Contents

1 Measuring Time

For measuring time I used the following time function

```
import time

start_time = time.time()
    #tasks for measurement
print("--- %s seconds ---" % (time.time() - start_time))
```

2 Generating Input Data File

I need six different files of size 25MB, 50MB, 75MB, 100MB, 500MB and 1000MB. I decided to generate some text files named ***MB25.txt***, ***MB50.txt***, ***MB75.txt***, ***MB100.txt***, ***MB500.txt*** and ***MB1000.txt*** with random characters by using the following code

```
import random,string

f = open('MB25.txt','w+')
key = ''.join(random.choices(string.ascii_uppercase + string.digits,
            k = (25*1024*1024)))
f.write(key)
f.close()

f = open('MB50.txt','w+')
key = ''.join(random.choices(string.ascii_uppercase + string.digits,
            k = (50*1024*1024)))
f.write(key)
f.close()

f = open('MB75.txt','w+')
key = ''.join(random.choices(string.ascii_uppercase + string.digits,
            k = (75*1024*1024)))
f.write(key)
f.close()


```

```
22 f = open('MB100.txt','w+')
23 key = ''.join(random.choices(string.ascii_uppercase + string.digits,
24             k = (100*1024*1024)))
25 f.write(key)
26 f.close()
27
28 f = open('MB500.txt','w+')
29 key = ''.join(random.choices(string.ascii_uppercase + string.digits,
30             k = (500*1024*1024)))
31 f.write(key)
32 f.close()
33
34 f = open('MB1000.txt','w+')
35 key = ''.join(random.choices(string.ascii_uppercase + string.digits,
36             k = (1000*1024*1024)))
37 f.write(key)
38 f.close()
```

3 Data Encryption Standard (DES)

3.1 Encryption Parameter Details

For measuring the performance of DES, I used DES where the mode of DES was ***Electronic CodeBook***(ECB)

3.2 Code

```
import time
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes

key = get_random_bytes(8)
print(key)
des = DES.new(key,DES.MODE_ECB)

def pad(text):
    while len(text) % 8 != 0:
        text += b' '
    return text

def dataEncryptionStandard(text,des):
    padded_text = pad(text.encode())
    start_time = time.time()
    encrypted_text = des.encrypt(padded_text)
    print("---Encryptin Time: %s seconds ---"
        % round((time.time() - start_time),2))
    start_time = time.time()
    d_text = des.decrypt(encrypted_text)
    print("---Decryption Time: %s seconds ---"
        % round((time.time() - start_time),2))

ls = ['MB100.txt','MB500.txt','MB1000.txt']
for i in ls:
    f=open(i, 'r')
    text = f.read()
    dataEncryptionStandard(text,des)
    f.close()
```

3.3 Used Key

After executing the code I got the key as follow

```
\xd8\xd9\x05D'\x90\xf21
```

3.4 Comparison of Execution times

Here is the comparison of different execution times

	Encryption Time	**Decryption Time**
100 MB	1.67 sec	1.45 sec
500 MB	7.47 sec	7.36 sec
1000 MB	14.73 sec	15.01 sec

4 Rivest–Shamir–Adleman (RSA)

4.1 Encryption and Decryption Procedure

The RSA cannot encrypt more data than Key-Size. So, instead of encrypting the whole file, I divided the file into blocks of **100 characters** and then stored them in encrypted file line-wise, then to decrypt we read the file line-by-line and then decrypt and concatenate the results to get original data.

4.2 Encryption Parameter Details

I have used Key_Length = 2048 bits, Mode = $PKCS1\_v1\_5$, Encoding = base64

4.3 Input data

As RSA algorithm takes huge time to encrypt and decrypt data, I have decided to minimize the file size. So, I have used 25MB, 50MB and 75MB text files for performance measurement.

