NETWORK SECURITY
LAB 2 REPORT

NAME: TRAN THI THOA
ID: S1242006

Problem 1: DES encryption/ decryption

The implementation of the tool to encrypt and decrypt files using DES in mode CBC is in the file “2019.07.15.s1242006.tempdes_cbc.c”

To compile and run the file, run the following command lines (on Mac OS):

gcc -o tempdes_cbc 2019.07.15.s1242006.tempdes_cbc.c -lssl -lcrypto -L/usr/local/opt/openssl/lib -I/usr/local/opt/openssl/include

./tempdes_cbc fedcba9876543210 40fedf386da13d57 test.txt my_test.des

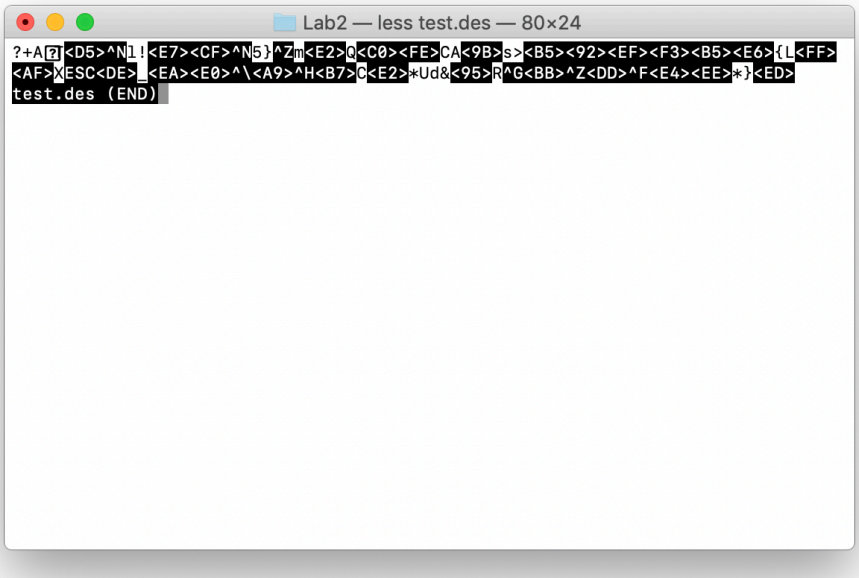
The result will be:

We can see that the encrypted texts by using built-in function des_cbc_encrypt and my DES-CBC function are the same.

Then, to see file test.des, run the following command line:

less test.des

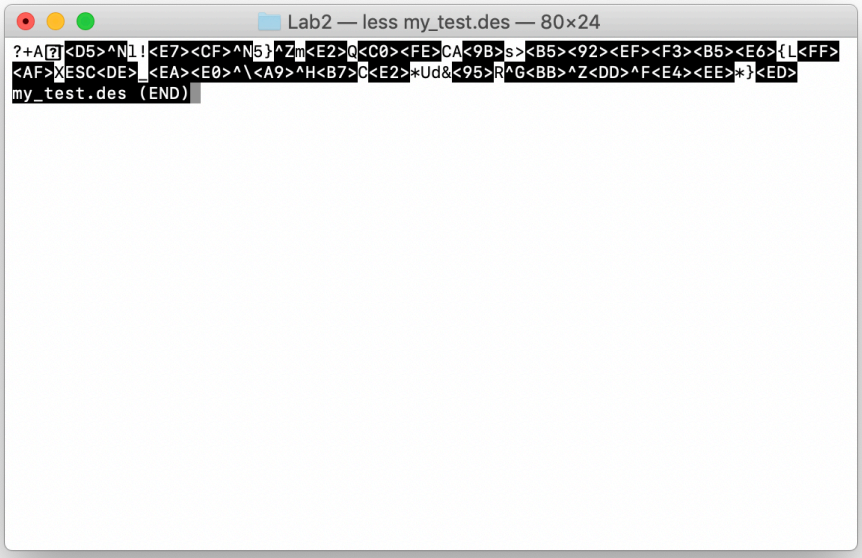
The output will be:



Then, to see the file my_test.des, run the following command line:

less my_test.des

The output will be:



Therefore, we can see that these 2 files are the same , we can conclude that the built-in function des_cbc_encrypt and my DES-CBC encrypt function are similar .

