HOMEWORK 2: REPORT

CSE 565 Fall 2019

BY:
Kapindran Kulandaivelu
Person#: 50316983
Mail: kapindra@buffalo.edu

Contents

SOURCE CODE

Packages Used

```
from Cryptodome.Random import get_random_bytes
from Cryptodome.Cipher import AES, PKCS1_OAEP
from Cryptodome.Util.Padding import pad, unpad
from Cryptodome.Hash import SHA256, SHA512, SHA3_256
from Cryptodome.PublicKey import RSA, DSA
from Cryptodome.Signature import DSS
from datetime import datetime, time
from base64 import b64encode
import os
```

AES-128 (MODE: CBC)

```
def AES_128(file_name):
    print()
    print('AES 128 for ' + file_name + ':    ')
    # ENCRYPTION
    t1 = datetime.now()
    key = get_random_bytes(16)
    t2 = datetime.now()
    key_speed = t2 - t1
    print('TIME TAKEN FOR KEY GENERATION:(micro seconds)     ' +
str(key_speed.microseconds))
    initial_vector = get_random_bytes(16)
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    cipher = AES.new(key, AES.MODE_CBC, iv=initial_vector, use_aesni='true')
    print('KEY:(byte) ' + str(key))
    print('INITIAL VECTOR:(byte)  ' + str(initial_vector))
    # print('PLAINTEXT AS BYTES:  ')
    # print(plaintext)
    # key_str = b64encode(key).decode('utf-8')
    # print(key_str)
    t1 = datetime.now()
    ciphertext = cipher.encrypt(pad(plaintext, AES.block_size))
    t2 = datetime.now()
    enc_speed = t2 - t1
    print('TIME TAKEN FOR ENCRYPTION:(micro seconds)     ' +
str(enc_speed.microseconds))
    # print('CIPHERTEXT AS BYTES: ')
    # print(ciphertext)
    # ct = b64encode(ciphertext).decode('utf-8')
    # print(ct)
    with open(file_name + "_Encrypt_AES128.txt", "w") as text_file:
        text_file.write(str(ciphertext))
    print("THE ENCRYPTED FILE IS SAVED AS: " + file_name + "_Encrypt_AES128.txt
.....SUCCESS!")
    # DECRYPTION
    try:
        plain = AES.new(key, AES.MODE_CBC, iv=initial_vector, use_aesni='true')
        t1 = datetime.now()
        pt = plain.decrypt(ciphertext)
        t2 = datetime.now()
        dec_speed = t2 - t1
        print('TIME TAKEN FOR DECRYPTION:(micro seconds)    ' +
str(dec_speed.microseconds))
        plaintext_decrypt = unpad(pt, AES.block_size).decode('utf-8')
        # print(plaintext_decrypt)
        with open(file_name + "_Decrypt_AES128.txt", "w") as text_file:
            text_file.write(plaintext_decrypt)
        print("THE DECRYPTED FILE IS SAVED AS: " + file_name +
"_Decrypt_AES128.txt .....SUCCESS!")
    except (ValueError, KeyError):
        print("INCORRECT DECRYPTION!")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    enc_byte = enc_speed.microseconds / size
    dec_byte = dec_speed.microseconds / size
    print("ENCRYPTION SPEED PER BYTE:   " + str(enc_byte))
    print("DECRYPTION SPEED PER BYTE:   " + str(dec_byte))
```

AES (MODE: CTR)

