Republic of the Philippines
CAMARINES SUR POLYTECHNIC COLLEGES
Nabua, Camarines Sur

COLLEGE *of* COMPUTER STUDIES

CRYPTOGRAPHY APPLICATION

CSEC 329 - Applied Cryptography

FINAL PROJECT

Submitted by:

Bernaldez, Karl Santiago B.

Daquioag, James N.

Villanueva, Lyndsy D.

BSCS-3B (Group-8)

INTRODUCTION

The Symmetric and Asymmetric Encryption, and Hashing App is a desktop application that provides users with tools to encrypt and decrypt files using symmetric and asymmetric encryption algorithms, as well as generate and verify the integrity of data using hash functions. The application is built using Python and utilizes several third-party libraries such as *PyQt5*, *pycrypto*, and *cryptography* to implement the cryptographic functionalities.

Encryption is the process of transforming plain text into a non-readable form, known as ciphertext, to prevent unauthorized access and protect sensitive data. Symmetric encryption algorithms use the same secret key to both encrypt and decrypt data, while asymmetric encryption algorithms use a public key to encrypt data and a private key to decrypt data. Hash functions, on the other hand, generate fixed-length unique representations of data, which can be used to verify the integrity of the data or detect unauthorized modifications.

The Symmetric and Asymmetric Encryption, and Hashing App offers an intuitive and user-friendly interface, which allows users to select files to encrypt or decrypt, choose encryption or decryption algorithms, and generate and verify hash values. The application provides an easy-to-use platform for users to ensure the confidentiality, integrity, and authenticity of their data, making it suitable for individuals and businesses alike.

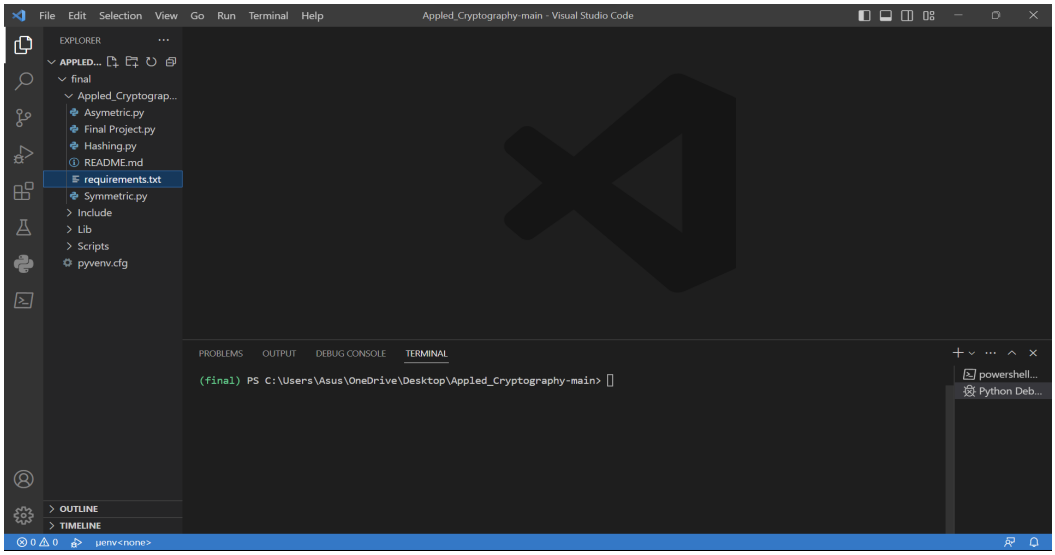
PROJECT OBJECTIVES:

1. To provide a user-friendly interface for encrypting and hashing files using both symmetric and asymmetric encryption techniques, as well as SHA-256 hashing algorithm.
2. To enable users to easily verify the integrity of files by comparing their hash values with the original hashed values.
3. To implement secure and reliable encryption and hashing methods using Fernet-AES, RSA, and SHA-256 to ensure the confidentiality, authenticity, and integrity of user data.

Sample Run

Step 1:

Load the folder of the app in your IDE , here we use VS Code to run our application and we use a virtual environment named final.



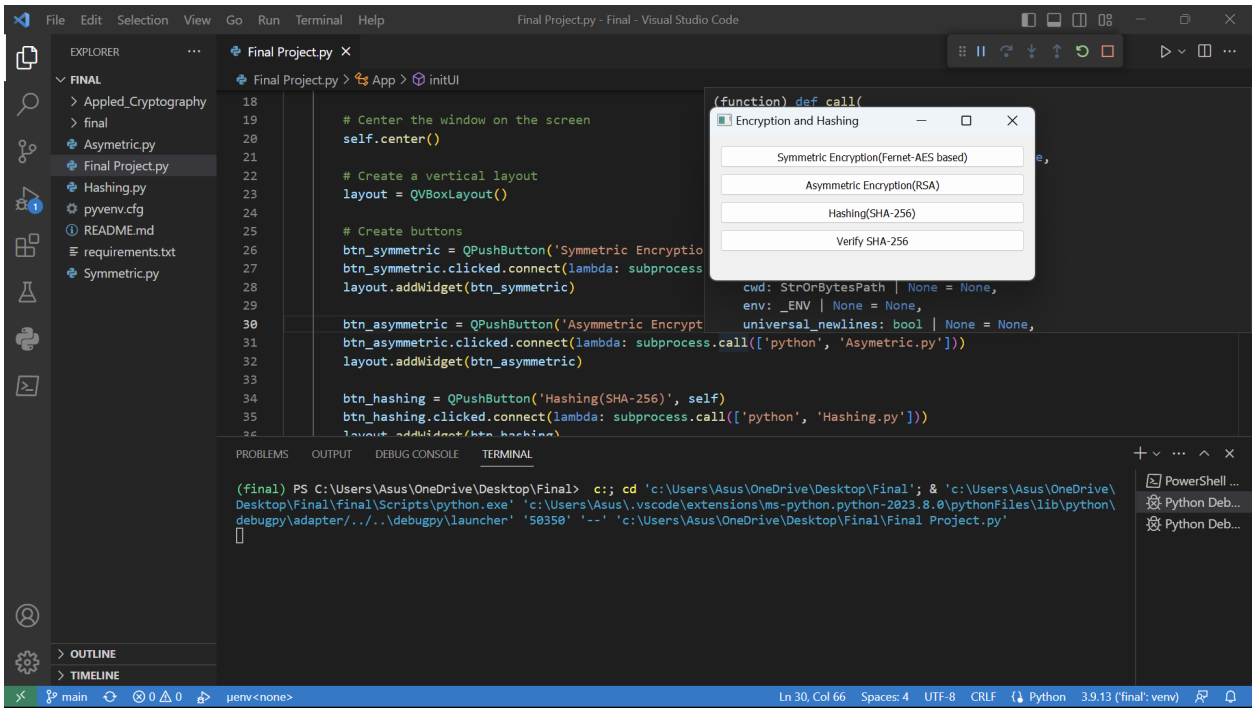
Step 2: First you are expected to have python installed in your machine. Then install the requirements.txt using the command **pip install -r** [requirements.txt]

```
PS D:\ACADS\CS\3RD C\2ND SEM\APPLIED CRYPTOGRAPHY\FINALS\Appled_Cryptography-main\final\Appled_Cryptography-main> pip install -r requirements.txt
Collecting certifi==2023.5.7
  Downloading certifi-2023.5.7-py3-none-any.whl (156 kB)
     |████████████████████████████████| 156 kB 181 kB/s
Collecting cffi==1.15.1
  Downloading cffi-1.15.1-cp310-cp310-win_amd64.whl (179 kB)
     |████████████████████████████████| 179 kB 467 kB/s
Collecting charset-normalizer==3.1.0
  Downloading charset_normalizer-3.1.0-cp310-cp310-win_amd64.whl (97 kB)
     |████████████████████████████████| 97 kB 595 kB/s
Collecting crypto==1.4.1
  Downloading crypto-1.4.1-py2.py3-none-any.whl (18 kB)
Collecting cryptography==40.0.2
  Downloading cryptography-40.0.2-cp36-abi3-win_amd64.whl (2.6 MB)
     |████████████████████████████████| 2.6 MB 819 kB/s
Requirement already satisfied: idna==3.4 in c:\python\python3101\lib\site-packages (from -r requirements.txt (line 6)) (3.4)
Collecting Naked==0.1.32
  Downloading Naked-0.1.32-py2.py3-none-any.whl (587 kB)
     |████████████████████████████████| 587 kB 819 kB/s
Collecting pycparser==2.21
  Downloading pycparser-2.21-py2.py3-none-any.whl (118 kB)
     |████████████████████████████████| 118 kB 1.7 MB/s
  Downloading PyQt5_sip-12.12.1-cp310-cp310-win_amd64.whl (78 kB)
     |████████████████████████████████| 78 kB 2.6 MB/s
Collecting PyYAML==6.0
  Using cached PyYAML-6.0-cp310-cp310-win_amd64.whl (151 kB)
Collecting requests==2.30.0
  Downloading requests-2.30.0-py3-none-any.whl (62 kB)
     |████████████████████████████████| 62 kB 297 kB/s
Collecting shellescape==3.8.1
  Downloading shellescape-3.8.1-py2.py3-none-any.whl (3.1 kB)
Collecting urllib3==2.0.2
  Downloading urllib3-2.0.2-py3-none-any.whl (123 kB)
     |████████████████████████████████| 123 kB 1.1 MB/s
Installing collected packages: urllib3, charset-normalizer, certifi, requests, PyYAML, pycparser, shellescape, PyQt5-sip, PyQt5-Qt5, Naked, cffi, PyQt5, pycryptodomex, pycr
yptodome, cryptography, crypto
Successfully installed Naked-0.1.32 PyQt5-5.15.9 PyQt5-Qt5-5.15.2 PyQt5-sip-12.12.1 PyYAML-6.0 certifi-2023.5.7 cffi-1.15.1 charset-normalizer-3.1.0 crypto-1.4.1 cryptograp
hy-40.0.2 pycparser-2.21 pycryptodome-3.17 pycryptodomex-3.17 requests-2.30.0 shellescape-3.8.1 urllib3-2.0.2
```

```
PS D:\ACADS\CS\3RD C\2ND SEM\APPLIED CRYPTOGRAPHY\FINALS\Appled_Cryptography-main\final\Appled_Cryptography-main> python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\python\python3101\lib\site-packages (21.2.4)
Collecting pip
  Downloading pip-23.1.2-py3-none-any.whl (2.1 MB)
     |████████████████████████████████| 2.1 MB 595 kB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 21.2.4
    Uninstalling pip-21.2.4:
      Successfully uninstalled pip-21.2.4
Successfully installed pip-23.1.2
PS D:\ACADS\CS\3RD C\2ND SEM\APPLIED CRYPTOGRAPHY\FINALS\Appled_Cryptography-main\final\Appled_Cryptography-main>
```