Problem 2: Benchmark DES, RSA and SHA-1

(*) DES encrytion/decryption

The implement code is In the file 2019.07.15.s1242006.tempdes.c

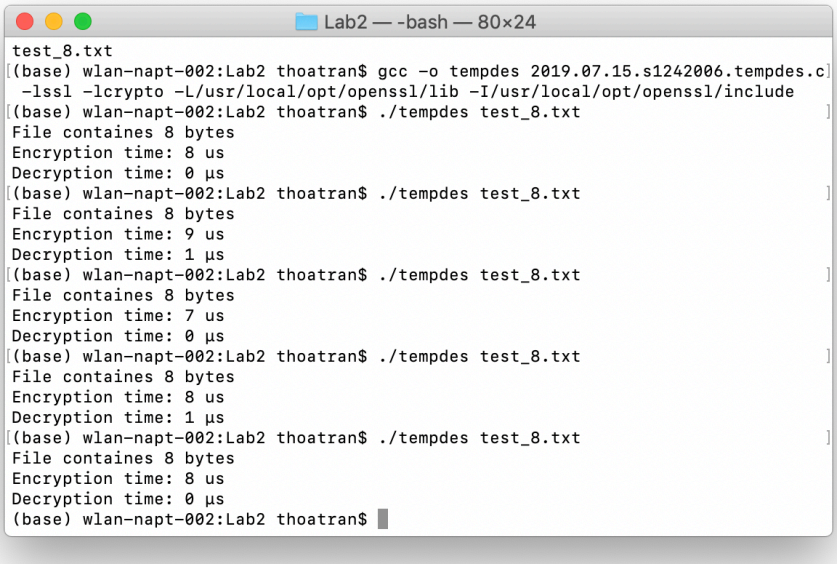
To compile and run the file, run the following command lines:

gcc -o tempdes 2019.07.15.s1242006.tempdes.c -lssl -lcrypto -L/usr/local/opt/openssl/lib -I/usr/local/opt/openssl/include

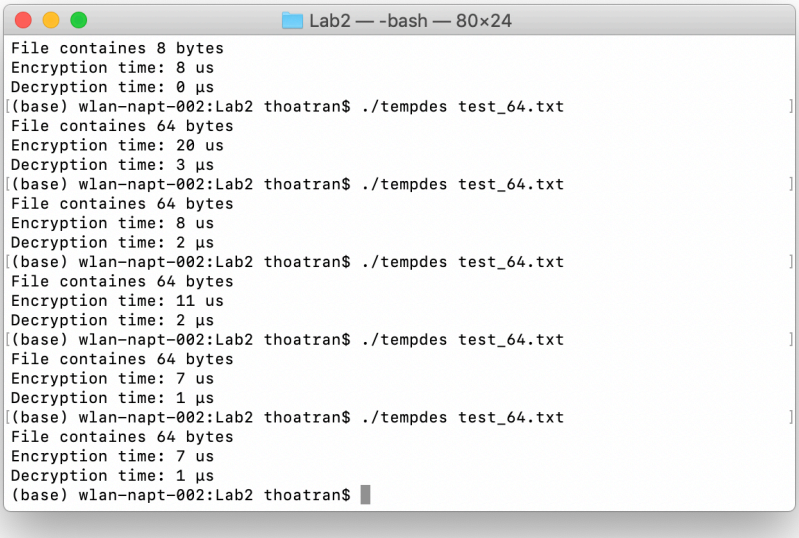
./tempdes [input_file]

With the input file here is test_8.txt, test_64.txt, test_512.txt,… ,respectively.

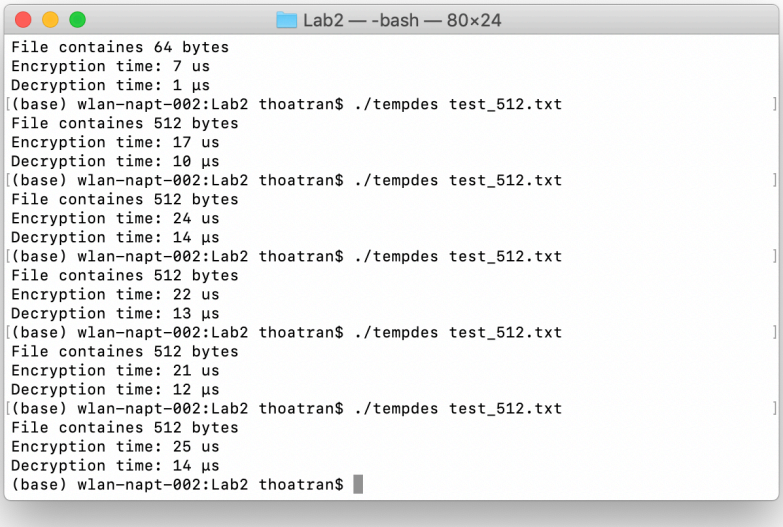
+) For the test_8.txt, we have the output:



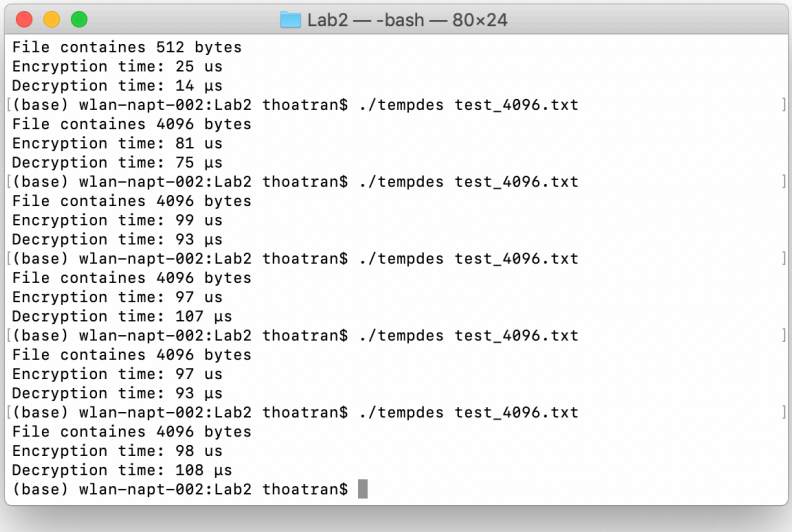
+) For the file test_64, we have the output:



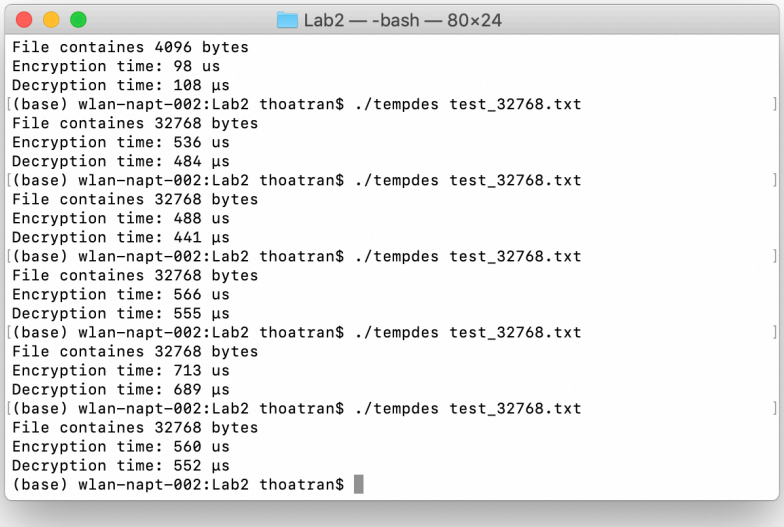
+) For the file test_512.txt, we have the output:



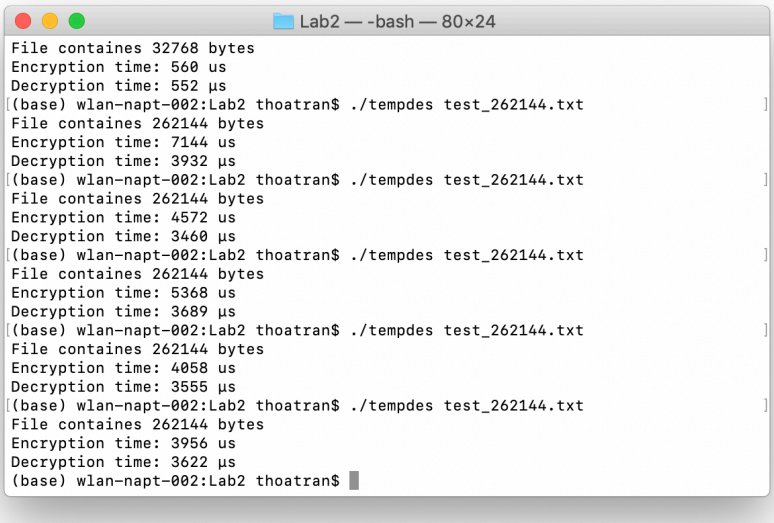
+) For the file test_4096.txt, the output will be:



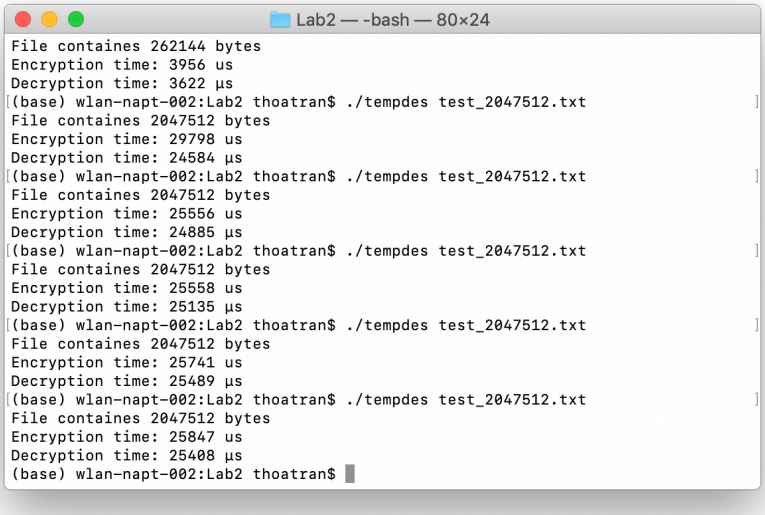
+) For the file test_32768.txt, the output will be:



+) For the file test_262144.txt, the output will be:



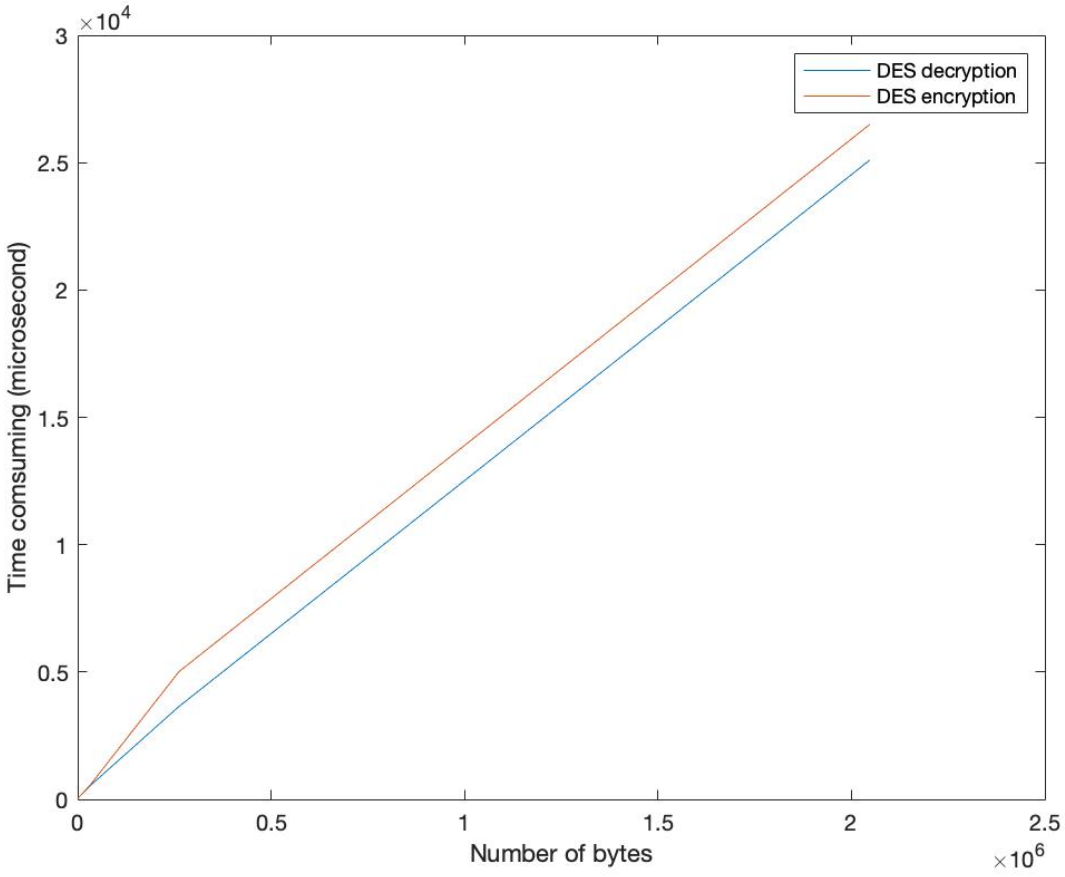
+) For the file test_2047512.txt, the output will be:



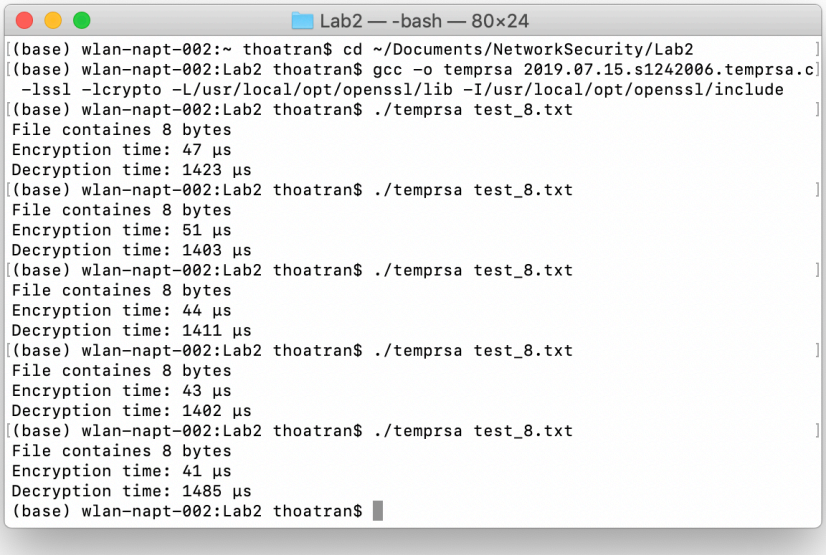
From the above results,we get the average time consuming of DES encryption/decryption (μs)

