COMP9337 - Securing Fixed and Wireless Networks

T1 2022

LAB 1

Group: T18B 7

Name: Yuhua Zhao – **ZID:** z5404443

Name: Yasin Khan – **ZID:** z5265047

Environment:

- **Programming Language:** Python
- **Version:** 3.9.2

Part A: DES Encryption and Decryption:

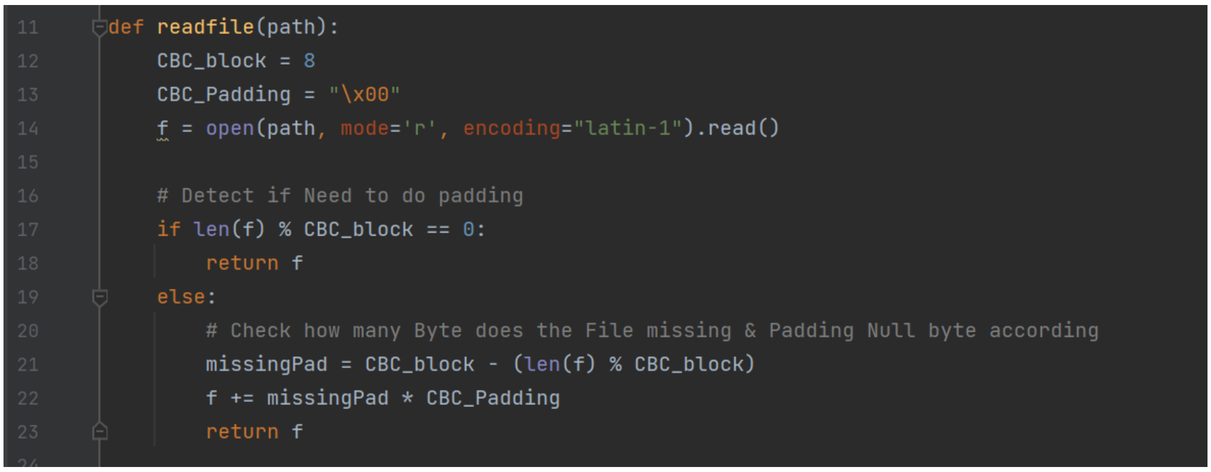
- **Please run the following argument to test the Outcome**

```
python tempdes.py fecdba9876543210 0123456789abcdef test.txt mytest.des
```

Argument Details:

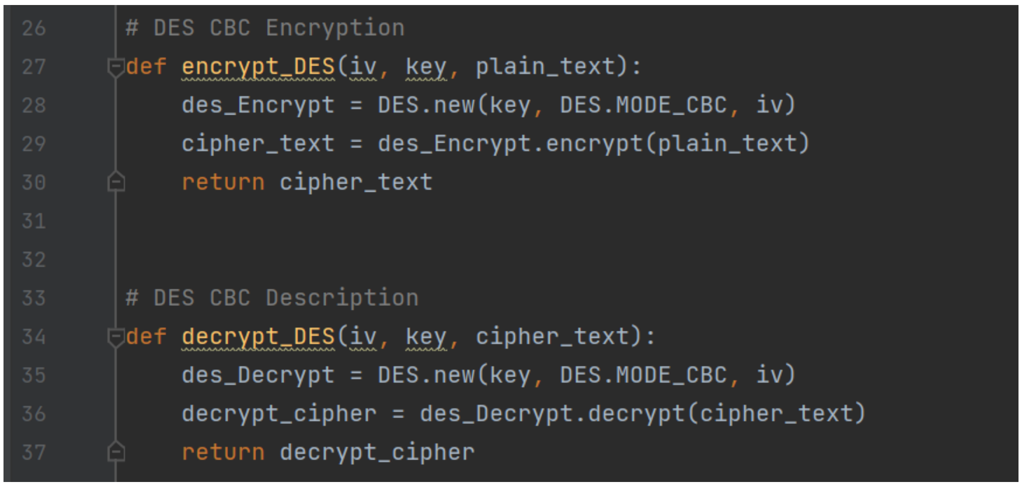
- Argument 1: Initialization Vector (IV)
- Argument 2: Key to be use for encryption
- Argument 3: Input file path of the file that will be read
- Argument 4: Encrypted file will be generated based on the argument 4 name.

