**CSE 565 Fall 2019**
**Homework 2**
**Computer Security**

**Swati Sajee Kumar**
**#50289560**

## (a) Create a 128-bit AES key, encrypt and decrypt file using AES in the CBC mode.

### Source Code:

```
# Code to create AES- 128 bit key and to encrypt and decrypt a text.txt file using AES in CBC mode

import pyaes,  os
import time

# Genrating the random key of size 128 bit that is 16 bytes, everytime a new key will be generated when u run this part of the code
key_start= time.time()
key=os.urandom(16)
key_end=time.time()

# to check the total time for key generation
key_gen_time = (key_end - key_start)*1000000

#print('Genrated 128 bit key for CBC mode : ' + str(key) )
print( 'Key generation time :  ' + str(key_gen_time) +' microseconds' )

print('--------------------------------------------------------------------------------')


# Encrypt the 1KB text file stored in the variable plaintext with the given key produced in first step:

#create a initial vector iv of 16bytes ,ALso remember CBC takes in one block size
iv = os.urandom(16)

#read the text file
with open('1kb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('1kb.txt')
  f_Size= statinfo.st_size


#perform the AES-256-CTR-Encryption using the text, key and iv. Also the text file must be 16bytes
ency_start = time.time()
encrypter = pyaes.Encrypter(pyaes.AESModeOfOperationCBC(key, iv))

ciphertext = []
for line in open('1kb.txt'):
  ciphertext += encrypter.feed(line)
  #Make a final call to flush any remaining bytes and add padding
ciphertext +=encrypter.feed()
ency_stop = time.time()

#to check total time for encryption
encr_time = (ency_stop - ency_start)*1000000

#write the cipher text into a file
file = open('cipher.txt','w')
file.write(str(ciphertext))


print ('Total time for encryption for 1KB file :' + str(encr_time) +' microseconds' )
total_speed = f_Size/ encr_time
print('Total speed for encryption for 1KB file  : ' + str(total_speed) + '  byte/microseconds')

# Decrypt the ciphertext with the same key for 1 KB
```

```
decry_start = time.time()
decrypter = pyaes.Decrypter(pyaes.AESModeOfOperationCBC(key, iv))

for line in open('cipher.txt'):
  decrypted = decrypter.feed(line)
decry_stop =time.time()

#to check total time for decryption
decry_time = (decry_stop - decry_start)*1000000

#print('Decrypted:', decrypted)
print ('Total time for decryption for 1KB file  :' + str(decry_time) + ' microseconds')
total_speed_1 = f_Size/ decry_time
print('Total speed for decryption for 1KB file  : ' + str(total_speed_1) + '  byte/microsecond')

print('--------------------------------------------------------------------------------')

# Encrypt the 10 MB text file stored in the variable plaintext with the given key produced in first step:

#create a initial vector iv of 16bytes ,ALso remember CBC takes in one block size
iv = os.urandom(16)

#read the text file
with open('10mb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('1kb.txt')
  f_Size= statinfo.st_size

#perform the AES-256-CTR-Encryption using the text, key and iv. Also the text file must be 16bytes
ency_start = time.time()
encrypter = pyaes.Encrypter(pyaes.AESModeOfOperationCBC(key, iv))

ciphertext = []
for line in open('10mb.txt'):
  ciphertext += encrypter.feed(line)
  #Make a final call to flush any remaining bytes and add padding
ciphertext +=encrypter.feed()
ency_stop = time.time()

#to check total time for encryption
encr_time = (ency_stop - ency_start)*1000000

#write the cipher text into a file
file = open('cipher_2.txt','w')
file.write(str(ciphertext))


print ('Total time for encryption for 10MB file  :' + str(encr_time) +' microseconds' )
total_speed = f_Size/ encr_time
print('Total speed for encryption for 10MB file : ' + str(total_speed) + '  byte/microseconds')

# Decrypt the ciphertext with the same key for  10MB
decry_start = time.time()
decrypter = pyaes.Decrypter(pyaes.AESModeOfOperationCBC(key, iv))

for line in open('cipher_2.txt'):
  decrypted = decrypter.feed(line)
decry_stop =time.time()

#to check total time for decryption
decry_time = (decry_stop - decry_start)*1000000

#print('Decrypted:', decrypted)
print ('Total time for decryption for 10MB file  :' + str(decry_time) + +' microseconds')
total_speed_1 = f_Size/ decry_time
print('Total speed for decryption for 10MB file  : ' + str(total_speed_1) + '  byte/microsecond')
```