Number of bytes	8	64	512	4096	32768	262144	2047512
DES encryption.	8	10.6	21.8	94.4	572.6	5019.6	26500
DES decryption.	0.4	1.8	12.6	95.2	544.2	3651.6	25100.2

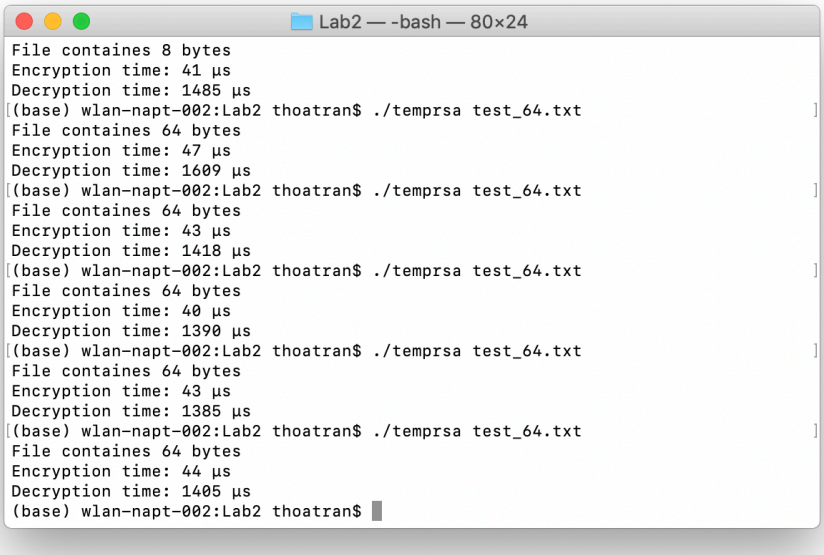
Therefore, we get the graph showing the relation between number of bytes and the time consuming for DES encryption/decryption:



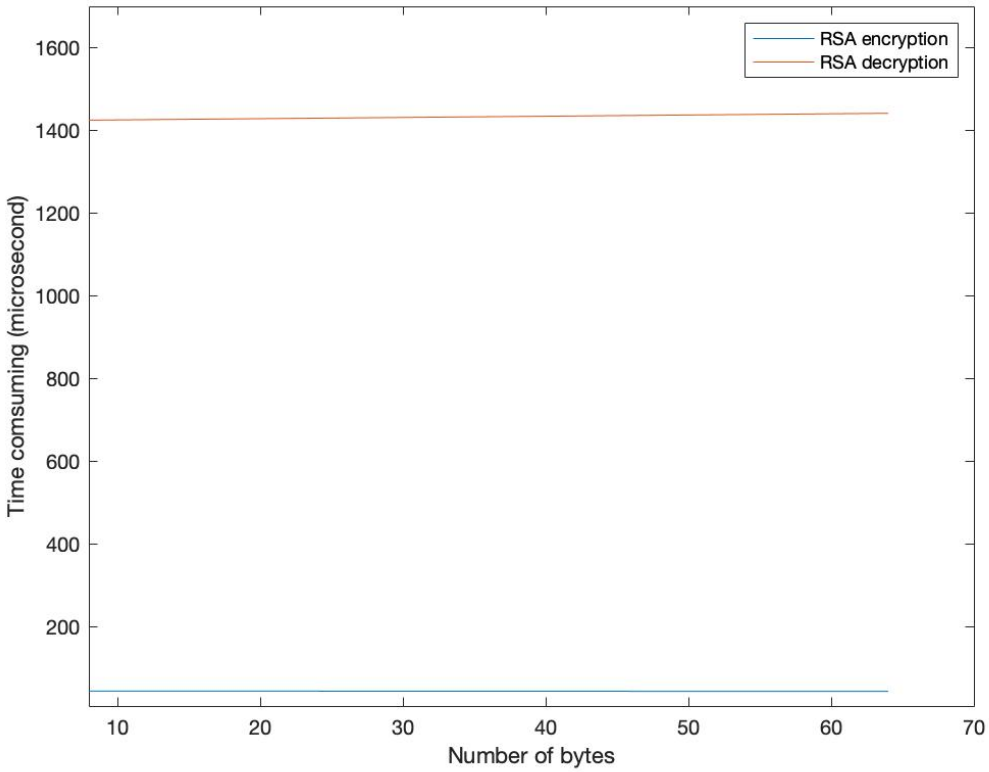
(*) RSA encryption/decryption
The implement code is In the file 2019.07.15.s1242006.temprsa.c
To compile and run the file, run the following command lines:
gcc -o tempras 2019.07.15.s1242006.temprsa.c -lssl -lcrypto -L/usr/local/opt/openssl/lib -I/usr/local/opt/openssl/include
./temprsa [input_file]
With the input file here is test_8.txt, test_64.txt, test_512.txt,… ,respectively.
+) For the file test_8.txt