Part A: Code Explanation:



```
11  def readfile(path):
12      CBC_block = 8
13      CBC_Padding = "\x00"
14      f = open(path, mode='r', encoding="latin-1").read()
15  
16      # Detect if Need to do padding
17      if len(f) % CBC_block == 0:
18          return f
19      else:
20          # Check how many Byte does the File missing & Padding Null byte according
21          missingPad = CBC_block - (len(f) % CBC_block)
22          f += missingPad * CBC_Padding
23          return f
```

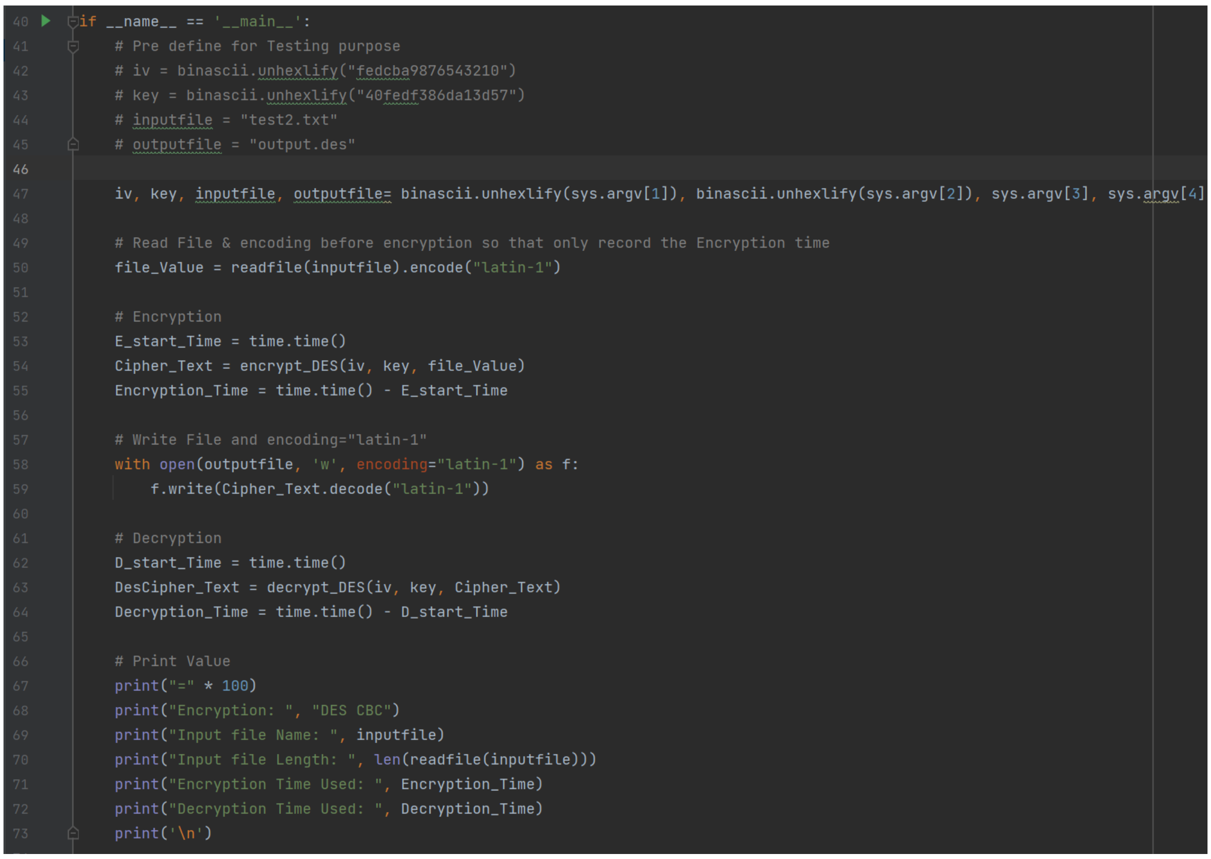
- DES encryption input should be multiple of 8 bytes. If this is not the case, then we will pad it to satisfy the requirement.
- Firstly, line 14 will read the file according to argument 3.
- Secondly, the program will check whether the length of the input is multiple of 8 bytes. If true, will return the input itself unmodified. Else it will detect how many missing bytes are and add the Padding accordingly.



```python
# DES CBC Encryption
def encrypt_DES(iv, key, plain_text):
    des_Encrypt = DES.new(key, DES.MODE_CBC, iv)
    cipher_text = des_Encrypt.encrypt(plain_text)
    return cipher_text


# DES CBC Description
def decrypt_DES(iv, key, cipher_text):
    des_Decrypt = DES.new(key, DES.MODE_CBC, iv)
    decrypt_cipher = des_Decrypt.decrypt(cipher_text)
    return decrypt_cipher
```