**Output 1(a):**

```
Key generation time :  44.82269287109375 microseconds
--------------------------------------------------------------------------------
Total time for encryption for 1KB file :5043.983459472656 microseconds
Total speed for encryption for 1KB file  : 0.20301414709775004  byte/microseconds
Total time for decryption for 1KB file  :415.0867462158203 microseconds
Total speed for decryption for 1KB file  : 2.46695421941413  byte/microsecond
--------------------------------------------------------------------------------
```

## (b) Create a 128-bit AES key, encrypt and decrypt file using AES in the CTR mode.

### Source code:

```
# Code to create AES- 128 bit key and to encrypt and decrypt a text.txt file using AES in CTR mode

import pyaes,  binascii, os, secrets
import time

# Genrating the random key of size 128 bit that is 16 bytes, everytime a new key will be generated when u run this part of the code
key_start= time.time()
key=os.urandom(16)
key_end=time.time()

# to check the total time for key generation
key_gen_time = (key_end - key_start) *1000000

#print('Genrated 128 bit key for CTR mode : ' + str(key) )
print( 'Key generation time :  ' + str(key_gen_time) + ' microseconds')


print('\n--------------------------------------------------------------------------------------------------------------------------------------------------------------')

# Encrypt the SMALL text file stored in the variable plaintext with the given key produced in first step:

#create a initial vector iv
iv = secrets.randbits(128)

#read the text file
with open('1kb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('1kb.txt')
  f_Size= statinfo.st_size


#perform the AES-256-CTR-Encryption using the text, key and iv
ency_start = time.time()
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
ciphertext = aes.encrypt(plaintext)
ency_stop = time.time()

#to check total time for encryption
encr_time = (ency_stop - ency_start)*1000000

#print the encrypted cipher text
#print('Encrypted:', binascii.hexlify(ciphertext))
print ('Total time for encryption for 1kb file :' + str(encr_time) + ' microseconds' )
total_speed = f_Size/ encr_time
print('Total speed for encryption per byte for 1kb file : ' + str(total_speed) + '  bytes/microseconds')
```

```
# Decrypt the ciphertext with the same key for SMALL file
decry_start = time.time()
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
decrypted = aes.decrypt(ciphertext)
decry_stop =time.time()

#to check total time for decryption
decry_time = (decry_stop - decry_start)*1000000

#print('Decrypted:', decrypted)
print ('Total time for decryption for 1kb file :' + str(decry_time) +' microseconds' )

total_speed_1 = f_Size/ decry_time
print('Total speed for decryption per byte for 1 kb file : ' + str(total_speed_1) + '  bytes/microsecond')

print('\n----------')
# Encrypt the LARGE text file stored in the variable plaintext with the given key produced in first step:

#create a initial vector iv
iv = secrets.randbits(128)

#read the text file
with open('10mb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('10mb.txt')
  f_Size= statinfo.st_size


#perform the AES-256-CTR-Encryption using the text, key and iv
ency_start = time.time()
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
ciphertext = aes.encrypt(plaintext)
ency_stop = time.time()

#to check total time for encryption
encr_time = (ency_stop - ency_start)*1000000

#print the encrypted cipher text
#print('Encrypted:', binascii.hexlify(ciphertext))
print ('Total time for encryption for 10MB file :' + str(encr_time) + ' microseconds' )
total_speed = f_Size/ encr_time
print('Total speed for encryption per byte for 10MB file : ' + str(total_speed) + '  bytes/microseconds')

# Decrypt the ciphertext with the same key for LARGE text file
decry_start = time.time()
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
decrypted = aes.decrypt(ciphertext)
decry_stop =time.time()

#to check total time for decryption
decry_time = (decry_stop - decry_start)*1000000

#print('Decrypted:', decrypted)
print ('Total time for decryption for 10MB file :' + str(decry_time) +' microseconds' )

total_speed_1 = f_Size/ decry_time
print('Total speed for decryption per byte for 10MB file : ' + str(total_speed_1) + '  bytes/microseconds')
```

## Output 1(b):

```
Key generation time :  68.42613220214844 microseconds

-----------------------------------------------------------------------------------------------------------------
Total time for encryption for 1kb file :6739.616394042969 microseconds
Total speed for encryption per byte for 1kb file : 0.15193743087590209  bytes/microseconds
Total time for decryption for 1kb file :6639.4805908203125 microseconds
Total speed for decryption per byte for 1 kb file : 0.15422893191611606  bytes/microsecond

-----------------------------------------------------------------------------------------------------------------
Total time for encryption for 10MB file :41901021.24214172 microseconds
Total speed for encryption per byte for 10MB file : 0.25025070246865494  bytes/microseconds
Total time for decryption for 10MB file :41782479.28619385 microseconds
Total speed for decryption per byte for 10MB file : 0.2509606940310218  bytes/microseconds
```