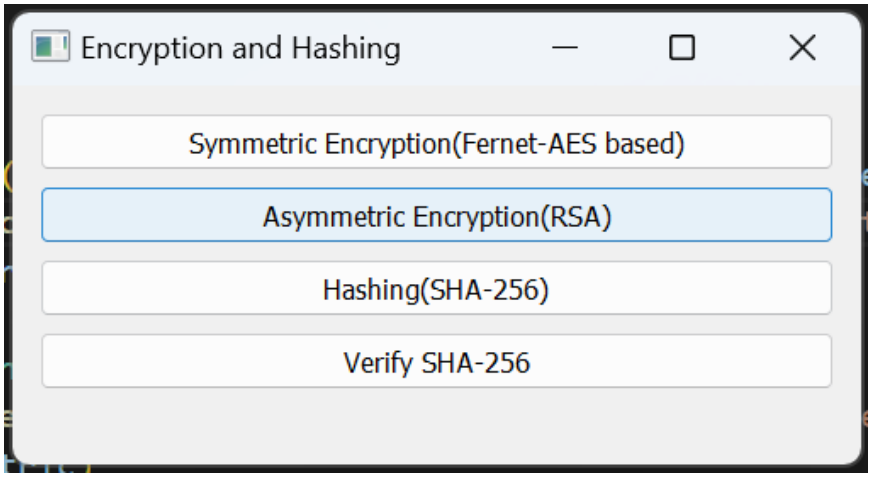
Step 3:

Now were going to run the Main File of the app which is named [Final Project.py]() Using the **Ctrl + F5 or CTRL + FN + F5.**



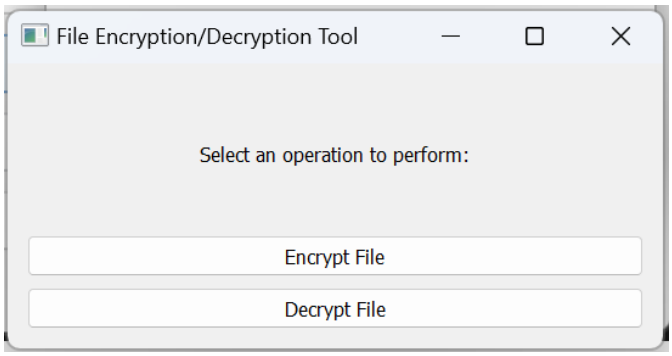
Step 4:

Select which function (Symmetric and Asymmetric Encryption, Hashing and Verification of SHA-256)



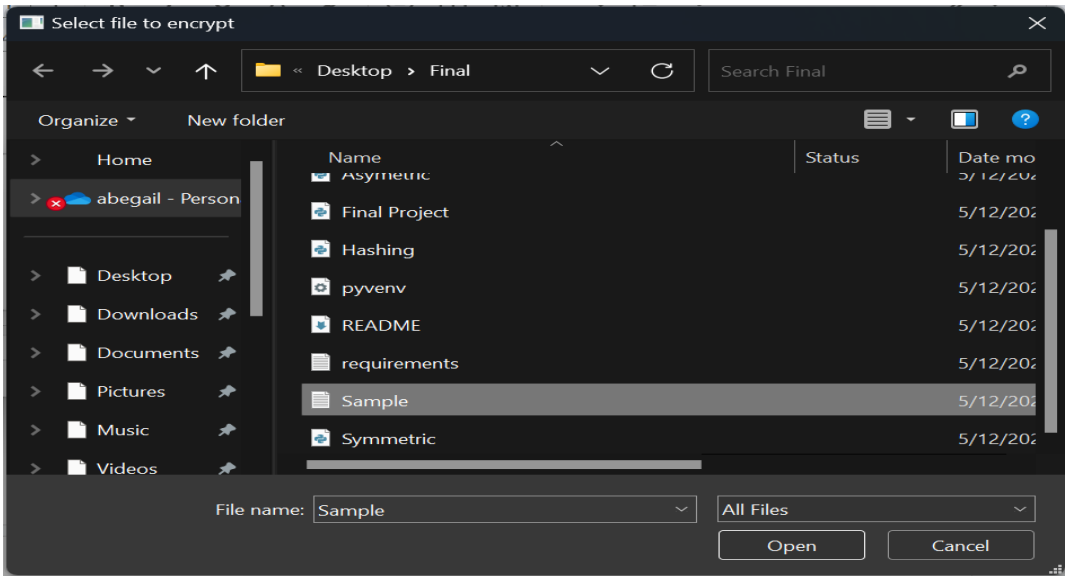
Step 4A: Symmetric Encryption(Fernet-AES Based)

Select an operation to perform:

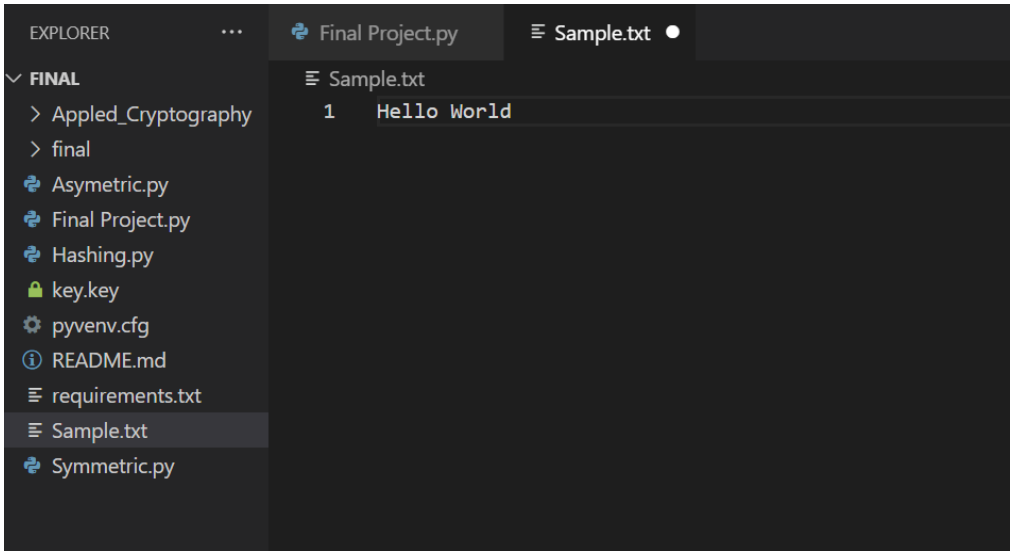


Step 4A: Encrypt File

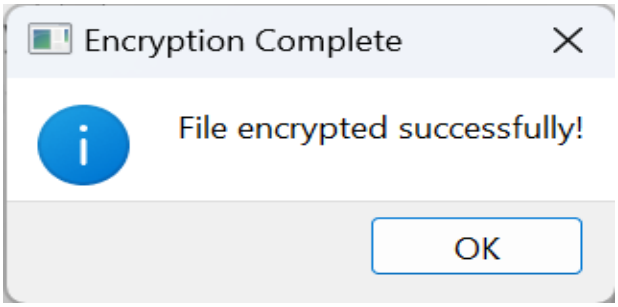
Select file to encrypt. Here we load the [sample.txt]



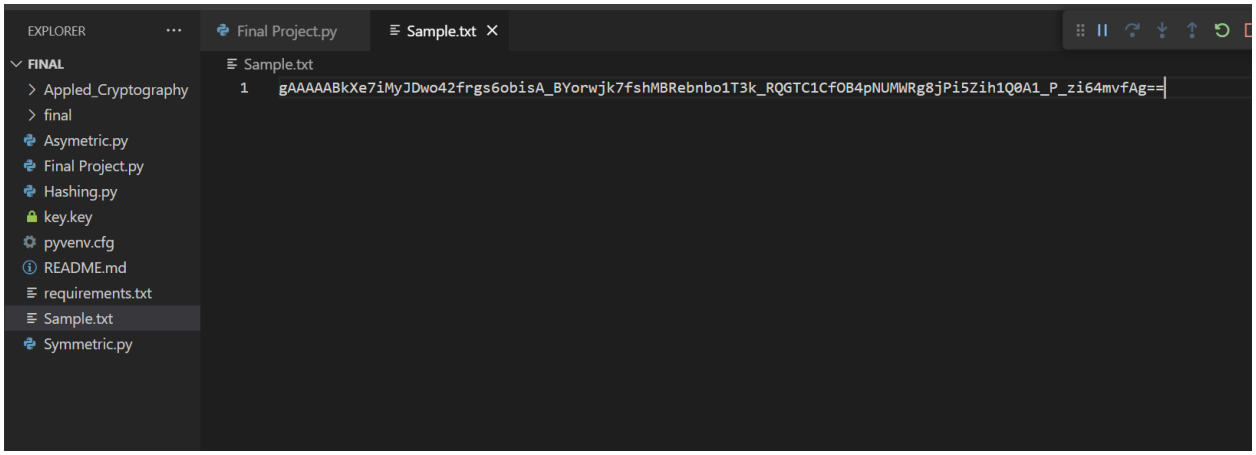
The Sample.txt contains “Hello World” before encryption.



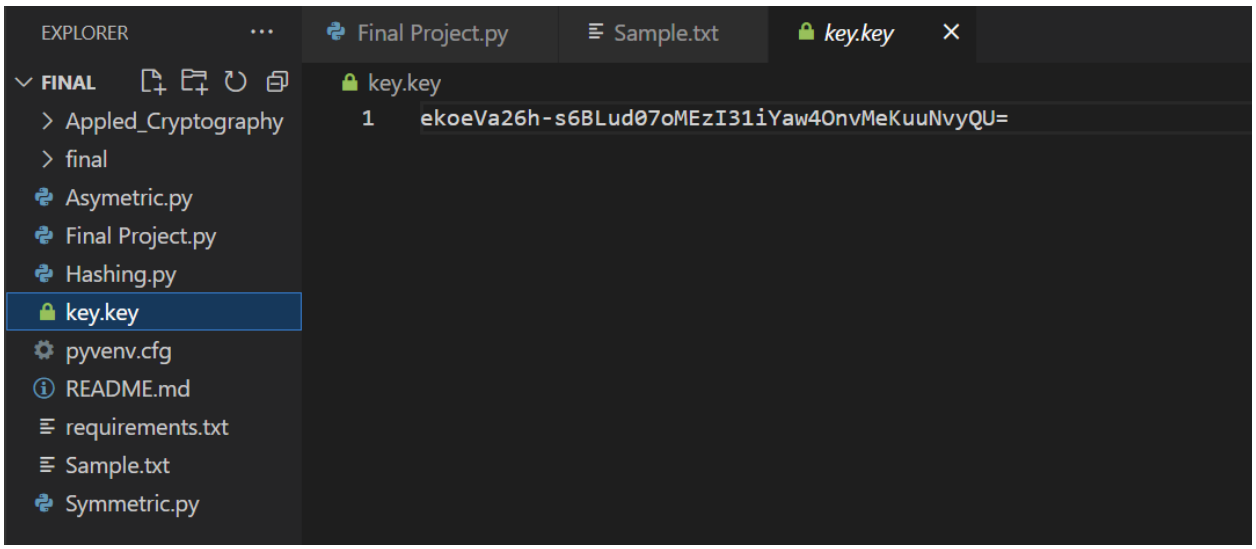
Now after successfully encrypting the file. A window will pop up saying that the encryption is completed successfully.



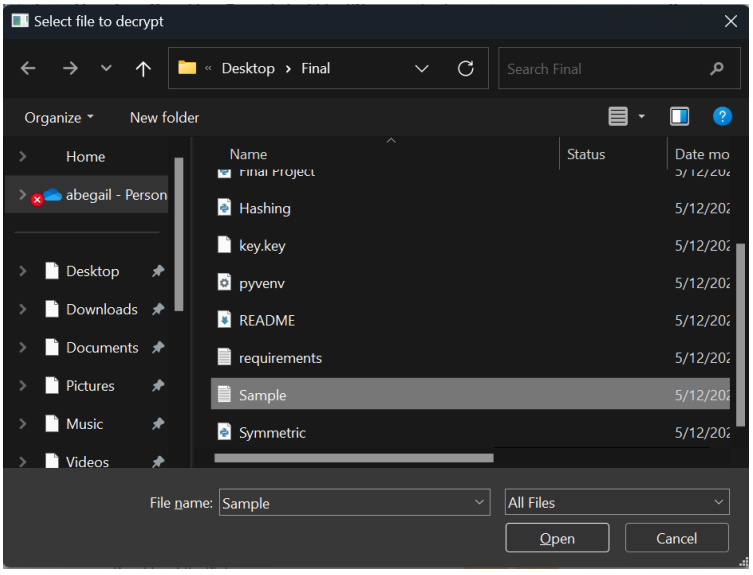
After Encryptioin the Sample.txt contains the encrypted string of the “Hello World”



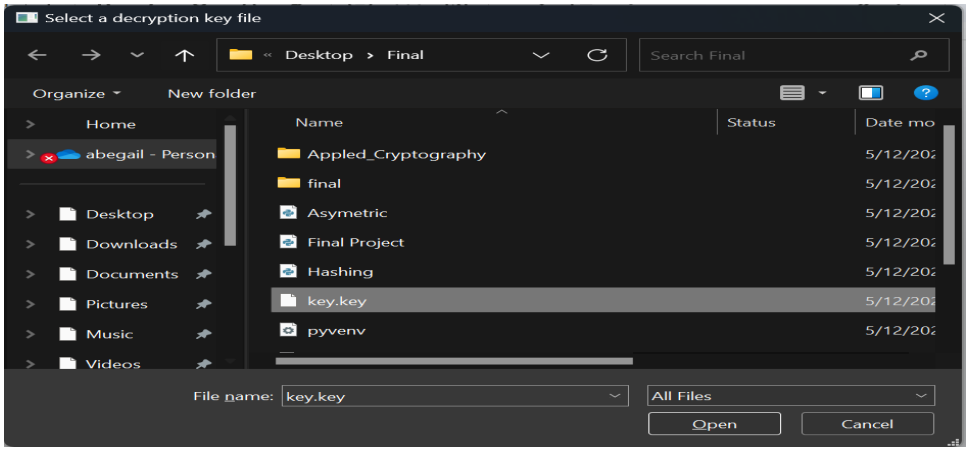
And the key file is also generated and can be use to decrypt.