- “Encrypt_DES” function is used to encrypt the plain text
- “Decrypt_DES” Function is used to decrypt the Cipher test that encrypt by the Encrypt_DES function.



```python
40  if __name__ == '__main__':
41      # Pre define for Testing purpose
42      # iv = binascii.unhexlify("fedcba9876543210")
43      # key = binascii.unhexlify("40fedf386da13d57")
44      # inputfile = "test2.txt"
45      # outputfile = "output.des"
46
47      iv, key, inputfile, outputfile= binascii.unhexlify(sys.argv[1]), binascii.unhexlify(sys.argv[2]), sys.argv[3], sys.argv[4]
48
49      # Read File & encoding before encryption so that only record the Encryption time
50      file_Value = readfile(inputfile).encode("latin-1")
51
52      # Encryption
53      E_start_Time = time.time()
54      Cipher_Text = encrypt_DES(iv, key, file_Value)
55      Encryption_Time = time.time() - E_start_Time
56
57      # Write File and encoding="latin-1"
58      with open(outputfile, 'w', encoding="latin-1") as f:
59          f.write(Cipher_Text.decode("latin-1"))
60
61      # Decryption
62      D_start_Time = time.time()
63      DesCipher_Text = decrypt_DES(iv, key, Cipher_Text)
64      Decryption_Time = time.time() - D_start_Time
65
66      # Print Value
67      print("=" * 100)
68      print("Encryption: ", "DES CBC")
69      print("Input file Name: ", inputfile)
70      print("Input file Length: ", len(readfile(inputfile)))
71      print("Encryption Time Used: ", Encryption_Time)
72      print("Decryption Time Used: ", Decryption_Time)
73      print('\n')
```

- Line 47: Detect arguments when this python script is being called and assigned to the Variable respectively.
- Line 50: Call the Read File to get the text that needs to be encrypted and encode it with the selected Format.
- Line 53 – 55: Record the time used for DES CBC encryption.
- Line 57 – 59: Generate a file named by Argument 4 and write Ciphertext in.
- Line 62 – 64: Record the time used for DES CBC decryption.
- Line 67 – 73: Print the necessary value for Lab Part B.

Part B: Performance Measures for various algorithms:

File and directory structure:

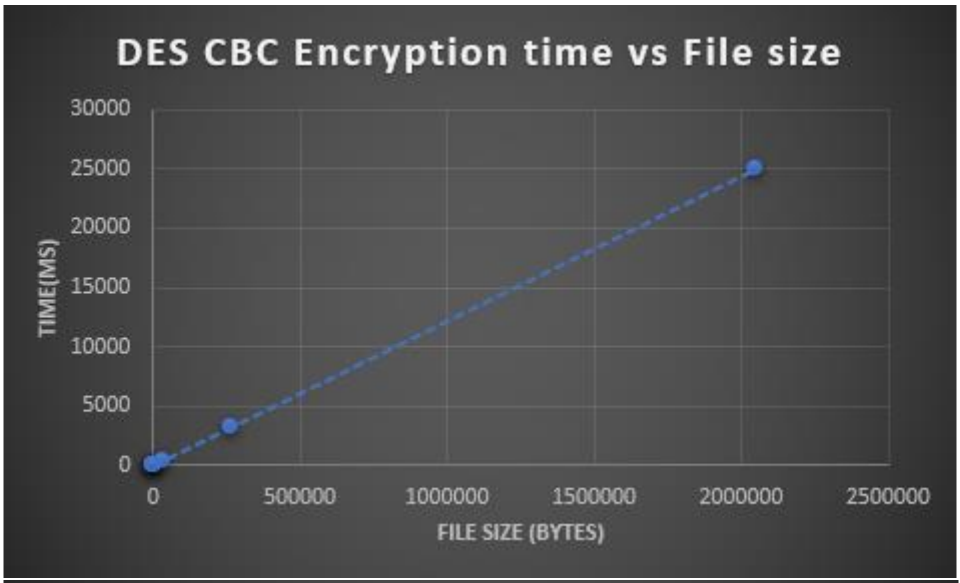
File/Directory name:	Description:
TestFile	Includes all the Plaintext files with different sizes that are used to examine performance of the various algorithms
tempdes.py	DES CBC Algorithm
tempaes.py	AES CBC Algorithm
temprsa.py	RSA Algorithm
tempsha1.py	SHA-1 Algorithm
tempHMAC.py	IHMAC Signature Algorithm
algPerformance.py	Simple script that runs the above algorithms on the plaintext files, and prints out summary statistics based on timing performance

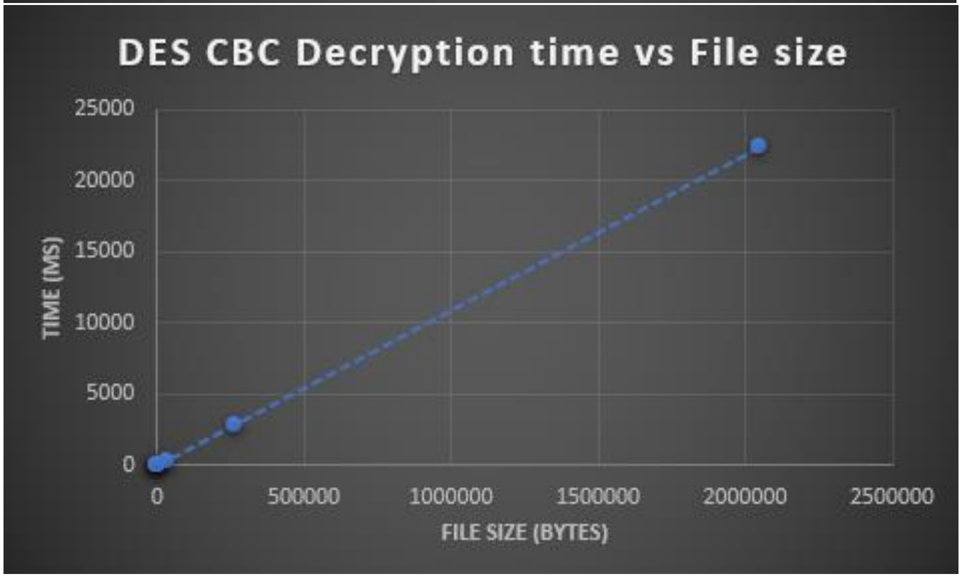
Code Testing:

Please run the following command to test the performance

```
python algPerformance.py
```

The command above will run all the encryption algorithms that have specified on the requirement and print out the Encryption time and Decryption according to the Algorithm.