## (c) Create a 256-bit AES key, encrypt and decrypt file using AES in the CTR mode.

### Source code :

```
# Code to create AES- 256 bit key and to encrypt and decrypt 1kb.txt file and 10mb.txt using AES in CTR mode

import pyaes,binascii, os, secrets
import time

# Genrating a 256 bit that is 32 bytes random key , everytime a new key will be generated when u run this part of the code
key_start= time.time()
key=os.urandom(32)
key_end=time.time()

# to check the total time for key generation
key_gen_time = (key_end - key_start)*1000000
print( 'Key generation time :  ' + str(key_gen_time) + ' microseconds')

# Encrypt the  SMALL text file stored in the variable plaintext with the given key produced in first step:
#create a initial vector iv
iv = secrets.randbits(256)

#read the small text file
with open('1kb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('1kb.txt')
  f_Size= statinfo.st_size

#perform the AES-256-CTR-Encryption using the text, key and iv
ency_start = time.time()
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
ciphertext = aes.encrypt(plaintext)
ency_stop = time.time()

#to check total time for encryption in seconds
encr_time = (ency_stop - ency_start)*1000000

print ('Total time for encryption for 1 kb file:' + str(encr_time) + ' microseconds' )
total_speed = f_Size/ encr_time
print('Total speed for encryption per byte for 1kb file: ' + str(total_speed) + '  byte/microseconds')

# Decrypt the ciphertext with the same key for SMALL text file
decry_start = time.time()
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
decrypted = aes.decrypt(ciphertext)
decry_stop =time.time()

#to check total time for decryption in seconds
decry_time = (decry_stop - decry_start)*1000000

#print('Decrypted:', decrypted)
print ('Total time for decryption for 1kb file :' + str(decry_time)+ ' microseconds')
total_speed_d = f_Size/decry_time
print('Total speed for decryption per byte for 1kb file : ' + str(total_speed_d) + '  byte/microseconds')

print('--------------------------------------------------------------------------------')
# Encrypt the LARGE text file stored in the variable plaintext with the given key produced in first step:
#create a initial vector iv
iv = secrets.randbits(256)

#read the small text file
with open('10mb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('10mb.txt')
  f_Size= statinfo.st_size

#perform the AES-256-CTR-Encryption using the text, key and iv
```

```
ency_start = time.time()
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
ciphertext = aes.encrypt(plaintext)
ency_stop = time.time()

#to check total time for encryption in seconds
encr_time = (ency_stop - ency_start)*1000000

#print the encrypted cipher text
print ('Total time for encryption for 10 MB file :' + str(encr_time) + ' microseconds' )
total_speed = f_Size/ encr_time
print('Total speed for encryption per byte for 10 MB file: ' + str(total_speed) + '  byte/microseconds')

# Decrypt the ciphertext with the same key for the LARGE text file
decry_start = time.time()
aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))
decrypted = aes.decrypt(ciphertext)
decry_stop =time.time()

#to check total time for decryption in seconds
decry_time = (decry_stop - decry_start)*1000000

print ('Total time for decryption for 10MB file :' + str(decry_time)+ ' microseconds')
total_speed_d = f_Size/decry_time
print('Total speed for decryption per byte for 10 MB file: ' + str(total_speed_d) + '  byte/microseconds')
```

## Output 1(c):

```
Key generation time :  37.90855407714844 microseconds
Total time for encryption for 1 kb file:6293.773651123047 microseconds
Total speed for encryption per byte for 1kb file: 0.16270048094552617  byte/microseconds
Total time for decryption for 1kb file :5017.518997192383 microseconds
Total speed for decryption per byte for 1kb file : 0.204084927346163  byte/microseconds
--------------------------------------------------------------------------------
Total time for encryption for 10 MB file :53461489.91584778 microseconds
Total speed for encryption per byte for 10 MB file: 0.19613669608732076  byte/microseconds
Total time for decryption for 10MB file :53323035.47859192 microseconds
Total speed for decryption per byte for 10 MB file: 0.19664596934302087  byte/microseconds
```