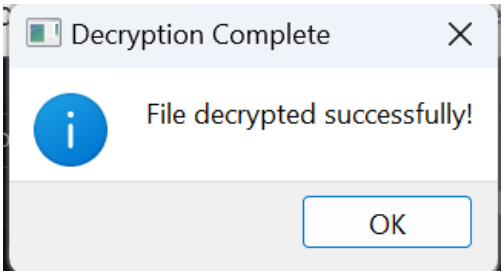
Now lets try to decrypt the file. First you need to load the file to decrypt.



Then, select the decryption key and hit “Open”. Here the decryption key is named key.key.

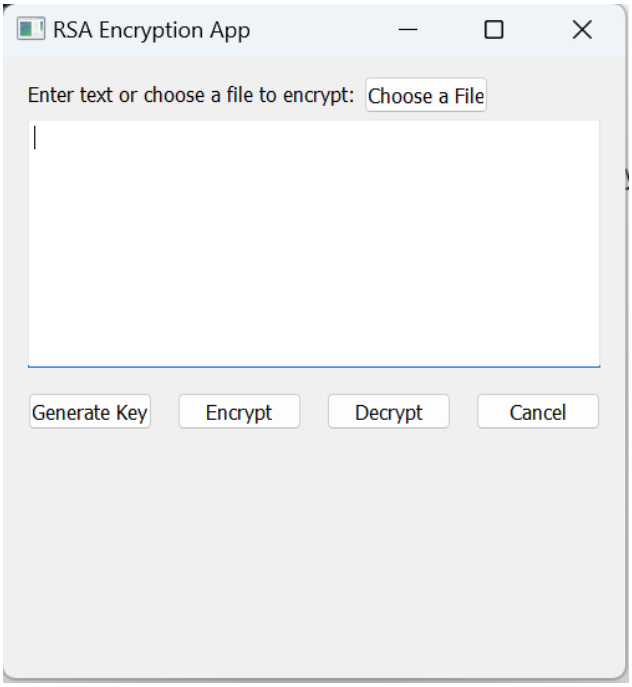


Now after successfully decrypting the file. A window will pop up saying that the decryption is completed successfully.

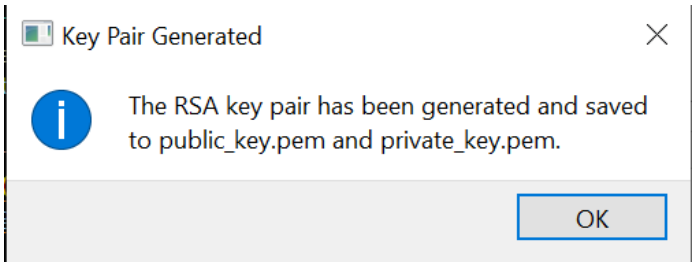


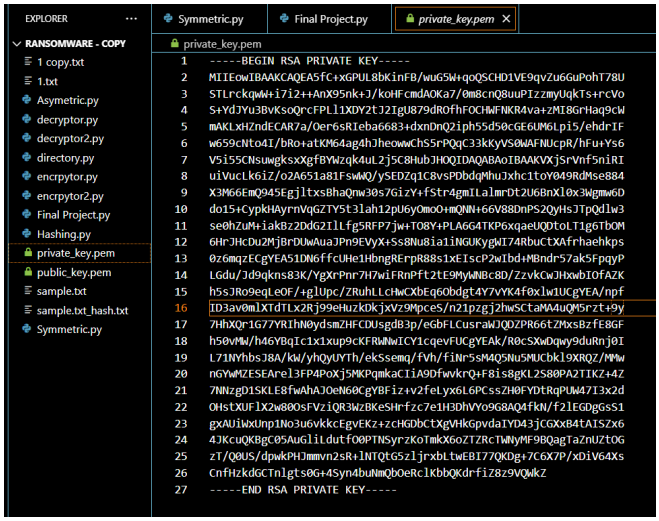
Step 4B: Asymmetric Encryption(RSA)

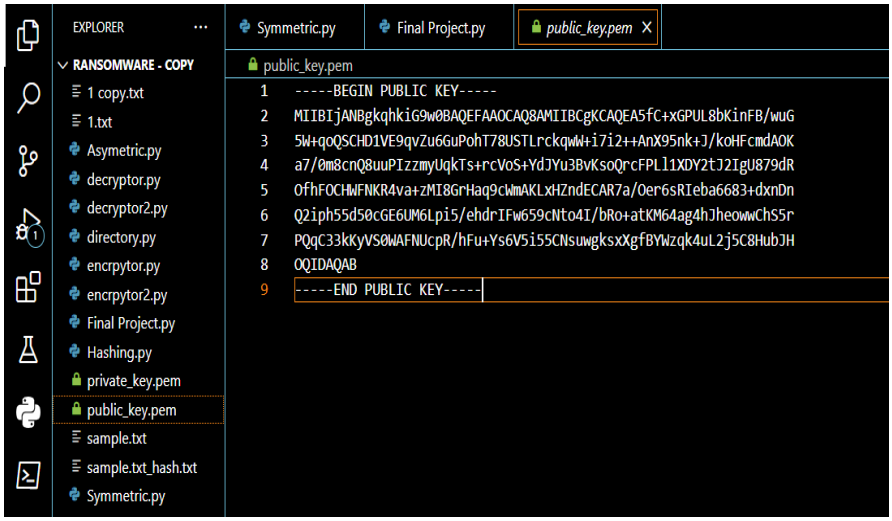
Choose which operation you want to perform. If you dont have a public key and private key , you can generate a pair key using the **RSA Encryption App .**



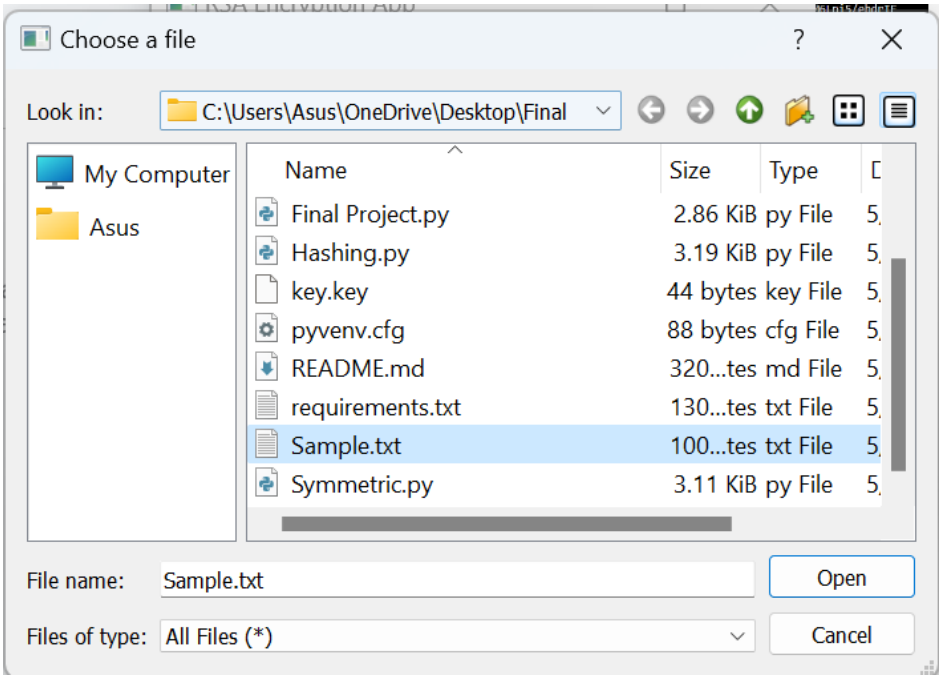
Enter text or Choose a file to encrypt using RSA. But if you dont have a public key and a private key, you can generate it by clicking the **Generate Key** button.