128-BIT KEY

```
def CTR_128(file_name):
    print()
    print('CTR 128 for ' + file_name + ':   ')
    # ENCRYPTION
    t1 = datetime.now()
    key = get_random_bytes(16)
    t2 = datetime.now()
    key_speed = t2 - t1
    print('TIME TAKEN FOR KEY GENERATION:(micro seconds)    ' +
str(key_speed.microseconds))
    #initial_vector = get_random_bytes(16)
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    cipher = AES.new(key, AES.MODE_CTR, use_aesni='true')
    nonce = cipher.nonce
    print('KEY:(byte) ' + str(key))
    print('NONCE:   ' + str(nonce))
    # print('PLAINTEXT AS BYTES:  ')
    # print(plaintext)
    # key_str = b64encode(key).decode('utf-8')
    # print(key_str)
    t1 = datetime.now()
    ciphertext = cipher.encrypt(plaintext)
    t2 = datetime.now()
    enc_speed = t2 - t1
    print('TIME TAKEN FOR ENCRYPTION:(micro seconds)    ' +
str(enc_speed.microseconds))
    # print('CIPHERTEXT AS BYTES: ')
    # print(ciphertext)
    # ct = b64encode(ciphertext).decode('utf-8')
    # print(ct)
    with open(file_name + "_Encrypt_CTR128.txt", "w") as text_file:
        text_file.write(str(ciphertext))
    print("THE ENCRYPTED FILE IS SAVED AS: " + file_name + "_Encrypt_CTR128.txt
.....SUCCESS!")
    # DECRYPTION
    try:
        plain = AES.new(key, AES.MODE_CTR, nonce=nonce, use_aesni='true')
        t1 = datetime.now()
        pt = plain.decrypt(ciphertext)
        t2 = datetime.now()
        dec_speed = t2 - t1
        print('TIME TAKEN FOR DECRYPTION:(micro seconds)    ' +
str(dec_speed.microseconds))
        plaintext_decrypt = pt.decode('utf-8')
        # print(plaintext_decrypt)
        with open(file_name + "_Decrypt_CTR128.txt", "w") as text_file:
            text_file.write(plaintext_decrypt)
        print("THE DECRYPTED FILE IS SAVED AS: " + file_name +
"_Decrypt_CTR128.txt .....SUCCESS!")
    except (ValueError, KeyError):
        print("INCORRECT DECRYPTION!")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    enc_byte = enc_speed.microseconds / size
    dec_byte = dec_speed.microseconds / size
    print("ENCRYPTION SPEED PER BYTE:       " + str(enc_byte))
    print("DECRYPTION SPEED PER BYTE:       " + str(dec_byte))
```

256-BIT KEY

```
def CTR_256(file_name):
    print()
    print('CTR 256 for ' + file_name + ':    ')
    # ENCRYPTION
    t1 = datetime.now()
    key = get_random_bytes(32)
    t2 = datetime.now()
    key_speed = t2 - t1
    print('TIME TAKEN FOR KEY GENERATION:(micro seconds)    ' +
str(key_speed.microseconds))
    #initial_vector = get_random_bytes(16)
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    cipher = AES.new(key, AES.MODE_CTR, use_aesni='true')
    nonce = cipher.nonce
    print('KEY:(byte) ' + str(key))
    print('NONCE:   ' + str(nonce))
    # print('PLAINTEXT AS BYTES:  ')
    # print(plaintext)
    # key_str = b64encode(key).decode('utf-8')
    # print(key_str)
    t1 = datetime.now()
    ciphertext = cipher.encrypt(plaintext)
    t2 = datetime.now()
    enc_speed = t2 - t1
    print('TIME TAKEN FOR ENCRYPTION:(micro seconds)    ' +
str(enc_speed.microseconds))
    # print('CIPHERTEXT AS BYTES: ')
    # print(ciphertext)
    # ct = b64encode(ciphertext).decode('utf-8')
    # print(ct)
    with open(file_name + "_Encrypt_CTR256.txt", "w") as text_file:
        text_file.write(str(ciphertext))
    print("THE ENCRYPTED FILE IS SAVED AS: "    + file_name + "_Encrypt_CTR256.txt
.....SUCCESS!")
    # DECRYPTION
    try:
        plain = AES.new(key, AES.MODE_CTR, nonce=nonce, use_aesni='true')
        t1 = datetime.now()
        pt = plain.decrypt(ciphertext)
        t2 = datetime.now()
        dec_speed = t2 - t1
        print('TIME TAKEN FOR DECRYPTION:(micro seconds)    ' +
str(dec_speed.microseconds))
        plaintext_decrypt = pt.decode('utf-8')
        # print(plaintext_decrypt)
        with open(file_name + "_Decrypt_CTR256.txt", "w") as text_file:
            text_file.write(plaintext_decrypt)
        print("THE DECRYPTED FILE IS SAVED AS: "    + file_name +
"_Decrypt_CTR256.txt .....SUCCESS!")
    except (ValueError, KeyError):
        print("INCORRECT DECRYPTION!")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    enc_byte = enc_speed.microseconds / size
    dec_byte = dec_speed.microseconds / size
    print("ENCRYPTION SPEED PER BYTE:   " + str(enc_byte))
    print("DECRYPTION SPEED PER BYTE:   " + str(dec_byte))
```