## (d) To compute the Hash of 1KB and 10MB files using SHA-256 , SHA-512, SHA3-256.

### Source code:

```
# To compute Hash of 1kb file using SHA256
import hashlib, os
import time

print('Hash of 1KB file using SHA256 : ')
with open('1kb.txt','rb') as f:
    bytes = f.read() # read entire file as bytes
    statinfo = os.stat('1kb.txt')
    f_Size= statinfo.st_size
    hash_start = time.time()

    readable_hash = hashlib.sha256(bytes).hexdigest();

    hash_stop = time.time()
    total_time = hash_stop - hash_start
    time_micro = total_time * 1000000

    print ('Total time taken to hash the 1 KB file is :   '  + str(time_micro) + ' microseconds')
total_speed = f_Size/ time_micro
print('Total speed for Hashing per byte for 1KB file: ' + str(total_speed) + '  byte/microsecond')

  # To compute Hash of 1kb file using SHA256
import hashlib

print('\nHash of 10MB file using SHA256 : ')
with open('10mb.txt','rb') as f:
    bytes = f.read() # read entire file as bytes
    statinfo = os.stat('1kb.txt')
    f_Size= statinfo.st_size
    hash_start = time.time()

    readable_hash = hashlib.sha256(bytes).hexdigest();

    hash_stop = time.time()
    total_time = hash_stop - hash_start
    time_micro = total_time * 1000000

total_speed = f_Size/ time_micro
print ('Total time taken to hash the 10 MB file is :   ' + str(time_micro) +' microseconds')
print('Total speed for Hashing per byte for 10MB file: ' + str(total_speed) + '  byte/microsecond')

print('\n-----------------------------------------------------------------------------------------------')

# To compute Hash of 1kb file using SHA512
import hashlib, os
import time

print('Hash of 1KB file using SHA512 : ')
with open('1kb.txt','rb') as f:
    bytes = f.read() # read entire file as bytes
    statinfo = os.stat('1kb.txt')
    f_Size= statinfo.st_size
    hash_start = time.time()

    readable_hash = hashlib.sha512(bytes).hexdigest();

    hash_stop = time.time()
    total_time = hash_stop - hash_start
    time_micro = total_time * 1000000

    print ('Total time taken to hash the 1 KB file is :   '  + str(time_micro) + ' microseconds')
total_speed = f_Size/ time_micro
print('Total speed for Hashing per byte for 1KB file: ' + str(total_speed) + '  byte/microsecond')
```

```
  # To compute Hash of 1kb file using SHA512
import hashlib

print('\nHash of 10MB file using SHA512 : ')
with open('10mb.txt','rb') as f:
    bytes = f.read() # read entire file as bytes
    statinfo = os.stat('1kb.txt')
    f_Size= statinfo.st_size
    hash_start = time.time()

    readable_hash = hashlib.sha512(bytes).hexdigest();

    hash_stop = time.time()
    total_time = hash_stop - hash_start
    time_micro = total_time * 1000000

total_speed = f_Size/ time_micro
print ('Total time taken to hash the 10 MB file is :   ' + str(time_micro) +' microseconds')
print('Total speed for Hashing per byte for 10MB file: ' + str(total_speed) + '  byte/microsecond')

print('\n----------------------------------------------------------------------------------------------------')


# To compute Hash of 1kb file using SHA3_256
import hashlib, os
import time

print('Hash of 1KB file using SHA3_256 : ')
with open('1kb.txt','rb') as f:
    bytes = f.read() # read entire file as bytes
    statinfo = os.stat('1kb.txt')
    f_Size= statinfo.st_size
    hash_start = time.time()

    readable_hash = hashlib.sha3_256(bytes).hexdigest();

    hash_stop = time.time()
    total_time = hash_stop - hash_start
    time_micro = total_time * 1000000

    print ('Total time taken to hash the 1 KB file is :   '  + str(time_micro) + ' microseconds')
total_speed = f_Size/ time_micro
print('Total speed for Hashing per byte for 1KB file: ' + str(total_speed) + '  byte/microsecond')

  # To compute Hash of 1kb file using SHA3_256
import hashlib

print('\nHash of 10MB file using SHA3_256 : ')
with open('10mb.txt','rb') as f:
    bytes = f.read() # read entire file as bytes
    statinfo = os.stat('1kb.txt')
    f_Size= statinfo.st_size
    hash_start = time.time()

    readable_hash = hashlib.sha512(bytes).hexdigest();

    hash_stop = time.time()
    total_time = hash_stop - hash_start
    time_micro = total_time * 1000000

total_speed = f_Size/ time_micro
print ('Total time taken to hash the 10 MB file is :   ' + str(time_micro) +' microseconds')
print('Total speed for Hashing per byte for 10MB file: ' + str(total_speed) + '  byte/microsecond')

print('\n----------------------------------------------------------------------------------------------------')
```