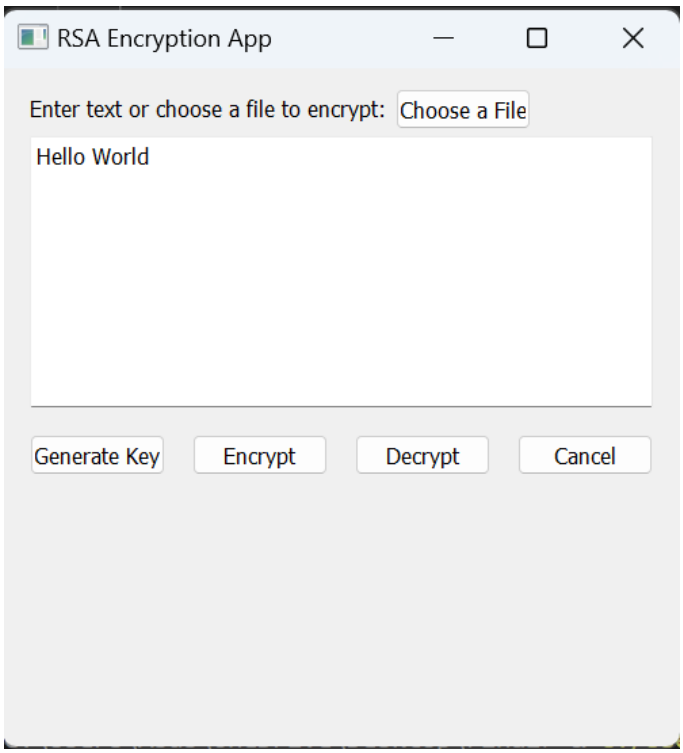




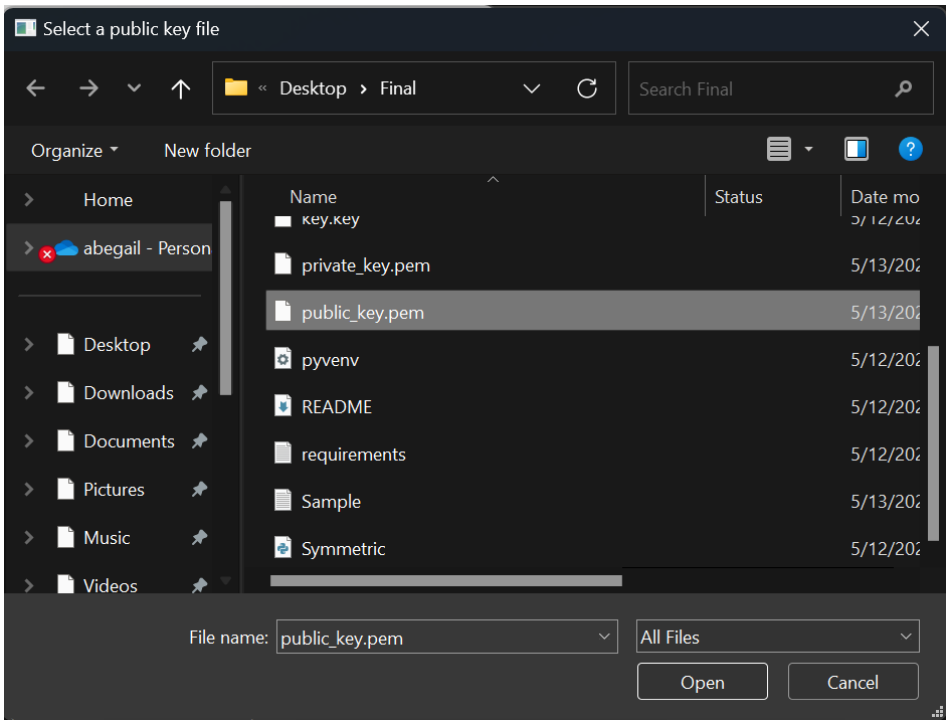
Now input the string or Choose a file to be encrypted.



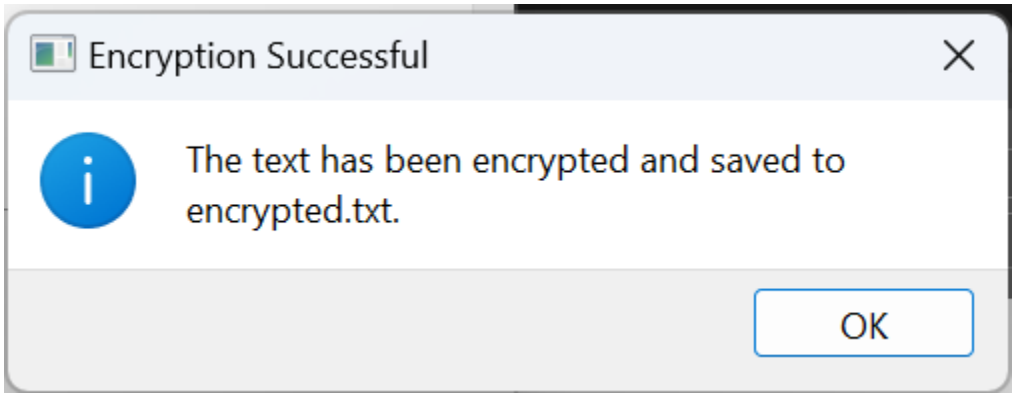
Now after loading the file the content of the file should appear on the text box.



To encrypt you need to load the public key file.

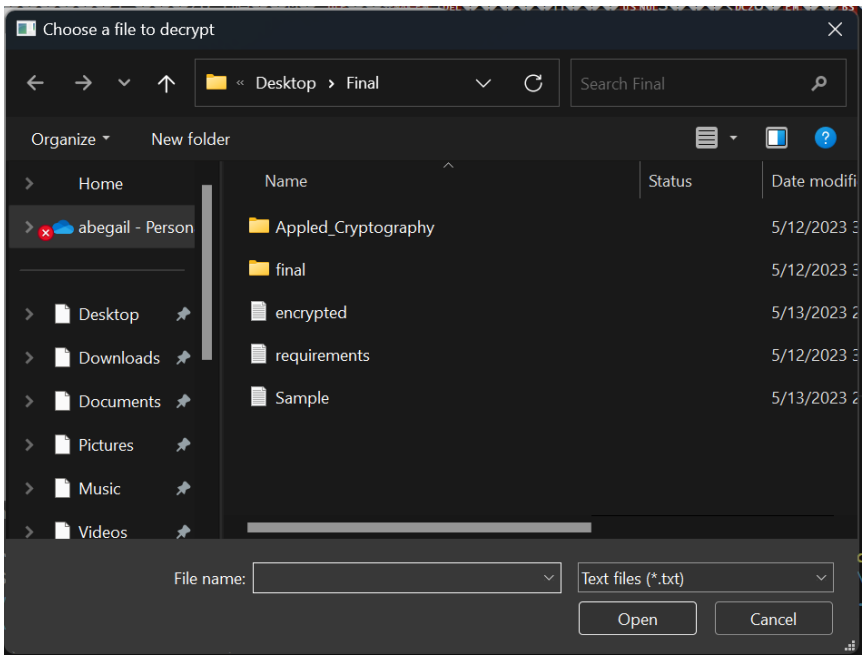


After loading the public key file it will automatically encrypt the content of the loaded file. A window should appear stating that the encryption is successfu and it will generate a new file named encrypted.txt .

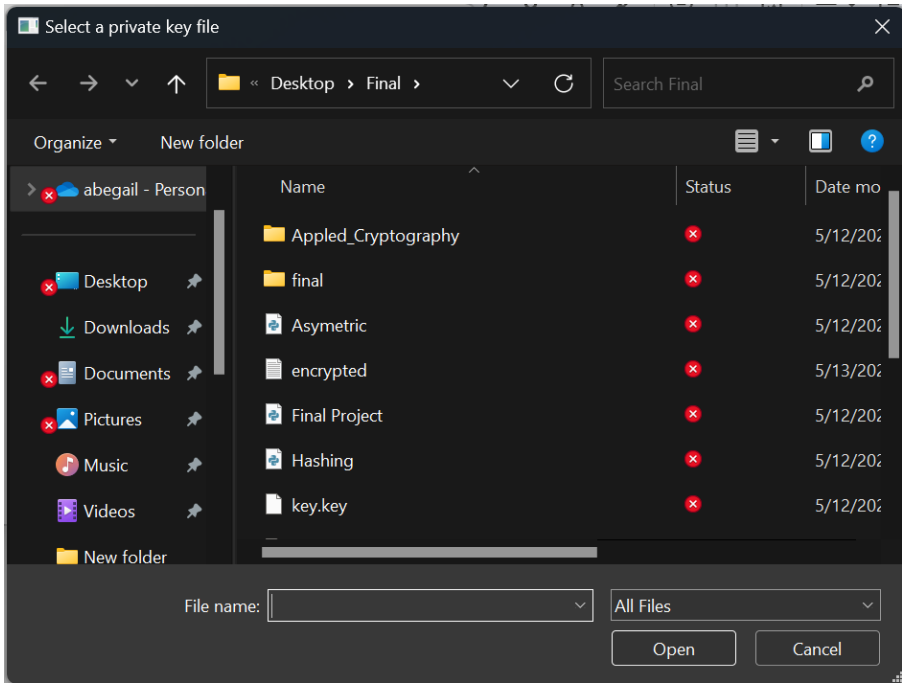


Here is the encrypted version of the loaded file.