SHA

SHA-256

```
def SHA_256(file_name):
    print()
    print('SHA-256:- The hash value for ' + file_name + ' is:   ')
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    hash_gen = SHA256.new()
    t1 = datetime.now()
    hash_gen.update(plaintext)
    t2 = datetime.now()
    hash_time = t2 -t1
    print('TIME TAKEN FOR HASHING:(micro seconds)    ' +
str(hash_time.microseconds))
    print (hash_gen.hexdigest())
    with open(file_name + "_SHA256.txt", "w") as text_file:
        text_file.write(hash_gen.hexdigest())
    print("THE HASH FILE IS SAVED AS: "    + file_name + "_SHA256.txt
.....SUCCESS!")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    hash_byte = hash_time.microseconds / size
    print("HASH SPEED PER BYTE:    " + str(hash_byte))
```

SHA-512

```
def SHA_512(file_name):
    print()
    print('SHA-512:- The hash value for ' + file_name + ' is:    ')
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    hash_gen = SHA512.new()
    t1 = datetime.now()
    hash_gen.update(plaintext)
    t2 = datetime.now()
    hash_time = t2 - t1
    print('TIME TAKEN FOR HASHING:(micro seconds)     ' +
str(hash_time.microseconds))
    print (hash_gen.hexdigest())
    with open(file_name + "_SHA512.txt", "w") as text_file:
        text_file.write(hash_gen.hexdigest())
    print("THE HASH FILE IS SAVED AS: "    + file_name + "_SHA512.txt
.....SUCCESS!")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    hash_byte = hash_time.microseconds / size
    print("HASH SPEED PER BYTE:    " + str(hash_byte))
```

SHA3-256

```
def SHA3__256(file_name):
    print()
    print('SHA3-256:- The hash value for ' + file_name + ' is:    ')
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    hash_gen = SHA3_256.new()
    t1 = datetime.now()
    hash_gen.update(plaintext)
    t2 = datetime.now()
    hash_time = t2 -t1
    print (hash_gen.hexdigest())
    print('TIME TAKEN FOR HASHING:(micro seconds)     ' +
str(hash_time.microseconds))
    with open(file_name + "_SHA3-256.txt", "w") as text_file:
        text_file.write(hash_gen.hexdigest())
    print("THE HASH FILE IS SAVED AS: "    + file_name + "_SHA2-256.txt
.....SUCCESS!")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    print(size)
    hash_byte = hash_time.microseconds / size
    print("HASH SPEED PER BYTE:    " + str(hash_byte))
```

RSA

2048-BIT KEY

```
def RSA_2048(file_name):
    print()
    print('RSA 2048 for ' + file_name + ':   ')
    #KEY GENERATION
    t1 = datetime.now()
    key_gen = RSA.generate(2048)
    t2 = datetime.now()
    key_speed = t2 - t1
    print('TIME TAKEN FOR KEY GENERATION:(micro seconds)    ' + 
str(key_speed.microseconds))
    f = open('RSA2048_CSE565.pem', 'wb')
    f.write(key_gen.export_key('PEM'))
    f.close()
    #ENCRYPTION
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    f = open('RSA2048_CSE565.pem', 'r')
    key = RSA.import_key(f.read())
    # public_key = key.publickey()
    cipher = PKCS1_OAEP.new(key)
    t1 = datetime.now()
    ciphertext = cipher.encrypt(plaintext)
    t2 = datetime.now()
    enc_speed = t2 - t1
    print('TIME TAKEN FOR ENCRYPTION:(micro seconds)    ' + 
str(enc_speed.microseconds))
    with open(file_name + "_Encrypt_RSA2048.txt", "w") as text_file:
        text_file.write(str(ciphertext))
    print("THE ENCRYPTED FILE IS SAVED AS:" + file_name + 
"_Encrypt_RSA2048.....SUCCESS!")
    #DECRYPTION
    try:
        # private_key = key.has_private()
        plain = PKCS1_OAEP.new(key)
        t1 = datetime.now()
        pt = plain.decrypt(ciphertext)
        t2 = datetime.now()
        dec_speed = t2 - t1
        print('TIME TAKEN FOR DECRYPTION:(micro seconds)    ' + 
str(dec_speed.microseconds))
        plaintext_decrypt = pt.decode('utf-8')
        # print(plaintext_decrypt)
        with open(file_name + "_Decrypt_RSA2048.txt", "w") as text_file:
            text_file.write(plaintext_decrypt)
        print("THE DECRYPTED FILE IS SAVED AS:" + file_name + 
"_Decrypt_RSA2048.....SUCCESS!")
    except (ValueError, KeyError):
        print("INCORRECT DECRYPTION!")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    enc_byte = enc_speed.microseconds / size
    dec_byte = dec_speed.microseconds / size
    print("ENCRYPTION SPEED PER BYTE:" + str(enc_byte))
    print("DECRYPTION SPEED PER BYTE:" + str(dec_byte))
```