**Output 1(d)**

```
Hash of 1KB file using SHA256 :
Total time taken to hash the 1 KB file is :   12.39776611328125 microseconds
Total speed for Hashing per byte for 1KB file: 82.59552492307692  byte/microsecond

Hash of 10MB file using SHA256 :
Total time taken to hash the 10 MB file is :   30630.826950073242 microseconds
Total speed for Hashing per byte for 10MB file: 0.0334303739715898  byte/microsecond

-----------------------------------------------------------------------------------------------
Hash of 1KB file using SHA512 :
Total time taken to hash the 1 KB file is :   16.689300537109375 microseconds
Total speed for Hashing per byte for 1KB file: 61.35667565714286  byte/microsecond

Hash of 10MB file using SHA512 :
Total time taken to hash the 10 MB file is :   20383.596420288086 microseconds
Total speed for Hashing per byte for 10MB file: 0.050236473431194806  byte/microsecond

-----------------------------------------------------------------------------------------------
Hash of 1KB file using SHA3_256 :
Total time taken to hash the 1 KB file is :   37.90855407714844 microseconds
Total speed for Hashing per byte for 1KB file: 27.01237293081761  byte/microsecond

Hash of 10MB file using SHA3_256 :
Total time taken to hash the 10 MB file is :   19789.21890258789 microseconds
Total speed for Hashing per byte for 10MB file: 0.05174534705187827  byte/microsecond

-----------------------------------------------------------------------------------------------
```

## (g) To create a 2048-bit DSA key, sign the two files and verify the corresponding signatures. If creating a key takes two parameters , use 224 bits for the exponent sizes. If the hash function algorithm needs to specified separately, use SHA-256

### Source Code :

```
import Crypto, os
import time
from Crypto.PublicKey import DSA
from Crypto.Signature import DSS
from Crypto.Hash import SHA256

#****************************************** DSA key generation , signing and verification for 1KB file
**************************************************
# Create a new DSA key
key_start = time.time()
key = DSA.generate(2048)
key_stop = time.time()
key_gen_time = (key_stop - key_start) * 1000000
print('Key Generation time for 1KB file is :  ' + str(key_gen_time) + ' microseconds')

# Sign the 1KB file
with open('1kb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('1kb.txt')
  f_Size= statinfo.st_size

#Hash the input file
hash_start = time.time()
hash_obj = SHA256.new(plaintext)
hash_stop = time.time()
hash_total = (hash_stop - hash_start)*1000000
print('Total time taken for hashing while signing the 1KB file is :  ' + str(hash_total) + ' microseconds')
hash_speed = f_Size/hash_total
print('Total speed taken for hashing while signing the 1KB file is :  ' + str(hash_speed)+ ' bytes/microseconds')

#sign the 1KB file
sign_start = time.time()
signer = DSS.new(key, 'fips-186-3')
signature = signer.sign(hash_obj)
sign_stop = time.time()
total_sign_time = (sign_stop - sign_start)*1000000
print('Total time to produce a signature for 1KB file : ' + str(total_sign_time) + ' microseconds')

# Given the plaintext and signature compute the hash and verify the signature
hash_start = time.time()
hash_obj = SHA256.new(plaintext)
hash_stop = time.time()
hash_total = (hash_stop - hash_start)*1000000
print('Total time taken for hashing while verifying the signature for the 1KB file is :  ' + str(hash_total) + ' microseconds')
hash_speed = f_Size/hash_total
print('Total speed taken for hashing while verifying the signature for the 1KB file is :  ' + str(hash_speed)+ '
bytes/microseconds')

verify_sign_start = time.time()
verifier = DSS.new(key, 'fips-186-3')
# Verify the authenticity of the text file
try:
  verifier.verify(hash_obj, signature)
  verify_sign_stop = time.time()
  total_time_verify = (verify_sign_stop - verify_sign_start) * 1000000
  print( "The time to verify the signature on 1KB file is : " + str(total_time_verify)+ ' microseconds')
except ValueError:
  print( "The text file is not authentic")

print('\n----------------------------------------------------------------------------')
```

```
#********************************************** DSA key generation , signing and verification for 10 MB file
**************************************************

# Create a new DSA key
key_start = time.time()
key = DSA.generate(2048)
key_stop = time.time()
key_gen_time = (key_stop - key_start) * 1000000
print('\nKey Generation time for 10MB file is :  ' + str(key_gen_time)+ ' microseconds')


# Sign the 1KB file
with open('10mb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('10mb.txt')
  f_Size= statinfo.st_size

#Hash the input file
hash_start = time.time()
hash_obj = SHA256.new(plaintext)
hash_stop = time.time()
hash_total = (hash_stop - hash_start)*1000000
print('Total time taken for hashing while signing the 10 MB file is :  ' + str(hash_total)+ ' microseconds')
hash_speed = f_Size/hash_total
print('Total speed taken for hashing while signing the 10 MB file is :  ' + str(hash_speed)+ ' bytes/microseconds')

#sign the 10MB file
sign_start = time.time()
signer = DSS.new(key, 'fips-186-3')
signature = signer.sign(hash_obj)
sign_stop = time.time()
total_sign_time = (sign_stop - sign_start)*1000000
print('Time to produce a signature for 10MB file : ' + str(total_sign_time) + ' microseconds')

# Given the plaintext and signature compute the hash and verify the signature
hash_start = time.time()
hash_obj = SHA256.new(plaintext)
hash_stop = time.time()
hash_total = (hash_stop - hash_start)*1000000
print('Total time taken for hashing while verifying the signature for the 10MB file is :  ' + str(hash_total)+ ' microseconds')
hash_speed = f_Size/hash_total
print('Total speed taken for hashing while verifying the signature for the 10MB file is :  ' + str(hash_speed)+ '
bytes/microseconds')

verify_sign_start = time.time()
verifier = DSS.new(key, 'fips-186-3')
# Verify the authenticity of the text file
try:
  verifier.verify(hash_obj, signature)
  verify_sign_stop = time.time()
  total_time_verify = (verify_sign_stop - verify_sign_start) * 1000000
  print( "The time to verify the signature on 10MB file is : " + str(total_time_verify)+ ' microseconds')
except ValueError:
  print( "The text file is not authentic")
```