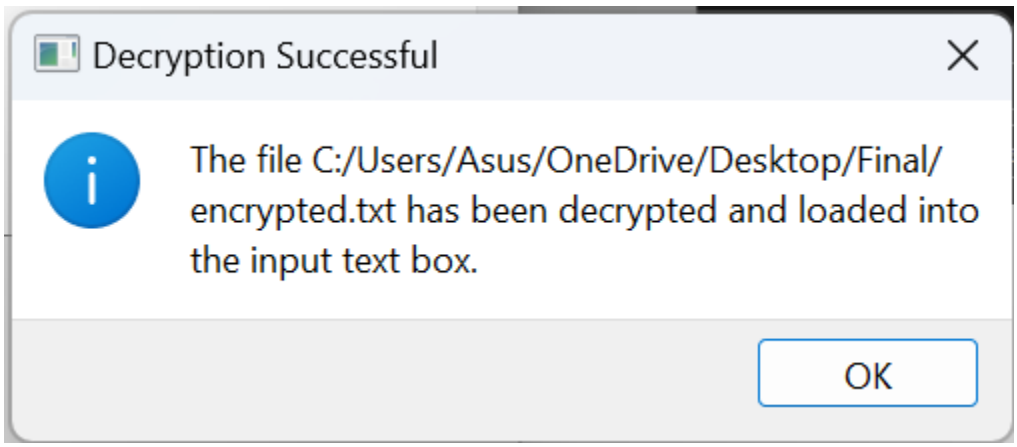
Now to decrypt this were going to load this in the file dialog box.



and we need to load the private key that we generate earlier.

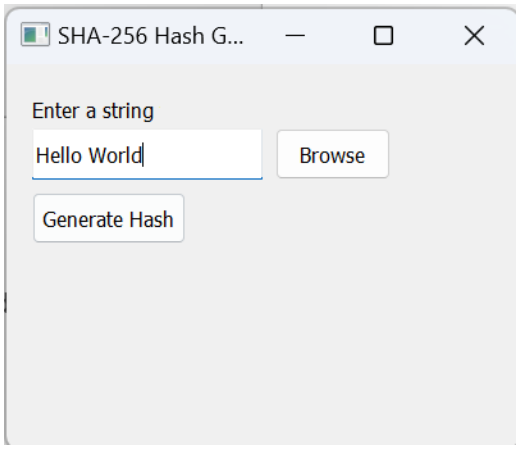


After loading the private key file, a window will appear saying that the Decryption is successful.

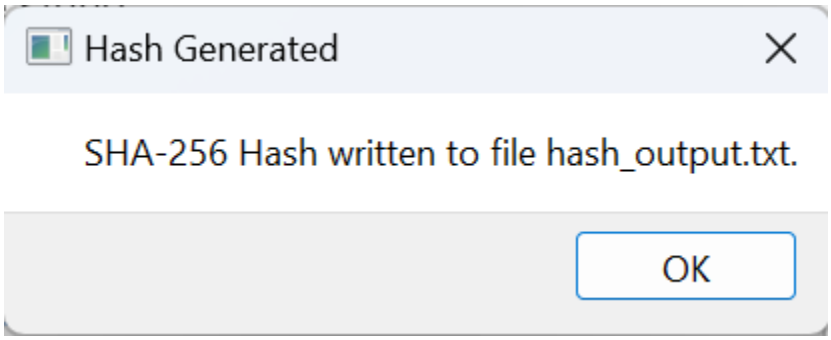


Step 4B: Hashing(SHA-256)

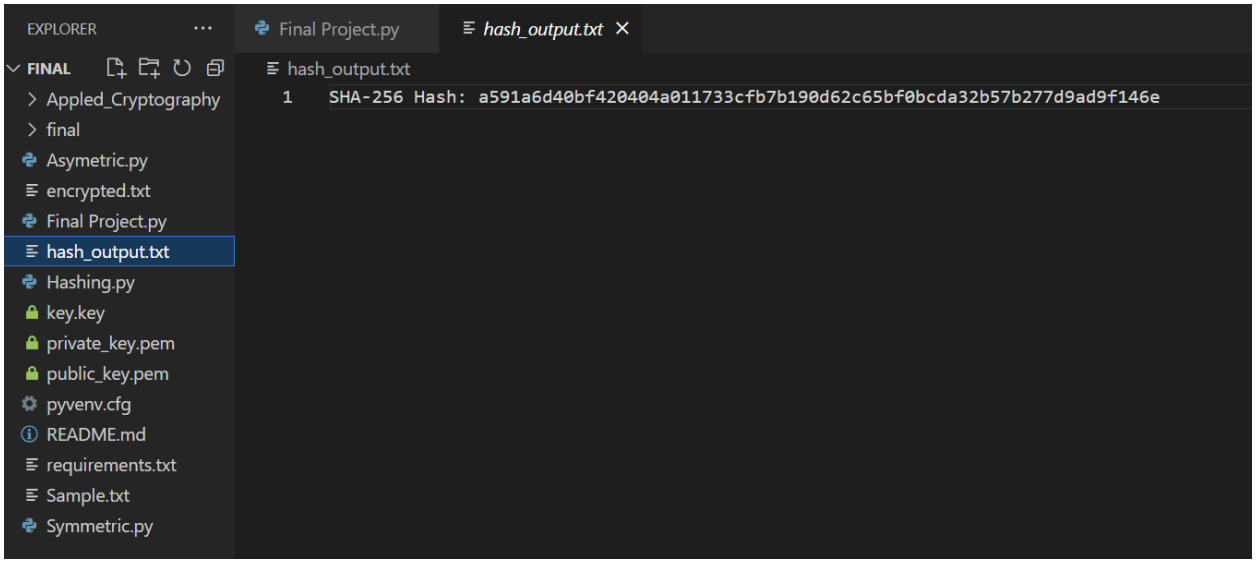
Enter a string in the text box or choose a file to be hashed by SHA-256, here we use the text box to input the string.



After clicking the **Generate Hash** a window pop up will appear saying that SHA-256 Hash is written to file hash_output.txt.