4.4 Code

```
import hashlib
from Crypto.Cipher import PKCS1_v1_5
from Crypto.PublicKey import RSA
from Crypto.Random import new as Random
from base64 import b64encode
from base64 import b64decode
import random
import time
from os import system

class RSA_Cipher:
    def generate_key(self, key_length):
        assert key_length in [1024, 2048, 4096]
        rng = Random().read
        self.key = RSA.generate(key_length, rng)

    def save_key(self):
        PK = self.key
        with open("RSAPrivateKeyFile", "wb") as RKF:
```

```
            RKF.write(PK.export_key("PEM"))

    def load_key(self):
        with open("RSAPrivateKeyFile", "rb") as PKF:
            self.key = RSA.import_key(PKF.read())

    def encrypt(self, data):
        plaintext = b64encode(data.encode())
        rsa_encryption_cipher = PKCS1_v1_5.new(self.key)
        ciphertext = rsa_encryption_cipher.encrypt(plaintext)
        return b64encode(ciphertext).decode()

    def decrypt(self, data):
        ciphertext = b64decode(data.encode())
        rsa_decryption_cipher = PKCS1_v1_5.new(self.key)
        plaintext = rsa_decryption_cipher.decrypt(ciphertext, 16)
        return b64decode(plaintext).decode()

def encryptD2file():
    EncryptedDataList = []
    RSACipherObject = RSA_Cipher()
    RSACipherObject.load_key()

    with open("MB100.txt", "r") as DF2:
        Fdata = DF2.read()
    s = 0
    e = s + 100
    length = len(Fdata)
    for i in range(length):

        if Fdata[s:e] == "":
            break
        EncryptedDataList.append(RSACipherObject.encrypt(Fdata[s:e]))
        s = s + 100
        e = e + 100
    try:
        system("del EncD2File")
    except Exception:
        print("NF : Ignore")
    with open("EncD2File", "a+") as EF:
```

```
60          for line in EncryptedDataList:
61              EF.write(line + "\n")
62
63  def decryptD2file():
64      RSACipherObject = RSA_Cipher()
65      RSACipherObject.load_key()
66      with open("EncD2File", "r") as EF:
67          ETs = EF.readlines()
68      try:
69          system("del DecD2File")
70      except Exception:
71          print("NF: Ignore")
72      with open("DecD2File", "a+") as DF:
73          t = 0
74          for i in ETs:
75              DF.write(RSACipherObject.decrypt(i))
76              t += 1
77
78  if __name__ == "__main__":
79      RSACipherObject = RSA_Cipher()
80
81      print("Generating Key | ", end="")
82      Mark = time.time()
83      RSACipherObject.generate_key(2048)
84      RSACipherObject.save_key()
85      print("Key Generation Runtime = " + str(time.time() - Mark))
86
87      print("Encrypting File | ", end="")
88      Mark = time.time()
89      encryptD2file()
90      print("Encryption Runtime = " + str(time.time() - Mark))
91
92      print("Decrypting File | ", end="")
93      Mark = time.time()
94      decryptD2file()
95      print("Decryption Runtime = " + str(time.time() - Mark))
```

4.5 Used Keys

After executing the key generation code, i have got the following keys

—BEGIN RSA PRIVATE KEY—
\nMIIEogIBAAKCAQEAxFRNssKIeG7zbgomQfEROsqOdxjj51KxomU5l
VUfmXtBkNXc\naChB0oYtOJN47aUyckPB75mALNqxSmCZecOycx0yX
/7DAEgJPwspHHwft7wy/fcX\n5CKR7oZupTG3Y7A2tEfTzUh0R9nW0A
Mw9RBKM5yfpMBpX5sr+VgWJ3Q1BN2janas\nLBbd9Q6B8B0kS3P8W
ZB994l1JDbr5GGDADQ4wawwe9xbTQr0wAvzxsVitSrrqfx1\njKl6n4jK3P
LvoYdovJah+8gmPWUXhwXrK8Oe0pyeLkAQNKpztUIeX2/CrtqQ2vSx\n
hRycmOklx9Fo1Yil9gk0NsJb1fq17JFgc8SXrwIDAQABAoIBAA6tK1ZwK
BSHXXI9\n7FM59VWa8SB3v126YaYSrBsnjw0BrkqTT36HzTfdfkGVoKY
fQdCnPmGuF2tYToWR\nXW0WqInmG68aaTMQWi9DeHzKeJ0wbUgX8
coaSCTNGYtcK+nrW9jelQzqXCryNBaT\nuSlbYONP041U3EELOrPRQRg
M81kGzlt0o79PKpES52p/mcIKyMQEZJVw+9PNjKH7\nDC5SVAy5PgLo
UVfcm/OPsIQcJvVMNDeRZK68zS3KL2rlFjgbNLbQry5whaMUS4ph\nyNy
Wi8fS0iZGF7pe9F4J7r3PyzOE5+QkqW31bZ3mMB6VzPg4k61IeeVpDp5Z
WcY9\ngo6J8A0CgYEA3AQnv58IwOMdbX16RaBgl9aRJQFIC3DqpprSLM
Mx0ptwSELF816O\ngBm8WmXOiPBsHXU3HVgjsUH1U2yd2WAxoBNfeR
G5AWmex19uDckxtKQY1oMqUOl1\npDEdn52dtpqwnflTdkVRjuUrtKgDK
H38AxhtkVdErdmNWKzPd7TW4xsCgYEA5HBm\n8z39r6xGWZE2Z1bbnP
BSTceSaxikUlcr2slw4QNAlIxT+VX1uGe6yzIgm0jKd012\ntzRU83oxLNDzC
06ByD3q6bulrJ7upZJW5FLzb5epV9ca1UtlpZDBQCTxr6kJuvMo\nfsoke1/ZG
cAW8TPXFfQGicDqjIOBPYjzPRbu0v0CgYAIkGTXsaIOmAiSNXsn8Kkx\n3
9lAfn9GarvHhmk98s8iqohBV9AKvBiB5f6N6j6S2ADtWJ2vl7mz8aRR/f1a88
Ka\nv4ZM2SLBalIS4qUPBlDX/jc1Sl/ATIYWwQ6tIt57uCO90nGsPeCB1mvQ
GWYeEmFX\n4YCKUsJKnhhBZmChAToIFwKBgGkE579xw1vs1la89Ibjnojy
ztOlKUIFDR4xpx6p\nNzSq2xivvrdE9nc1cOGThPv+pd3dBfPJojhg95SD5whO
qkmVdZbIxqyqUEpcEYhc\nPkASOC8C2/os9Gj+OGm10kqQasHAoeBQLB9v
/A3jimiPDZWxHwRbZTj+B/9zI2f1\nUAw9AoGAWFIOlM/qVcphtRP+eTd5m
BJX9/UicQHrbULUnkTCeGzwcwaPLzgDcD5s\ne7bVn/1vOPqt4lJ1/PYL1oa3
5l0ceYZ9V52rfxJ8OlkcY9XdmC7C6zXNviJxKaeu\nYPG8ZB1MWCbcLjT1Lh
ldEqtr88w5cMJlmTIsXJ3ffd/zma9fhXc=\n
—END RSA PRIVATE KEY—