3072-BIT KEY

```
def RSA_3072(file_name):
    print()
    print('RSA 3072 for ' + file_name + ':    ')
    #KEY GENERATION
    t1 = datetime.now()
    key_gen = RSA.generate(3072)
    t2 = datetime.now()
    key_speed = t2 - t1
    print('TIME TAKEN FOR KEY GENERATION:(micro seconds)    ' +
str(key_speed.microseconds))
    f = open('RSA3072_CSE565.pem', 'wb')
    f.write(key_gen.export_key('PEM'))
    f.close()
    #ENCRYPTION
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    f = open('RSA3072_CSE565.pem', 'r')
    key = RSA.import_key(f.read())
    # public_key = key.publickey()
    cipher = PKCS1_OAEP.new(key)
    t1 = datetime.now()
    ciphertext = cipher.encrypt(plaintext)
    t2 = datetime.now()
    enc_speed = t2 - t1
    print('TIME TAKEN FOR ENCRYPTION:(micro seconds)    ' +
str(enc_speed.microseconds))
    with open(file_name + "_Encrypt_RSA3072.txt", "w") as text_file:
        text_file.write(str(ciphertext))
    print("THE ENCRYPTED FILE IS SAVED AS:" + file_name +
"_Encrypt_RSA3072.....SUCCESS!")
    #DECRYPTION
    try:
        # private_key = key.has_private()
        plain = PKCS1_OAEP.new(key)
        t1 = datetime.now()
        pt = plain.decrypt(ciphertext)
        t2 = datetime.now()
        dec_speed = t2 - t1
        print('TIME TAKEN FOR DECRYPTION:(micro seconds)    ' +
str(dec_speed.microseconds))
        plaintext_decrypt = pt.decode('utf-8')
        # print(plaintext_decrypt)
        with open(file_name + "_Decrypt_RSA3072.txt", "w") as text_file:
            text_file.write(plaintext_decrypt)
        print("THE DECRYPTED FILE IS SAVED AS:" + file_name +
"_Decrypt_RSA3072.....SUCCESS!")
    except (ValueError, KeyError):
        print("INCORRECT DECRYPTION!")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    enc_byte = enc_speed.microseconds / size
    dec_byte = dec_speed.microseconds / size
    print("ENCRYPTION SPEED PER BYTE:" + str(enc_byte))
    print("DECRYPTION SPEED PER BYTE:" + str(dec_byte))
```

DSA

2048-BIT KEY

```
def DSA_2048(file_name):
    print()
    print('DSA 2048 for ' + file_name + ':    ')
    #KEY GENERATION
    t1 = datetime.now()
    key = DSA.generate(2048)
    t2 = datetime.now()
    key_speed = t2 - t1
    print('TIME TAKEN FOR KEY GENERATION:(micro seconds)     ' +
str(key_speed.microseconds))
    f = open("DSA2048_CSE565.pem", "wb")
    f.write(key.publickey().export_key())
    f.close()
    #SIGNING
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    hash_gen = SHA256.new(plaintext)
    signer = DSS.new(key, 'fips-186-3')
    t1 = datetime.now()
    signature = signer.sign(hash_gen)
    t2 = datetime.now()
    sign_time = t2 - t1
    print('TIME TAKEN TO SIGN:(micro seconds)     ' + str(sign_time.microseconds))
    #VERIFIER
    f = open("DSA2048_CSE565.pem", "r")
    hash_gen = SHA256.new(plaintext)
    public_key = DSA.import_key(f.read())
    verifier = DSS.new(public_key, 'fips-186-3')
    try:
        t1 = datetime.now()
        verifier.verify(hash_gen, signature)
        t2 = datetime.now()
        verify_time = t2 - t1
        print('TIME TAKEN TO VERIFY:(micro seconds)     ' +
str(verify_time.microseconds))
        print("THE SIGNATURE MATCHES!   The message is authentic")
    except ValueError:
        print("The message is not authentic.")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    sign_byte = sign_time.microseconds / size
    verify_byte = verify_time.microseconds / size
    print("TIME TO SIGN PER BYTE:    " + str(sign_byte))
    print("TIME TO VERIFY PER BYTE:    " + str(verify_byte))
```