## Output (g):

```
Key Generation time for 1KB file is :  2375499.2485046387 microseconds
Total time taken for hashing while signing the 1KB file is :  123.9776611328125 microseconds
Total speed taken for hashing while signing the 1KB file is :  8.259552492307693 bytes/microseconds
Total time to produce a signature for 1KB file : 1259.8037719726562 microseconds
Total time taken for hashing while verifying the signature for the 1KB file is :  389.8143768310547 microseconds
Total speed taken for hashing while verifying the signature for the 1KB file is :  2.6268913125382265 bytes/microseconds
The time to verify the signature on 1KB file is : 1852.273941040039 microseconds

---------------------------------------------------------------------------

Key Generation time for 10MB file is :  85862.87498474121 microseconds
Total time taken for hashing while signing the 10 MB file is :  111675.26245117188 microseconds
Total speed taken for hashing while signing the 10 MB file is :  93.89510057865073 bytes/microseconds
Time to produce a signature for 10MB file : 1435.7566833496094 microseconds
Total time taken for hashing while verifying the signature for the 10MB file is :  109780.78842163086 microseconds
Total speed taken for hashing while verifying the signature for the 10MB file is :  95.51543717947938 bytes/microseconds
The time to verify the signature on 10MB file is : 1555.4428100585938 microseconds
```