+) For the file test_64.txt



Because in the implementation we use the RSA_PKCS1_PADDING mode, it just can encrypt the file having the number of bytes smaller than the number of key minus 11 bytes, therefore, it just can encrypt the file test_8.txt and file test_64.txt



We have the graph showing the relation between the number of bytes and the time consuming(μs) of RSA encryption/decryption:

(*) SHA1 encryption:

The implement code is in the file 2019.07.15.s1242006.tempsha1.c

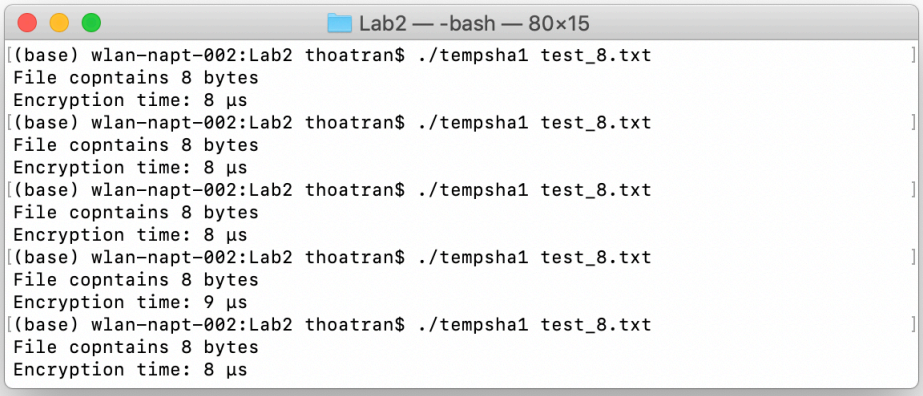
To compile and run the file, run the following command lines:

gcc -o tempsha1 2019.07.15.s1242006.tempsha1.c -lssl -lcrypto -L/usr/local/opt/openssl/lib -I/usr/local/opt/openssl/include

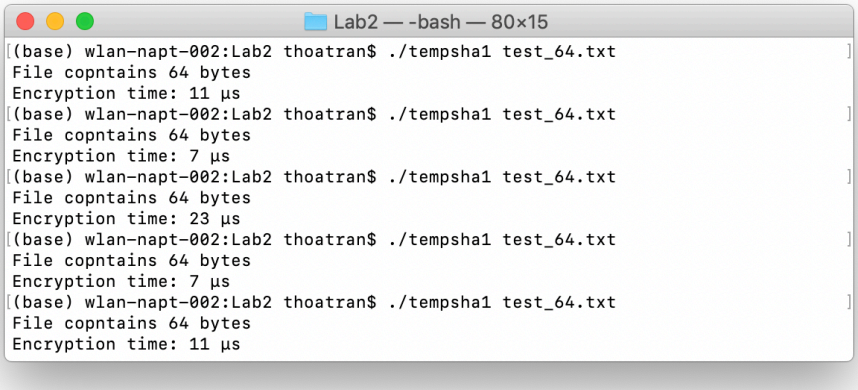
./tempdes [input_file]

With the input file here is test_8.txt, test_64.txt, test_512.txt,… ,respectively.