3072-BIT KEY

```
def DSA_3072(file_name):
    print()
    print('DSA 3072 for ' + file_name + ':    ')
    #KEY GENERATION
    t1 = datetime.now()
    key = DSA.generate(3072)
    t2 = datetime.now()
    key_speed = t2 - t1
    print('TIME TAKEN FOR KEY GENERATION:(micro seconds)     ' + 
str(key_speed.microseconds))
    f = open("DSA3072_CSE565.pem", "wb")
    f.write(key.publickey().export_key())
    f.close()
    #SIGNING
    with open (file_name + ".txt", "r") as myfile:
        data=myfile.read()
    plaintext = bytes(data, 'utf-8')
    hash_gen = SHA256.new(plaintext)
    signer = DSS.new(key, 'fips-186-3')
    t1 = datetime.now()
    signature = signer.sign(hash_gen)
    t2 = datetime.now()
    sign_time = t2 - t1
    print('TIME TAKEN TO SIGN:(micro seconds)     ' + str(sign_time.microseconds))
    #VERIFIER
    f = open("DSA3072_CSE565.pem", "r")
    hash_gen = SHA256.new(plaintext)
    public_key = DSA.import_key(f.read())
    verifier = DSS.new(public_key, 'fips-186-3')
    try:
        t1 = datetime.now()
        verifier.verify(hash_gen, signature)
        t2 = datetime.now()
        verify_time = t2 - t1
        print('TIME TAKEN TO VERIFY:(micro seconds)     ' + 
str(verify_time.microseconds))
        print("THE SIGNATURE MATCHES!   The message is authentic")
    except ValueError:
        print("The message is not authentic.")
    statinfo = os.stat(file_name + ".txt")
    size = statinfo.st_size
    sign_byte = sign_time.microseconds / size
    verify_byte = verify_time.microseconds / size
    print("TIME TO SIGN PER BYTE:    " + str(sign_byte))
    print("TIME TO VERIFY PER BYTE:    " + str(verify_byte))
```

Main Function

```
if __name__ == "__main__":
    AES_128('INPUT_SMALL')
    AES_128('INPUT_BIG')
    CTR_128('INPUT_SMALL')
    CTR_128('INPUT_BIG')
    CTR_256('INPUT_SMALL')
    CTR_256('INPUT_BIG')
    SHA_256('INPUT_SMALL')
    SHA_256('INPUT_BIG')
    SHA_512('INPUT_SMALL')
    SHA_512('INPUT_BIG')
    SHA3__256('INPUT_SMALL')
    SHA3__256('INPUT_BIG')
    RSA_2048('INPUT_SMALL')
    RSA_2048('INPUT_BIG')
    RSA_3072('INPUT_SMALL')
    RSA_3072('INPUT_BIG')
    DSA_2048('INPUT_SMALL')
    DSA_2048('INPUT_BIG')
    DSA_3072('INPUT_SMALL')
    DSA_3072('INPUT_BIG')
```

RESULTS

AES 128 for INPUT_SMALL:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    6

KEY:(byte) b'\x16>tr\xa1\x7f\x80\xc45p[\xa0N\xaf\xa0\x04'

INITIAL VECTOR:(byte)  b'\xee\x1e\x14\xd0e\xba9\x91ob\xac\xf2\xc0r\x9e2'

TIME TAKEN FOR ENCRYPTION:(micro seconds)    58

THE ENCRYPTED FILE IS SAVED AS: INPUT_SMALL_Encrypt_AES128.txt .....SUCCESS!

TIME TAKEN FOR DECRYPTION:(micro seconds)    22

THE DECRYPTED FILE IS SAVED AS: INPUT_SMALL_Decrypt_AES128.txt .....SUCCESS!

