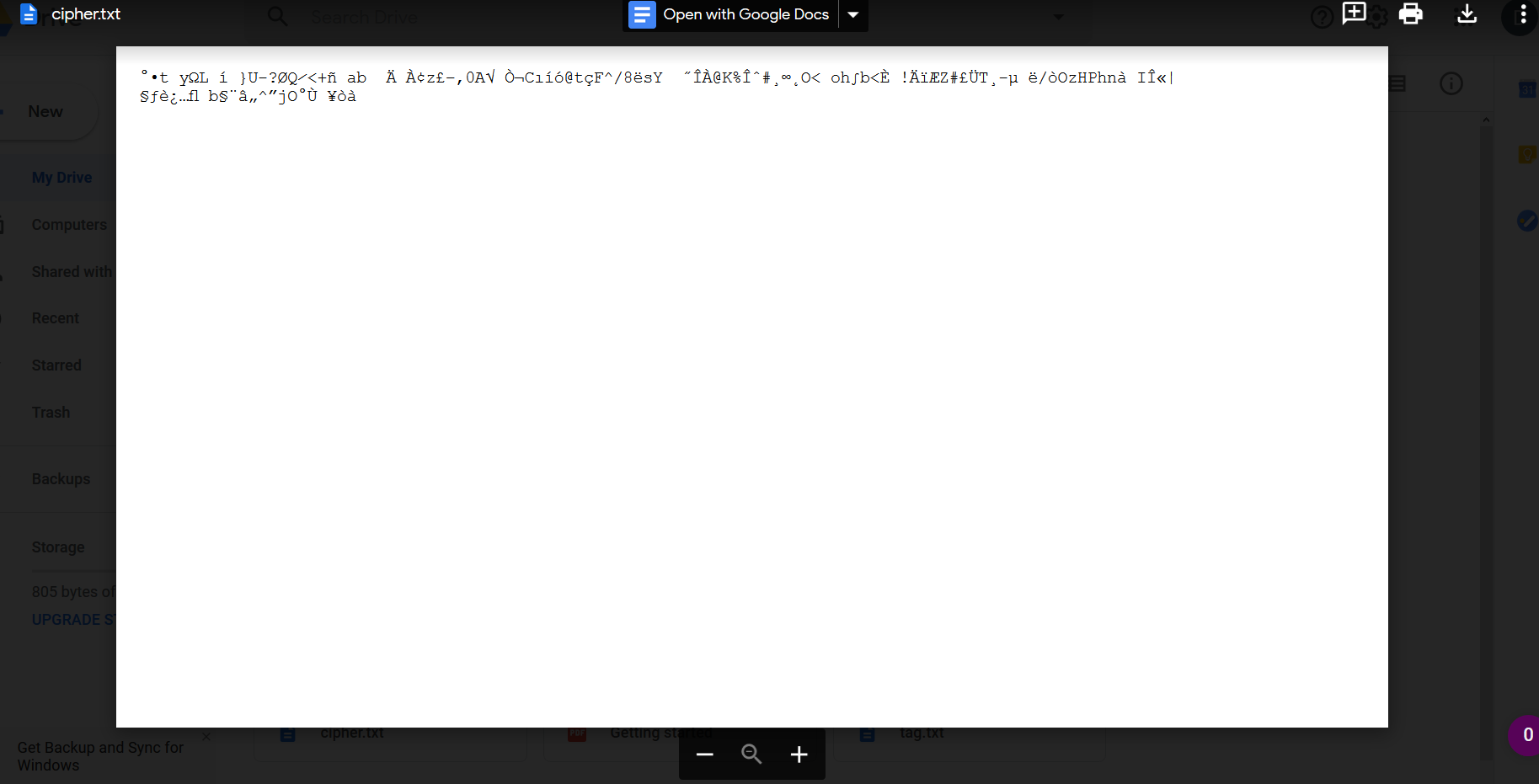
[After encrypting, I will see 3 .txt files within my directory each holding the IV, KEY and TAG.]

Within google drive I will now see:



With Tag being:



And Cipher being:

**2: Decryption**

-When the user hit decryption, they are shown all text files within their local directory

-They are then asked to enter in the file name on google drive that they wish to decrypt.

-Then they are asked to enter the file name that has the tag key.

* This tag files has a random string of characters which will serve as a key to SHA and generate a message which can check integrity.

- Gdrive will then download the file as written in the first step of of this process and the tag.txt file.