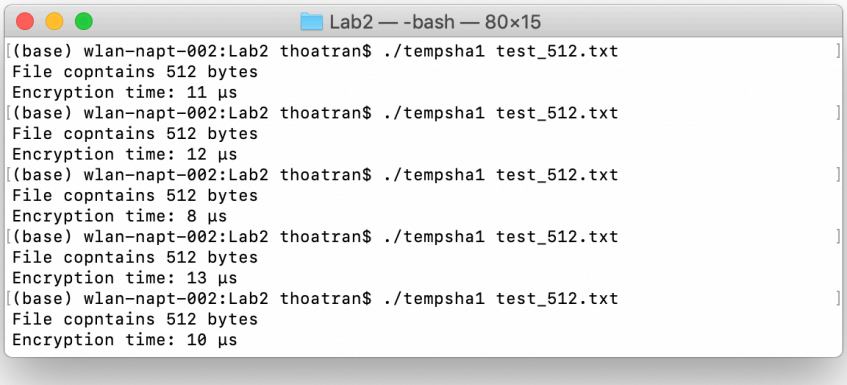
+) For the file test_8.txt



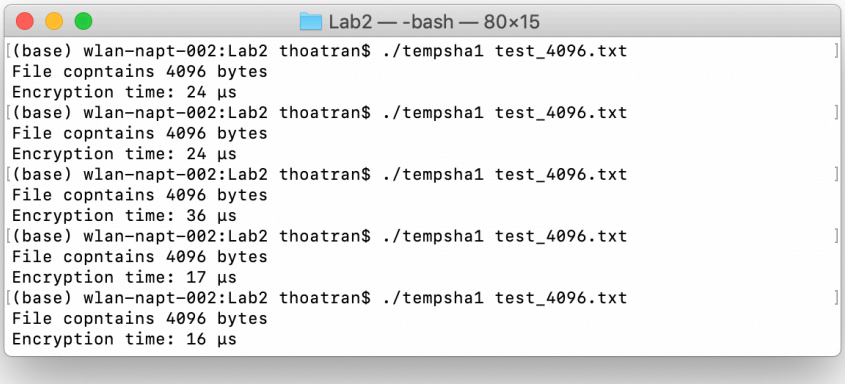
+) For the file test_64.txt



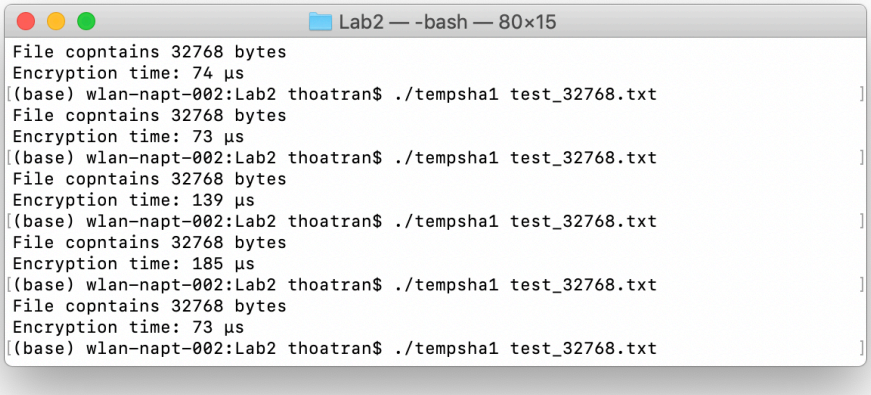
+)For the file test_512.txt



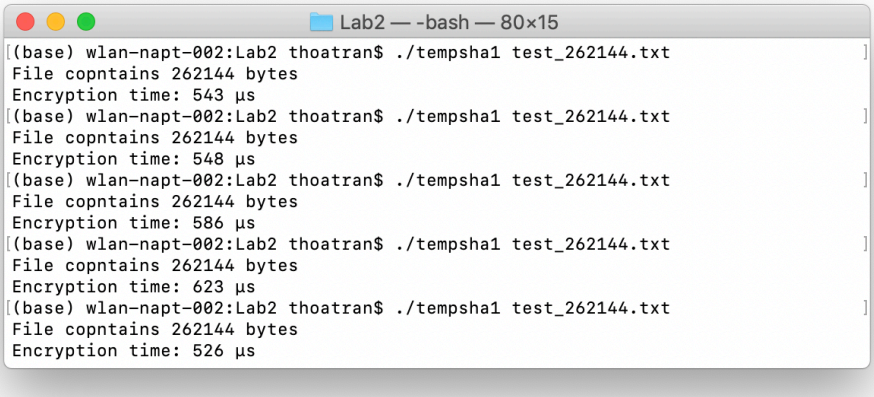
+)For the file test_4096.txt



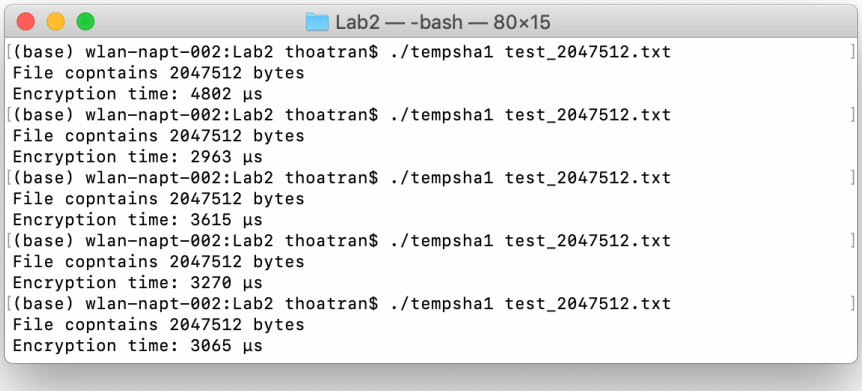
+) For the file test_32768.txt



+) For the file test_262144.txt



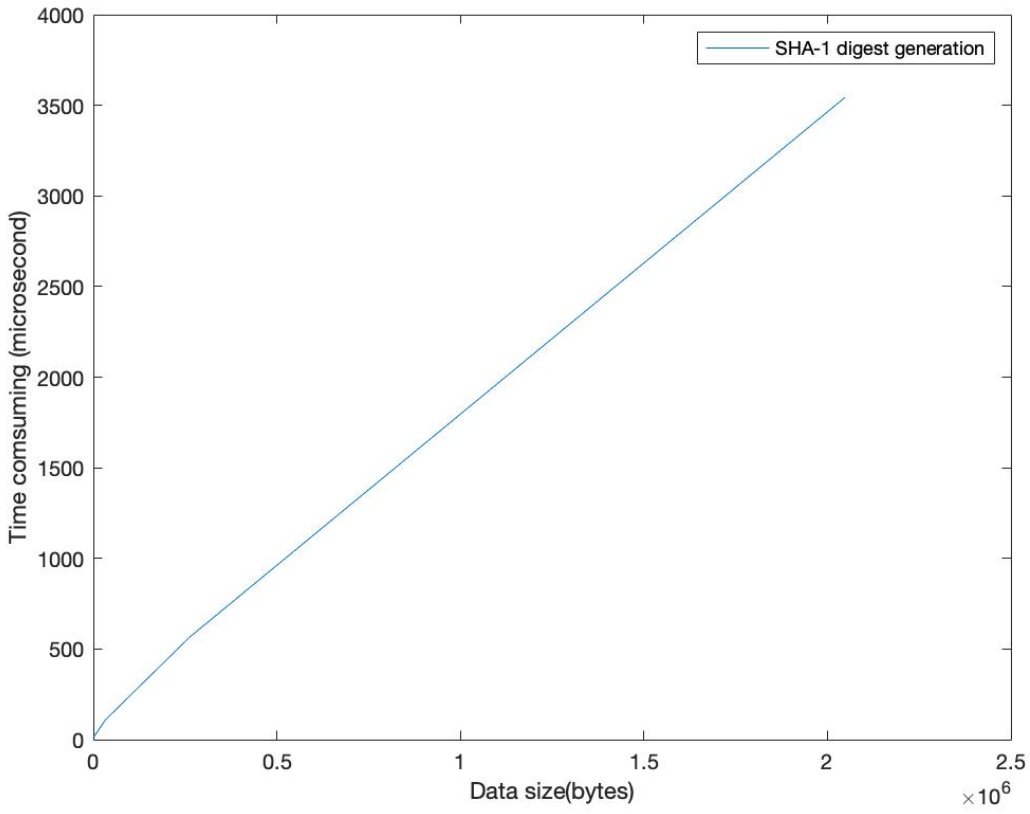
+)For the file test_2047512.txt



We have the average time consuming of SHA-1 digest generation:

Data size (bytes)	8.	64	512	4096	32768	262144	2047512
SHA1 digest generation.	8.	10.8.	11.8.	23.4	108.8	565.2	3543

Therefore, we have the gray showing the SHA-1 digest generation time when increasing the data size:



From the above graph and figure, we can see that:
+) The RSA encryption time is significantly much bigger than DES encryption time or in other way, DES is much faster than RSA encryption. The reason is while DES encryption is symmetric key encryption, RSA is asymmetric key encryption, involveing doing computation with very large numbers, in particular deciphering is computing a large number to a huge power.

+) The DES encryption time consuming and SHA-1 digest generation time consuming increase when increasing the number of bytes in data, but we can see that the SHA-1 digest generation time increasing speed is much slower compared to DES encryption's one. That's because SHA-1 (and many other hashes designed to be cryptographically strong) are based on repeated application of an encryption or decryption routine to fixed-sized blocks of data. Consequently, when computing a hash value of a long string, the algorithm takes proportionally more time than computing the hash value of a small string. Mathematically, we say that the

runtime to hash a string of length N is O(N) when using SHA-1, whereas DES works with the plaintext with 64 bit key. The DES imposes 16 complex rounds to encrypt the data. The 16 rounds are iterated and the ciphers are same but it uses a different key derived, therefore when increasing the data size, the encryption time consuming increases promptly.

+) The RSA decryption time is significantly higher then the RSA encryption time or in other way, the RSA deception is much slower than RSA encryption, that's because both RSA encryption and decryption involve modular exponentiation, but whereas the public encryption exponent is normally small and fixed, the secret decryption exponent is usually almost as long as the modulus. Thus, doubling the modulus size makes encryption take twice as long, but makes decryption take *four times* as long