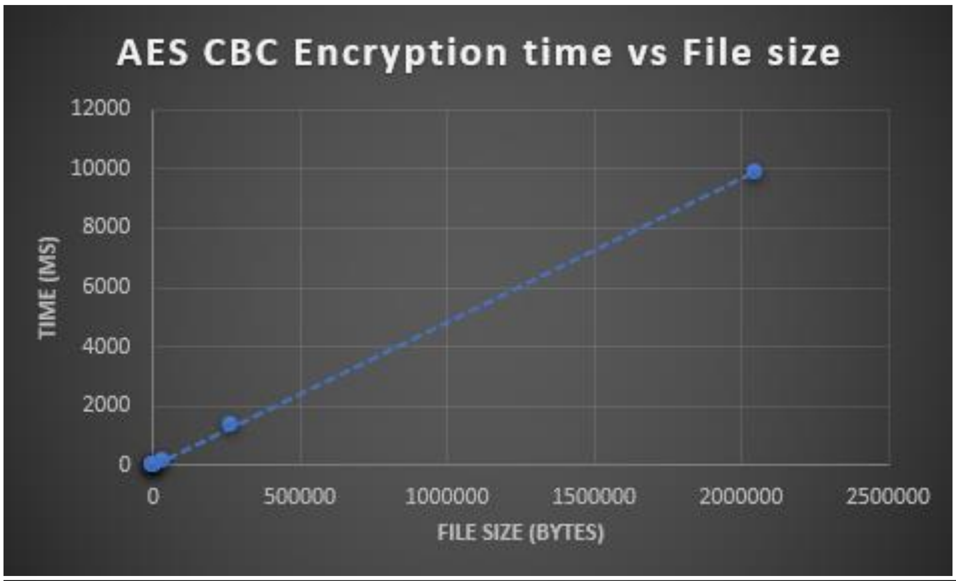


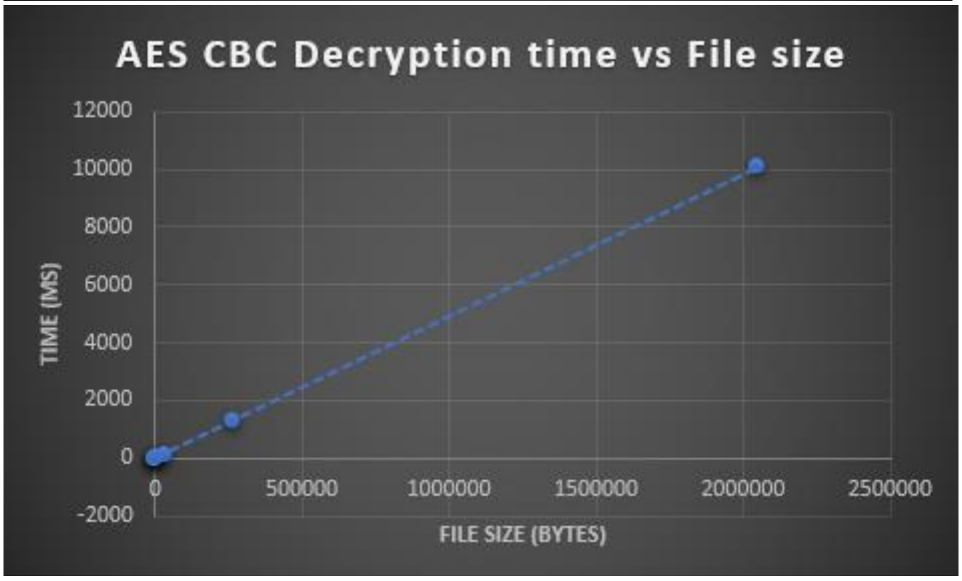


File Size (bytes)	**Encryption time (µs)**	**Decryption time (µs)**
16	23	17
64	24	19
512	38	23
4096	81	96
32768	470	362
262144	3315	2909
2047152	25007	22396

Observation notes:

- Both Encryption and Decryption times seem to have a positive trend
- Decryption seems to be slightly faster at times, but times are generally comparable
- Encryption and decryption appear much faster than RSA but slower than AES as well as the HASH algorithms