## (h) Repeat part (g) with a 3072-bit DSA key (if the second parameter is required, use 256)

### Source Code:

```
import Crypto, os
import time
from Crypto.PublicKey import DSA
from Crypto.Signature import DSS
from Crypto.Hash import SHA256

#********************************************** DSA key generation , signing and verification for 1KB file
*****************************************************
# Create a new DSA key
key_start = time.time()
key = DSA.generate(3072)
key_stop = time.time()
key_gen_time = (key_stop - key_start) * 1000000
print('Key Generation time for 1KB file is :  ' + str(key_gen_time) + ' microseconds')


# Sign the 1KB file
with open('1kb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('1kb.txt')
  f_Size= statinfo.st_size

#Hash the input file
hash_start = time.time()
hash_obj = SHA256.new(plaintext)
hash_stop = time.time()
hash_total = (hash_stop - hash_start)*1000000
print('Total time taken for hashing while signing the 1KB file is :  ' + str(hash_total) + ' microseconds')
hash_speed = f_Size/hash_total
print('Total speed taken for hashing while signing the 1KB file is :  ' + str(hash_speed)+ ' bytes/microseconds')

#sign the 1KB file
sign_start = time.time()
signer = DSS.new(key, 'fips-186-3')
signature = signer.sign(hash_obj)
sign_stop = time.time()
total_sign_time = (sign_stop - sign_start)*1000000
print('Total time to produce a signature for 1KB file : ' + str(total_sign_time) + ' microseconds')

# Given the plaintext and signature compute the hash and verify the signature
hash_start = time.time()
hash_obj = SHA256.new(plaintext)
hash_stop = time.time()
hash_total = (hash_stop - hash_start)*1000000
print('Total time taken for hashing while verifying the signature for the 1KB file is :  ' + str(hash_total) + ' microseconds')
hash_speed = f_Size/hash_total
print('Total speed taken for hashing while verifying the signature for the 1KB file is :  ' + str(hash_speed)+ '
bytes/microseconds')


verify_sign_start = time.time()
verifier = DSS.new(key, 'fips-186-3')
# Verify the authenticity of the text file
try:
  verifier.verify(hash_obj, signature)
  verify_sign_stop = time.time()
  total_time_verify = (verify_sign_stop - verify_sign_start) * 1000000
  print( "The time to verify the signature on 1KB file is : " + str(total_time_verify)+ ' microseconds')
except ValueError:
  print( "The text file is not authentic")

print('\n---------------------------------------------------------------------------')
```

```
#********************************************** DSA key generation , signing and verification for 10 MB file
**************************************************

# Create a new DSA key
key_start = time.time()
key = DSA.generate(3072)
key_stop = time.time()
key_gen_time = (key_stop - key_start) * 1000000
print('\nKey Generation time for 10MB file is :  ' + str(key_gen_time)+ ' microseconds')


# Sign the 1KB file
with open('10mb.txt','rb') as f :
  plaintext = f.read()
  statinfo = os.stat('10mb.txt')
  f_Size= statinfo.st_size

#Hash the input file
hash_start = time.time()
hash_obj = SHA256.new(plaintext)
hash_stop = time.time()
hash_total = (hash_stop - hash_start)*1000000
print('Total time taken for hashing while signing the 10 MB file is :  ' + str(hash_total)+ ' microseconds')
hash_speed = f_Size/hash_total
print('Total speed taken for hashing while signing the 10 MB file is :  ' + str(hash_speed)+ ' bytes/microseconds')

#sign the 10MB file
sign_start = time.time()
signer = DSS.new(key, 'fips-186-3')
signature = signer.sign(hash_obj)
sign_stop = time.time()
total_sign_time = (sign_stop - sign_start)*1000000
print('Time to produce a signature for 10MB file : ' + str(total_sign_time) + ' microseconds')

# Given the plaintext and signature compute the hash and verify the signature
hash_start = time.time()
hash_obj = SHA256.new(plaintext)
hash_stop = time.time()
hash_total = (hash_stop - hash_start)*1000000
print('Total time taken for hashing while verifying the signature for the 10MB file is :  ' + str(hash_total)+ ' microseconds')
hash_speed = f_Size/hash_total
print('Total speed taken for hashing while verifying the signature for the 10MB file is :  ' + str(hash_speed)+ '
bytes/microseconds')

verify_sign_start = time.time()
verifier = DSS.new(key, 'fips-186-3')
# Verify the authenticity of the text file
try:
  verifier.verify(hash_obj, signature)
  verify_sign_stop = time.time()
  total_time_verify = (verify_sign_stop - verify_sign_start) * 1000000
  print( "The time to verify the signature on 10MB file is : " + str(total_time_verify)+ ' microseconds')
except ValueError:
  print( "The text file is not authentic")
```

**Output (h):**

```
Key Generation time for 1KB file is :  212199.44953918457 microseconds
Total time taken for hashing while signing the 1KB file is :  87.97645568847656 microseconds
Total speed taken for hashing while signing the 1KB file is :  11.63947776693767 bytes/microseconds
Total time to produce a signature for 1KB file : 1702.0702362060547 microseconds
Total time taken for hashing while verifying the signature for the 1KB file is :  74.14817810058594 microseconds
Total speed taken for hashing while verifying the signature for the 1KB file is :  13.810184231511254 bytes/microseconds
The time to verify the signature on 1KB file is : 2473.3543395996094 microseconds

---------------------------------------------------------------------------

Key Generation time for 10MB file is :  6324871.301651001 microseconds
Total time taken for hashing while signing the 10 MB file is :  105064.86892700195 microseconds
Total speed taken for hashing while signing the 10 MB file is :  99.8027228995584 bytes/microseconds
Time to produce a signature for 10MB file : 1908.0638885498047 microseconds
Total time taken for hashing while verifying the signature for the 10MB file is :  106972.45597839355 microseconds
Total speed taken for hashing while verifying the signature for the 10MB file is :  98.02299016223324 bytes/microseconds
The time to verify the signature on 10MB file is : 2999.0673065185547 microseconds
```