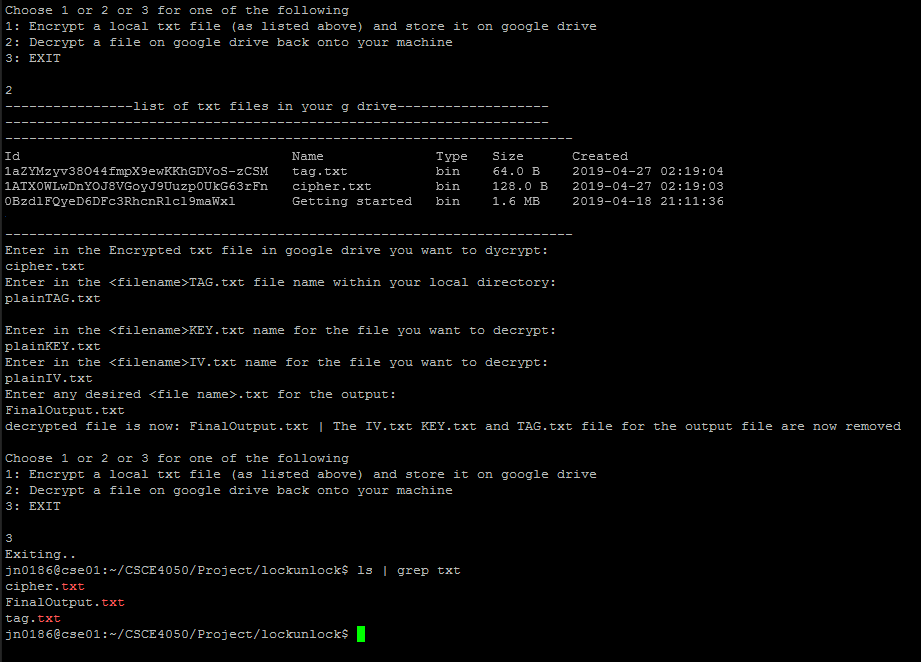
-An integrity will occur at this moment and what happens is the file that is downloaded from gdrive (tag.txt) will be read into a variable. This message that is read into this variable is the tag that was generated during encryption. another tag will be generated given that the user enter in the tag key with “cat <tag.txt> | ./hmacsha256 <tag key>”.A comparison is made between both variables, If they are different then the script will terminate, and not decrypt the file. However if the tags are the same then the decryption process will proceed. (I will provide pictures below to show what happen in both cases.)

-Afterwards the script will ask for the file names of the KEY and IV so that it can open the file and read the messages to a bash variable.

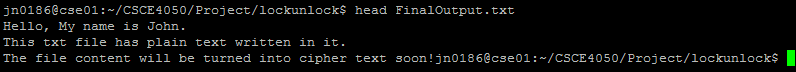
- then ask for an output file name for the decrypted message.

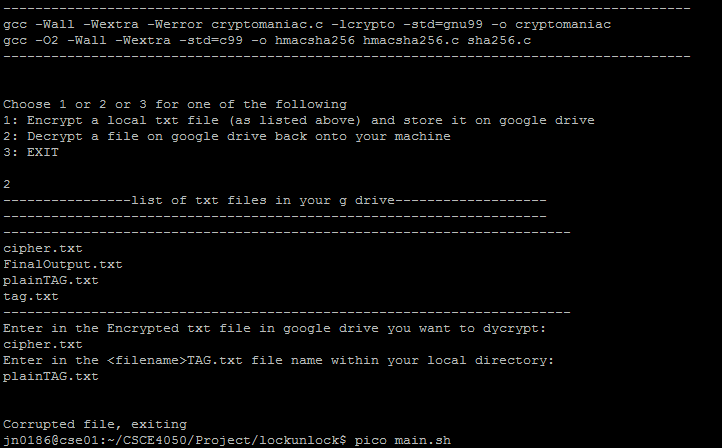
- Once all of the information is entered the script will run the decryption program and the output file will be generated. The TAG, KEY and IV text files will be removed.

Picture of Process:



Here You can see the list of files on g drive is displayed and once the user has entered in the “plainTAG.txt” there was no error afterwards, but the prompt for the Key, IV and, output file.

Once those file names are entered, a peek inside the output file will reveal the plain text message as seen in 2en picture under “encryption”

Here is a picture when the tag from gdrive does not match the tag generated from plainTAG.txt:

Note About Project.

Part of the reason why I added pictures is give visual representation and so that if you cannot run the code on your machine because UNT won't let you install gdrive library, the pictures explain what my code does.

However,

I will submit all required code to run the above processes. you must remember that you got to install gdrive in order for my script to work. link on how to install gdrive via linux terminal:

(link here:<https://www.techpunch.xyz/2018/07/how-to-upload-files-to-gdrive-through.html>)

Let me know if you have questions or need me to come execute this program for you in person.

Reference:

Alin Tomescu

For the AES encryption code – cryptomaniac.c

<https://github.com/alinush/cryptomaniac>

Mike J Savage

For the MAC SHA256 code – hmacsha256.c

<https://github.com/mikejsavage/hmac>