ENCRYPTION SPEED PER BYTE:   0.056640625

DECRYPTION SPEED PER BYTE:   0.021484375
```

AES 128 for INPUT_BIG:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    3

KEY:(byte) b'\xd2..4\xf2p\xd2"\x15\x86rC\x1e$\x1eK'

INITIAL VECTOR:(byte)  b'\xa0\xcf\x905\x07\x8b\xca\x16\x9d\xcf\x8a\xcdC\x10{C'

TIME TAKEN FOR ENCRYPTION:(micro seconds)    113021

THE ENCRYPTED FILE IS SAVED AS: INPUT_BIG_Encrypt_AES128.txt .....SUCCESS!

TIME TAKEN FOR DECRYPTION:(micro seconds)    113704
```

```
THE DECRYPTED FILE IS SAVED AS: INPUT_BIG_Decrypt_AES128.txt .....SUCCESS!

ENCRYPTION SPEED PER BYTE:   0.010778522491455078

DECRYPTION SPEED PER BYTE:   0.010843658447265625
```

CTR 128 for INPUT_SMALL:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    11

KEY:(byte) b'4\xd7$\xd4\x02z\x08\xdb\x88#\xd81=5\xb0\r'

NONCE:  b'\xcch}\xad\x01}lP'

TIME TAKEN FOR ENCRYPTION:(micro seconds)    109

THE ENCRYPTED FILE IS SAVED AS: INPUT_SMALL_Encrypt_CTR128.txt .....SUCCESS!

TIME TAKEN FOR DECRYPTION:(micro seconds)    21

THE DECRYPTED FILE IS SAVED AS: INPUT_SMALL_Decrypt_CTR128.txt .....SUCCESS!

ENCRYPTION SPEED PER BYTE:      0.1064453125

DECRYPTION SPEED PER BYTE:      0.0205078125
```

CTR 128 for INPUT_BIG:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    3

KEY:(byte) b'\xc0\xc0\xdc\nph[\xeb\xd0\xee\xc4T\x87h\xa2\x7f'

NONCE:  b'*\xea$\xb40\xc6\xa4J'

TIME TAKEN FOR ENCRYPTION:(micro seconds)    39533

THE ENCRYPTED FILE IS SAVED AS: INPUT_BIG_Encrypt_CTR128.txt .....SUCCESS!

TIME TAKEN FOR DECRYPTION:(micro seconds)    23201

THE DECRYPTED FILE IS SAVED AS: INPUT_BIG_Decrypt_CTR128.txt .....SUCCESS!

ENCRYPTION SPEED PER BYTE:      0.0037701606750488283

DECRYPTION SPEED PER BYTE:      0.0022126197814941405
```

CTR 256 for INPUT_SMALL:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    7

KEY:(byte) b'\xca\x7f\xe0U\x11\xd6\x95\xafm\x0cy&\x90\x08B\x1aGd";) \n\xb8N3 |~\x0fE/'
```

```
NONCE:  b'\xb5Y\xba\xbf\xc1\x16\xab\xf7'

TIME TAKEN FOR ENCRYPTION:(micro seconds)    19

THE ENCRYPTED FILE IS SAVED AS: INPUT_SMALL_Encrypt_CTR256.txt .....SUCCESS!

TIME TAKEN FOR DECRYPTION:(micro seconds)    10

THE DECRYPTED FILE IS SAVED AS: INPUT_SMALL_Decrypt_CTR256.txt .....SUCCESS!

ENCRYPTION SPEED PER BYTE:   0.0185546875

DECRYPTION SPEED PER BYTE:   0.009765625
```

CTR 256 for INPUT_BIG:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    2

KEY:(byte) b'\x14\x01\xd2\xf4k\x14\xd2-<fJ7\x0f\xfev\x1d\xde.\x81\x9f\xedu(Y*\xab\xfe@\x7ftx`'

NONCE:  b'6\x99\x81\x7f?T\x97\xa1'

TIME TAKEN FOR ENCRYPTION:(micro seconds)    45722

THE ENCRYPTED FILE IS SAVED AS: INPUT_BIG_Encrypt_CTR256.txt .....SUCCESS!

TIME TAKEN FOR DECRYPTION:(micro seconds)    39972

THE DECRYPTED FILE IS SAVED AS: INPUT_BIG_Decrypt_CTR256.txt .....SUCCESS!

ENCRYPTION SPEED PER BYTE:   0.004360389709472656

DECRYPTION SPEED PER BYTE:   0.0038120269775390623
```

SHA-256:- The hash value for INPUT_SMALL is:

```
TIME TAKEN FOR HASHING:(micro seconds)    26

5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef

THE HASH FILE IS SAVED AS: INPUT_SMALL_SHA256.txt .....SUCCESS!

HASH SPEED PER BYTE:   0.025390625
```

SHA-256:- The hash value for INPUT_BIG is:

```
TIME TAKEN FOR HASHING:(micro seconds)    43809

e5b844cc57f57094ea4585e235f36c78c1cd222262bb89d53c94dcb4d6b3e55d

THE HASH FILE IS SAVED AS: INPUT_BIG_SHA256.txt .....SUCCESS!

HASH SPEED PER BYTE:   0.00417795181274414
```