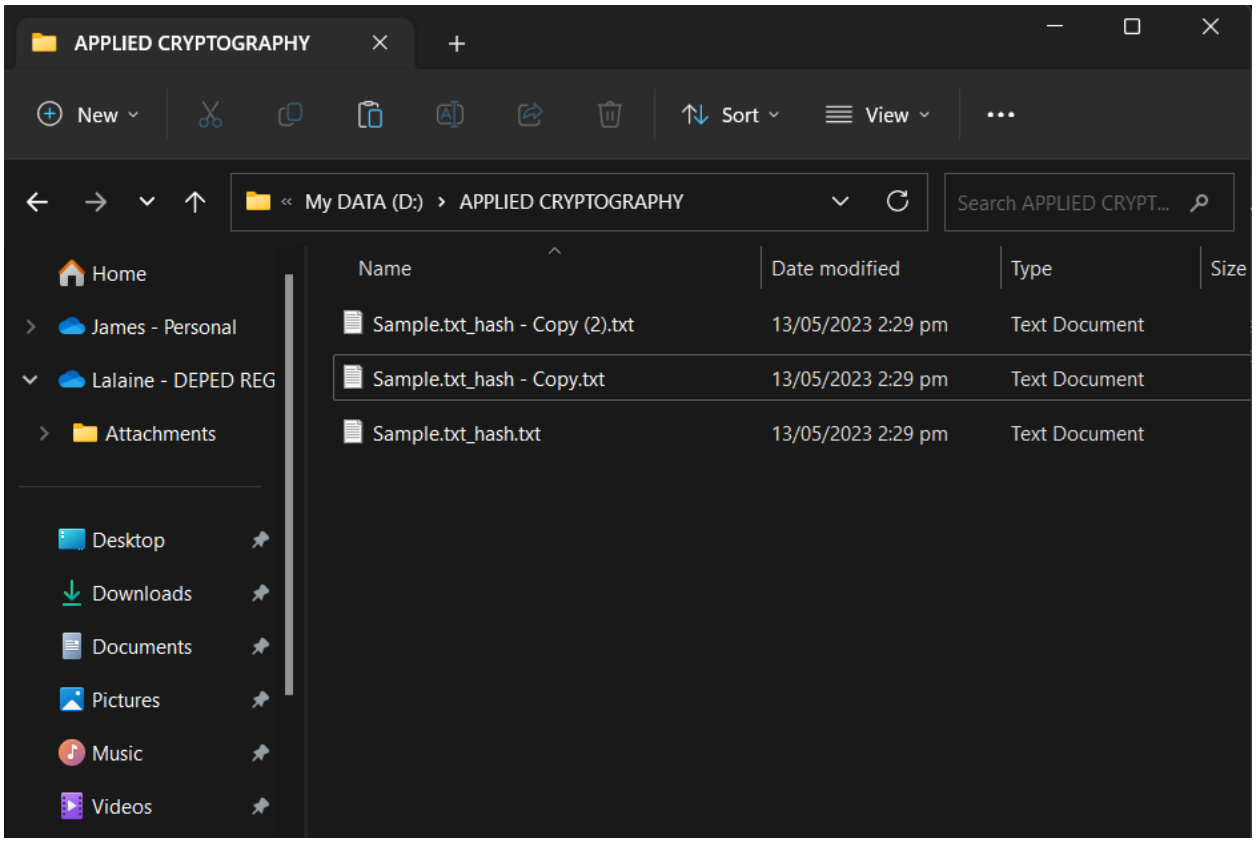


This is the hashed version of the message “Hello World”.

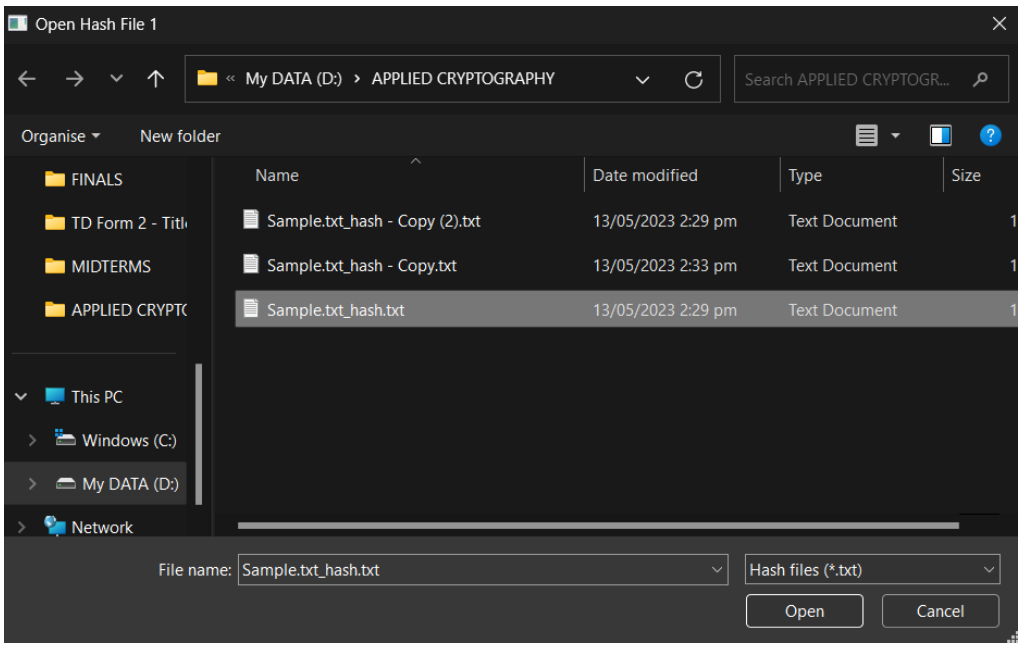


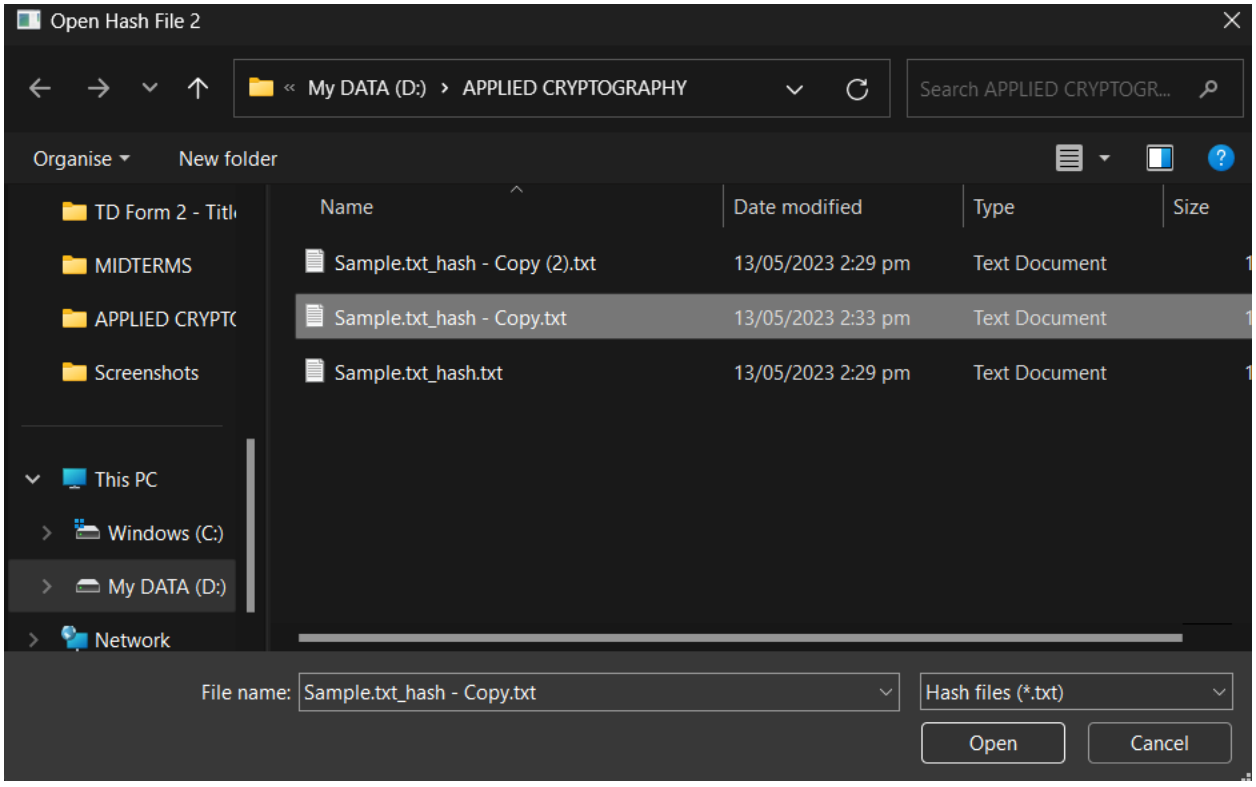
Step 4B: Verify SHA-256

Here we have a 3 samples that were going to compare and verify if the hashed value is the same.