# Results and Discussion :

## 1 (a)(b)(c):

| Mode | Key size | File Size | Key generation time (microsec) | Ecryption time (microsec) | Decryption time (microsec) | Encryption Speed (byte/microsec) | Decryption Speed( byte/microsec) |
|---|---|---|---|---|---|---|---|
| CBC | 128 | 1Kb | 44.822693 | 5043.9835 | 415.08675 | 0.2030141 | 2.4669542 |
| | | 10Mb | 44.822693 | - | - | - | |
| CTR | 128 | 1Kb | 6.84E+01 | 6739.6163 | 6639.4805 | 0.151937 | 0.15422 |
| | | 10Mb | 6.84E+01 | 41901021 | 41782479 | 0.25025 | 0.2509 |
| CTR | 256 | 1Kb | 37.908554 | 6293.7737 | 5017.519 | 0.1627005 | 0.2040849 |
| | | 10Mb | 37.908554 | 53461490 | 53323035 | 0.1961367 | 0.196646 |

**Observations:**

- We can notice that the key generation time for CBC is comparatively low considered to that of CTR 128.
- Also we can notice that the encryption time is more in CTR 128 compared to CBC and CTR 256. Also we can notice that the encryption time for 10MB file is way more than decryption time
- We can also notice that the speed per byte for encryption and decryption is more in CBC mode compared to CTR mode.
- I wasn't able to get encryption time for 10MB file .

## 1(d):

| | Hash Time in microseconds | | Hashing speed in byte/microsecond | |
|---|---|---|---|---|
| | 1KB | 10MB | 1KB | 10MB |
| SHA-256 | 12.397766 | 30630.827 | 82.595525 | 0.0334304 |
| SHA-512 | 16.689301 | 20383.596 | 61.356676 | 0.0502365 |
| SHA3-256 | 37.908554 | 19789.219 | 27.012373 | 0.0517453 |

As we know SHA-256 and SHA-512 are from SHA-2 family, where SHA-256 uses 32-bit words and SHA-512 uses 64-bit word size. Also that SHA3-256 is similar to SHA-2 family's SHA-256 however the internal structure differs

**Observations:**

- We can notice that as the word size increases the hash time required for both 1KB and 10MB decreases and the speed (byte/microsecond) increases.
- We can also notice that hashing time taken for 10MB file is way more than hashing time taken for 1 KB file. Hence speed also decreases.

## 1(g)(h):

| | Key Generation time (microsec) | | Hashing1 (microsec) | | Speed for Hashing1 (bytes/microsec) | |
|---|---|---|---|---|---|---|
| Key Size | 1kb | 10mb | 1kb | 10mb | 1kb | 10mb |
| 2048 | 2375499.2 | 85862.875 | 2375499.2 | 111675.26 | 8.2595525 | 93.895101 |
| 3072 | 212199.45 | 6324871.3 | 87.976456 | 105064.87 | 11.639478 | 99.802723 |

| | Time for signature (microsec) | | Time for Hash2 | | Speed Hash2 | | Signature verification | |
|---|---|---|---|---|---|---|---|---|
| Key Size | 1kb | 10mb | 1kb | 10mb | 1kb | 10mb | 1kb | 10mb |
| 2048 | 1259.8038 | 1435.7567 | 389.81438 | 109780.79 | 2.6268913 | 95.515437 | 1852.2739 | 1555.4428 |
| 3072 | 1702.0702 | 1908.0639 | 74.148178 | 106972.46 | 13.810184 | 98.02299 | 2473.3543 | 2999.0673 |

As we know DSA is (Digital Signature Algorithm) is used in public key cryptography. It uses a pair of public and private .It hashes the message and generates a signature. Later this signature is verified by hashing the message again using public key and then verifying the signature.

**Observations:**

- We can notice that key generation time is almost the same even after increase in key size, rather it is less in key size =3072 for 1kB however it is more for size 3072 in case of 10MB
- Hashing time increases with increase in file size. And the hashing time decreases with increase in key size.
- We can notice that Speed for hashing byte per micro sec increases with the increase in key size.
- However the time taken to sign the files increases with increase in key size
- Also, with the increase in key size the time taken for signature verification also increases.

**REFERENCES:**


[1] Class Slides from which I got to know in depth about CBC, CTR, SHA, and DSS.
[2] Learnt about the libraries which could be used for AES CBC,CTR encryption and decryption http://ww1.microchip.com/downloads/en/appnotes/atmel-42508-software-library-for-aes-128-encryption-and-decryption_applicationnote_at10764.pdf
[3] Python cryptography libraries https://cryptography.io/en/latest/
[4] Hash libraries for python https://docs.python.org/3/library/hashlib.html
[5] For DSA key generation libraries in Python I referred to the link : https://pycryptodome.readthedocs.io/en/latest/src/public_key/dsa.html