SHA-512:- The hash value for INPUT_SMALL is:

```
TIME TAKEN FOR HASHING:(micro seconds)    13

8efb4f73c5655351c444eb109230c556d39e2c7624e9c11abc9e3fb4b9b9254218cc5085b454a9698d0
85cfa92198491f07a723be4574adc70617b73eb0b6461

THE HASH FILE IS SAVED AS: INPUT_SMALL_SHA512.txt .....SUCCESS!

HASH SPEED PER BYTE:   0.0126953125
```

SHA-512:- The hash value for INPUT_BIG is:

```
TIME TAKEN FOR HASHING:(micro seconds)    28162

868d3a190f2723758d1a64498a4ac1f14b0297e16e731a0eec3a446b775c65cb8428ab33140cee13ef5
1e7bb3764b5ff1900cfb342a3dbf3fcc41dd6cdd9fcea

THE HASH FILE IS SAVED AS: INPUT_BIG_SHA512.txt .....SUCCESS!

HASH SPEED PER BYTE:   0.002685737609863281
```

SHA3-256:- The hash value for INPUT_SMALL is:

```
6841b2c10aa6e5f7a384143e4de58fbc9aa28a4b742e9ad4ed14ba148a723a43

TIME TAKEN FOR HASHING:(micro seconds)    16

THE HASH FILE IS SAVED AS: INPUT_SMALL_SHA2-256.txt .....SUCCESS!

HASH SPEED PER BYTE:   0.015625
```

SHA3-256:- The hash value for INPUT_BIG is:

```
50b7513f2a2a2eb9687a07917bff807247f43ae715fa58b7c8e5620c947c814a

TIME TAKEN FOR HASHING:(micro seconds)    63479

THE HASH FILE IS SAVED AS: INPUT_BIG_SHA2-256.txt .....SUCCESS!

HASH SPEED PER BYTE:   0.006053829193115234
```

DSA 2048 for INPUT_SMALL:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    450443
TIME TAKEN TO SIGN:(micro seconds)    507

TIME TAKEN TO VERIFY:(micro seconds)    674

THE SIGNATURE MATCHES!   The message is authentic

TIME TO SIGN PER BYTE:   0.4951171875

TIME TO VERIFY PER BYTE:   0.658203125
```

DSA 2048 for INPUT_BIG:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    472329

TIME TAKEN TO SIGN:(micro seconds)    666

TIME TAKEN TO VERIFY:(micro seconds)    667

THE SIGNATURE MATCHES!   The message is authentic

TIME TO SIGN PER BYTE:   0.0000635147094726

TIME TO VERIFY PER BYTE:   0.0000636100769042
```

DSA 3072 for INPUT_SMALL:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    681670

TIME TAKEN TO SIGN:(micro seconds)    1098

TIME TAKEN TO VERIFY:(micro seconds)    1454

THE SIGNATURE MATCHES!   The message is authentic

TIME TO SIGN PER BYTE:   1.072265625

TIME TO VERIFY PER BYTE:   1.419921875
```

DSA 3072 for INPUT_BIG:

```
TIME TAKEN FOR KEY GENERATION:(micro seconds)    795287

TIME TAKEN TO SIGN:(micro seconds)    1010

TIME TAKEN TO VERIFY:(micro seconds)    1510

THE SIGNATURE MATCHES!   The message is authentic

TIME TO SIGN PER BYTE:   0.000096321111059570

TIME TO VERIFY PER BYTE:   0.00014400482177734375
```

TABULATION OF THE RESULTS

AES-128 CBC	KEY GENERATION	ENCRYPTION	DECRYPTION	ENCRYPTION PER BYTE	DECRYPTION PER BYTE
INPUT_SMALL	6	58	22	0.056640625	0.021484375
INPUT_BIG	3	113021	113704	0.010778522491455078	0.010843658447265625

AES-128 CTR	KEY GENERATION	ENCRYPTION	DECRYPTION	ENCRYPTION PER BYTE	DECRYPTION PER BYTE
INPUT_SMALL	11	109	21	0.1064453125	0.0205078125
INPUT_BIG	3	39533	23201	0.0037701606750488283	0.0022126197814941405

AES-256 CTR	KEY GENERATION	ENCRYPTION	DECRYPTION	ENCRYPTION PER BYTE	DECRYPTION PER BYTE
INPUT_SMALL	7	19	10	0.0185546875	0.009765625
INPUT_BIG	2	45722	39972	0.004360389709472656	0.0038120269775390623

SHA-256	TIME TAKEN FOR HASHING	HASH SPEED PER BYTE
INPUT_SMALL	26	0.025390625
INPUT_BIG	43809	0.0041779518127441414

SHA-512	TIME TAKEN FOR HASHING	HASH SPEED PER BYTE
INPUT_SMALL	13	0.0126953125
INPUT_BIG	28162	0.002685737609863281

SHA3-256	TIME TAKEN FOR HASHING	HASH SPEED PER BYTE
INPUT_SMALL	16	0.015625
INPUT_BIG	63479	0.006053829193115234

DSA 2048	KEY GENERATION	SIGNING TIME	VERIFYING TIME	TIME TO SIGN PER BYTE	TIME TO VERIFY PER BYTE
INPUT_SMALL	450443	507	674	0.4951171875	0.658203125
INPUT_BIG	472329	666	667	0.0000635147094726	0.0000636100769042

DSA 3072	KEY GENERATION	SIGNING TIME	VERIFYING TIME	TIME TO SIGN PER BYTE	TIME TO VERIFY PER BYTE
INPUT_SMALL	681670	1098	1454	1.072265625	1.419921875
INPUT_BIG	795287	1010	1510	0.000096321111059570	0.00014400482177734375

ENVIRONMENT SPECIFICATION

1. Check Python version.
   Recommended: Python 3.6.8
2. Install pip3.
   Instructions:
   ```
   $ sudo apt update
   $ sudo apt install python3-pip
   $ pip3 –version
   ```
   Output: pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)
3. Install PyCryptodome.
   ```
   $ sudo apt-get install build-essential python3-dev
   $ pip3 install pycryptodomex
   ```
4. Run HW2_CSE565.py (Make sure the INPUT_BIG.txt and INPUT_SMALL.txt are in the same directory as the python program)

OBSERVATION AND COMMENTS

1. **Encryption/Decryption per byte**: For AES (CBC and CTR mode), RSA and Hash (SHA) algorithms, the encryption per byte speed is faster for a bigger file(10MB) when compared to a much smaller file(1KB). On the other hand, the encryption per byte speed is very much slower for a bigger file in DSA algorithm. The smaller files is signed comparatively quicker when compared to a bigger file. In AES, the encryption speed per byte increased by approximately 25 times when the file size is increased from 1KB to 10MB. Similar trends are observed in decryption per byte speed.
2. **Encryption and Decryption speed for encryption algorithms**: All three AES algorithms, namely: AES-128 Mode: CBC, AES-128 Mode: CTR and AES-256 Mode: CTR, have similar effect for increase in file. As the file size increases, the encryption time also increases. In other words, the encryption time of the file is proportional to the size of the file. Further, the decryption time is less compared to the encryption time for all the encryption algorithms. We could also conclude that the encryption and decryption time increases by a huge factor when the file size if increased.

3. **Increase in key size:** Comparing AES-128 and AES-256, we can conclude that the increase in key size decreases the time taken for generating the key. For a small file(1KB), AES-128 generated a key in 11 microseconds whereas AES-256 generated a key in 7 microseconds. Therefore, increase in key size results in quicker key generation in AES algorithms. Contradicting the above statement, the key generation time increases in the case of DSA algorithm. Hence, in digital signature algorithm's the increase in key size will result in greater time for key generation. Furthermore, the increase in key size in AES-128 and AES-256 leads to increase in encryption and decryption time of a file. Similar trend is followed in RSA.
4. **Hash Length:** Comparing SHA-256 and SHA-512, it is obvious that the increase in length of the hash results in significantly faster hashing of the file. Hence, increase in hash size give faster hashing time.
5. **Comparison between AES,RSA and Hash:** The RSA algorithm (Public key encryption) is relatively much slower when compared to AES (Symmetric encryption). It is always true that Asymmetric encryption is slow when compared to Symmetric encryption. The slow speed of RSA when compared to AES is because of the absence of a shared key. The Hash is faster compared to AES but provides less authentication and confidentiality. As the size of the hash increases the process of hashing is faster; similar to the effect of increasing key size in AES. Generally, for a given file, Hashing would be the fastest encryption process and RSA would be the slowest. The most secure is AES encryption algorithm.