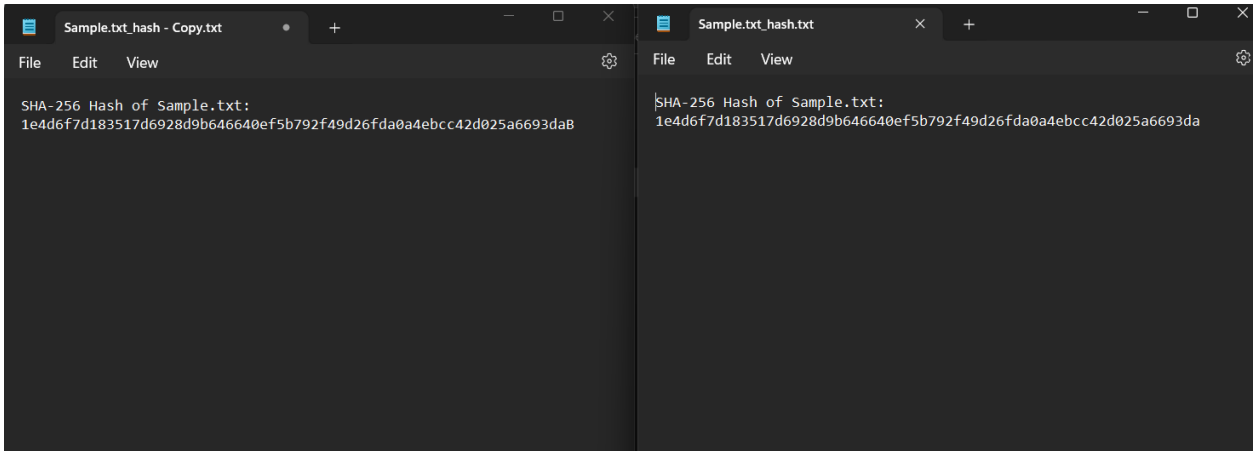


Here we going to load the 1st file and 2nd file to be verified.

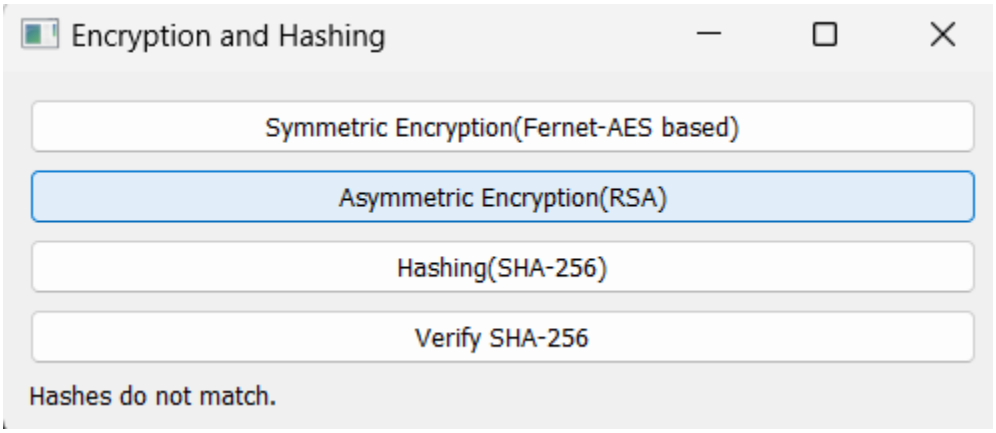




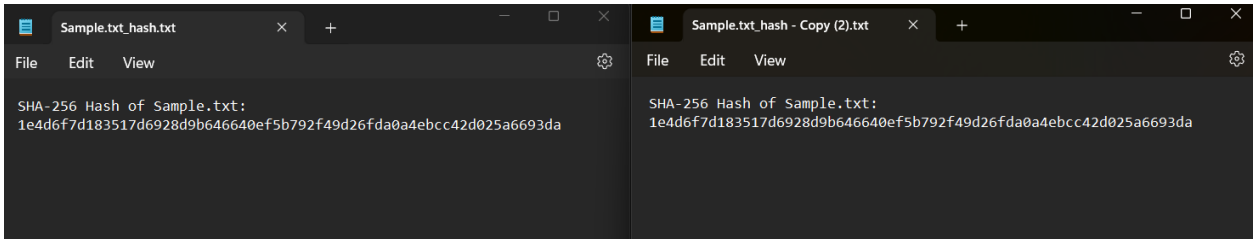
Now let's go to the first case which is the hashed value is not the same.



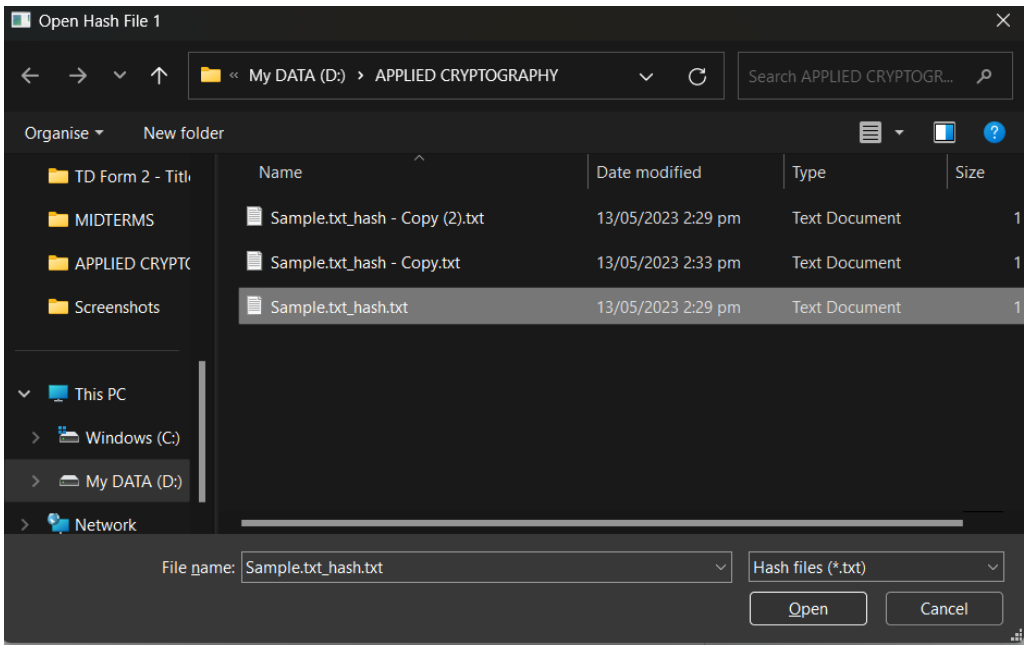
A message "**Hashed do not match**" on the lower left will appear in this case.



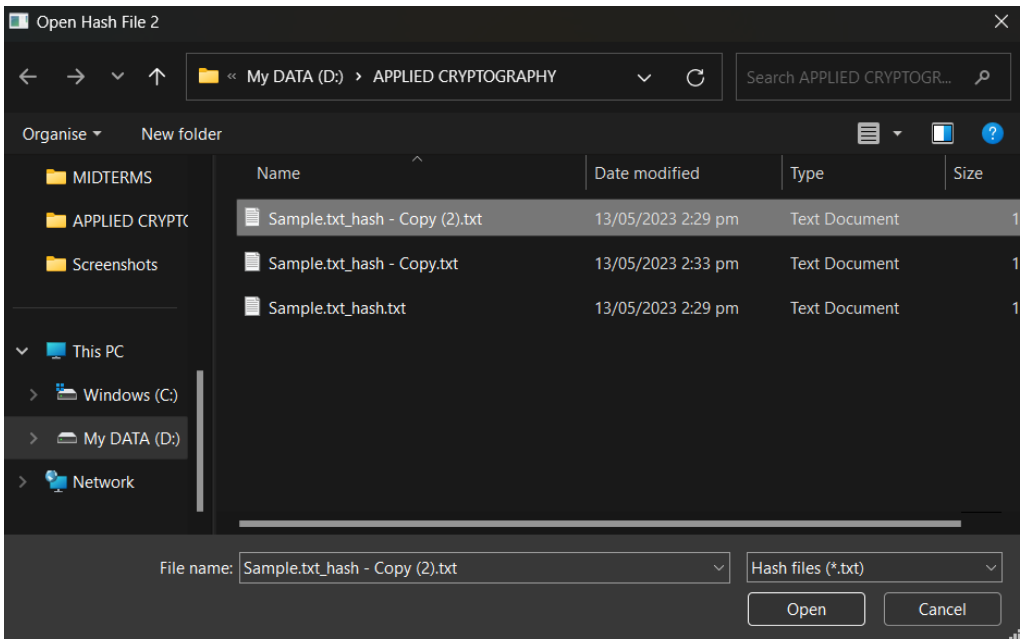
Now lets move on the next case which is the same hashed value.



Again let's load first the control file of the Hash File 1.



and load the next file or the Hash File 2.



After loading the 2 files which have the same hashed value content. A message “Hashes match!” on the lower left.