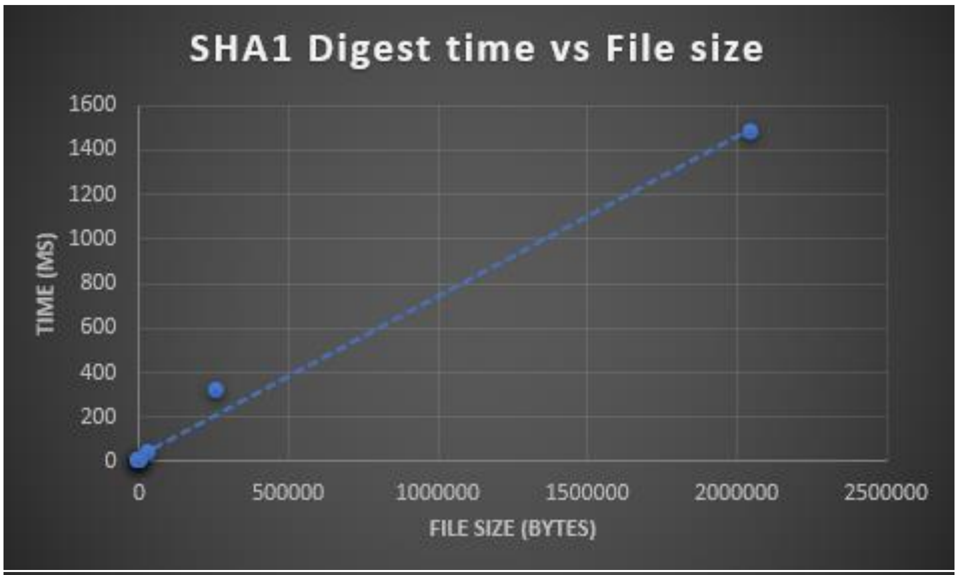


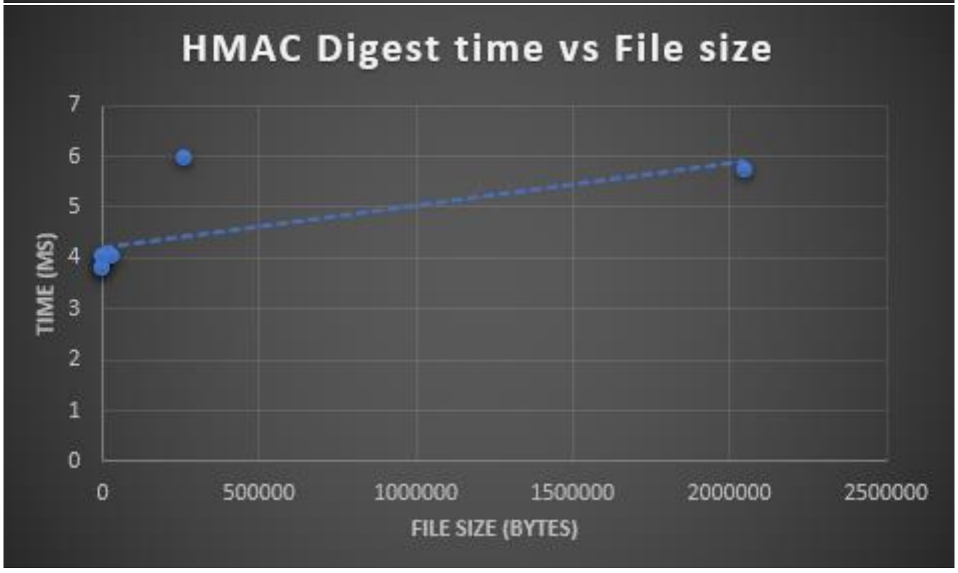


File Size (bytes)	Encryption time (µs)	Decryption time (µs)
16	7	4
64	7	5
512	10	7
4096	26	24
32768	153	154
262144	1386	1273
2047152	9900	10106

Observation notes:

- Both Encryption and Decryption times seem to have a positive trend
- Decryption time is slightly faster than encryption for smaller file sizes and longer for larger
- Encryption and Decryption times faster than RSA, DES but slower than HASH algorithms