—BEGIN PUBLIC KEY—
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxFRNssKIeG7z
bgomQfER\nOsqOdxjj51KxomU5lVUfmXtBkNXcaChB0oYtOJN47aUyckPB75
mALNqxSmCZecOy\ncx0yX/7DAEgJPwspHHwft7wy/fcX5CKR7oZupTG3Y7A
2tEfTzUh0R9nW0AMw9RBK\nM5yfpMBpX5sr+VgWJ3Q1BN2janasLBbd9Q6B
8B0kS3P8WZB994l1JDbr5GGDADQ4\nwawwe9xbTQr0wAvzxsVitSrrqfx1jKl6n
4jK3PLvoYdovJah+8gmPWUXhwXrK8Oe\n0pyeLkAQNKpztUIeX2/CrtqQ2vSx
hRycmOklx9Fo1Yil9gk0NsJb1fq17JFgc8SX\nrwIDAQAB
—-END PUBLIC KEY—

4.6 Comparison of Execution times

I have found the following result

	Encryption Time	Decryption Time
25 MB	166.07 sec	469.53 sec
50 MB	329.02 sec	937.45 sec
75 MB	489.96 sec	1410.77 sec

5 Secure Hash Algorithm-512 (SHA-512)

5.1 Code

I used the following code to generate hash using SHA-512

```
import time
from Crypto.Hash import SHA512

h = SHA512.new()

ls = ['MB100.txt','MB500.txt','MB1000.txt']

for i in ls:
    f=open(i, 'r')
    text = f.read()
    start_time = time.time()
    h.update(text.encode())
    print(h.digest())
    print("--- %s seconds ---" % round((time.time() - start_time),2))
    f.close()
```

5.2 Generated Hash

After executing the code I got the hash as follow

	Cryptographic Hash
100 MB	\xe17\xe0\xc7\xc2\xd0\xea\xe1}\x95&tv\xf5\xf6\x84\xd2\x90 \xbefgT \xcc!\xbc\x0e\xae:\xd4\xae\xe8\x9eV\x9e\x12#m\x89 \xe8\xff\xdf\xd3\xb9lM\xc9\xd7.\x16\x01\xa7\xe4\x862;WP \xaa\x077J\x9d\xd8
500 MB	\xd7(\x95\x98\x9f\xfacP\xa8\x03(\xe5\xf4P\x8fay\x0e\xd4\xe4 \xd0\xf5\x04\xd7\xf5\xfc\xf1\xbe\x8e\x83\x1cA\xe7\x06E\x1b <\xdf\x99\x83H\xd8\xba\n\xf5\xab\xd0\xc0Gfq\x01\xcf0\xab \x9c\x81S\xf57\x99Ad\x11
1000 MB	\x89C\xe6‘x—\x1a\xe6>\x08_\xd3\x15\x18g\xff\xf7X\xb5\x02 \x8as\xa4\x80:tN\xdb*\xfdi\x98\x12{\xc3\x1dtw\xfeE\xf5\xdfk \xc9zj\x94\xb6T\x8b\x1c\xf3\x1fOJI\xa3s\xd2\xab\x02\x9eD

5.3 Comparison of Execution times

Here is the comparison of different execution times

	Time Required
100MB	0.41 sec
500MB	2.0 sec
1000MB	4.08 sec