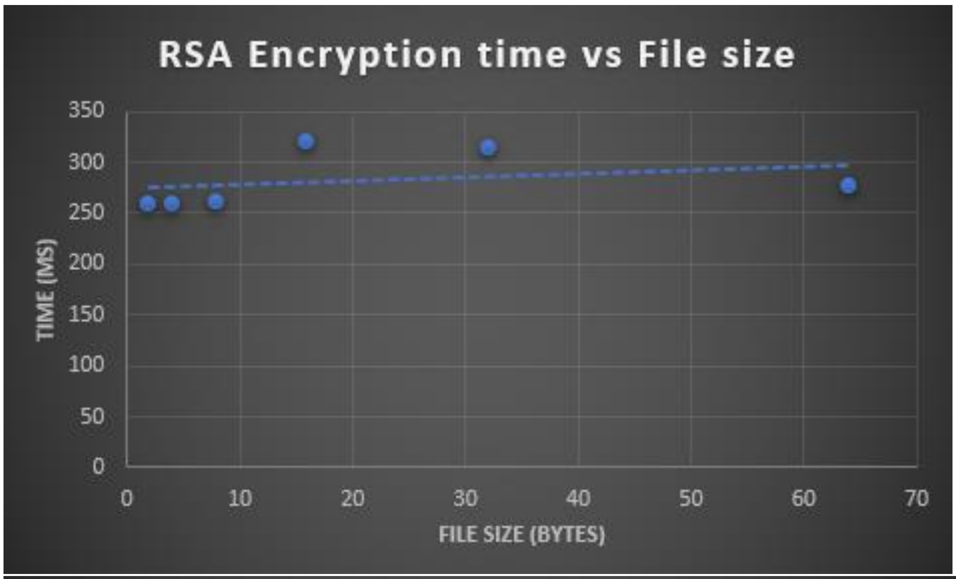


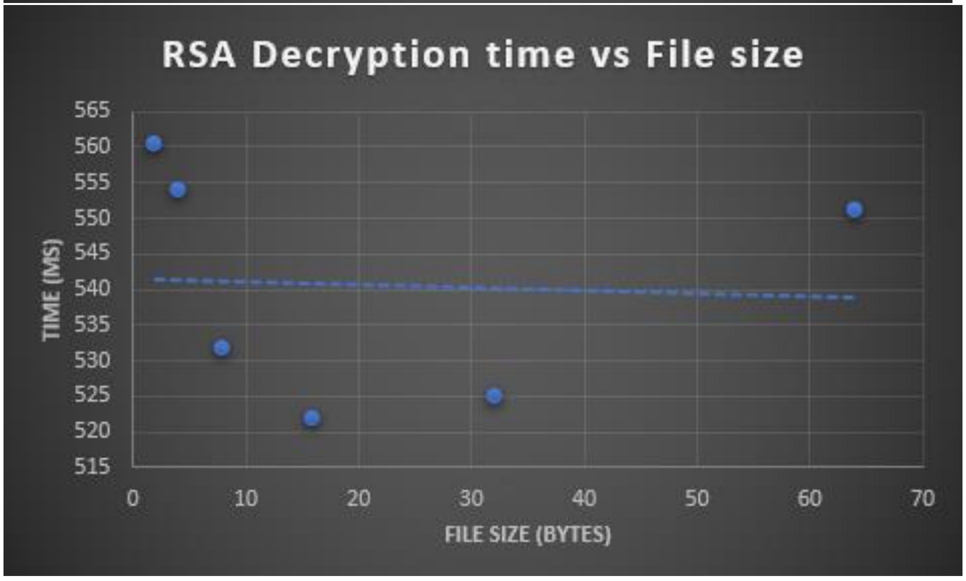


File Size (bytes)	SHA1 Digestion time (µs)	HMAC Digestion time (µs)
16	2	4
64	5	4
512	3	4
4096	8	4
32768	41	4
262144	323	6
2047152	1484	6

Observation notes:

- SHA1 and HMAC both have a generally positive trend with time vs file size
- It's worth noting that SHA1 shapes better to a linear trend where has HMAC retains similar time for all byte lengths.
- This means HMAC maintains a fast time even for very large file sizes compared to SHA1
- HMAC appears to be the fastest algorithm of them all followed by SHA-1. Overall HASH algorithms outperformed counterparts





File Size (bytes)	Encryption time (µs)	Decryption time (µs)
2	259	561
4	260	554
8	261	532
16	320	522
32	314	525
64	276	551

Observation notes:

- There doesn't seem to be a trend for RSA encryption
- However one thing that is interesting is that the times are all in a similar range for all the file sizes despite some being much larger than others
- For RSA decryption there doesn't seem to be any pattern, times look random for file sizes however its clear that decryption takes longer around twice the time of encryption
- RSA looks like the slowest, having a comparable time to others despite having only being tested with small file samples capping at 64 bytes.