I confirm that I will keep the content of this Lab confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this Lab. I acknowledge that a mark of 0 may be assigned for copied work."

Name: Siddharth Samber

Student number: 110124156

LAB 5

Master of Applied Computing

Networking & Data Security

COMP 8677

University of Windsor

Submitted By:

Siddharth Samber

110124156

Submission Date:

08 March 2024

QUESTION 1

KEY GENERATION

```
[03/06/24]seed@VM:~$ openssl genrsa -aes128 -out private.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
........+++++
..................+++++
e is 65537 (0x010001)
Enter pass phrase for private.pem:
Verifying - Enter pass phrase for private.pem:
[03/06/24]seed@VM:~$ openssl rsa -in private.pem -pubout > public.pem
Enter pass phrase for private.pem:
writing RSA key
```

PRIVATE KEY

```
[03/06/24]seed@VM:~$ openssl rsa -in private.pem -text -noout
Enter pass phrase for private.pem:
RSA Private-Key: (1024 bit, 2 primes)
modulus:
    00:c4:5c:7c:49:9d:6c:c4:38:ac:62:fd:25:41:61:
    08:ad:c8:3f:17:43:fd:42:ef:96:3c:1b:96:c4:e7:
    51:be:bd:21:bb:4b:b0:0a:c4:ff:8c:24:66:01:33:
    9a:7c:3a:67:6b:be:22:96:29:99:89:6d:d3:be:28:
    23:05:22:a8:85:26:22:86:35:97:97:ae:b6:98:1a:
    2b:36:15:2e:11:eb:16:e1:bd:21:7f:97:13:a8:56:
    13:02:63:2d:8d:5a:9a:e0:a9:0d:2c:05:6e:a8:2c:
    8b:e7:fb:1b:7c:db:b5:e4:88:6d:04:24:e2:df:df:
    60:e3:8d:37:1e:bd:a8:98:5d
publicExponent: 65537 (0x10001)
privateExponent:
    6b:bb:e6:81:29:3b:44:c9:67:63:84:4a:8d:7d:64:
    9a:9c:54:69:3a:67:58:f3:44:b5:43:d9:cb:bc:b2:
    af:f8:ea:e9:ed:13:f0:44:b7:84:b7:6f:b3:d7:11:
    3a:79:7c:c6:b3:72:1c:7b:44:7f:0f:5f:ee:63:ed:
    1a:e3:32:1b:ac:63:99:16:d0:f7:10:c6:ac:0f:d8:
    cd:d6:13:eb:36:76:4d:8d:5a:77:91:93:c3:7c:4b:
    d7:81:e9:18:55:3c:8e:23:0e:b1:12:d2:34:78:56:
    96:c3:a5:bd:30:f6:f5:4a:b0:25:10:4d:fd:14:e1:
    1c:43:af:8d:a7:25:78:6d
prime1:
    00:f7:51:15:9d:c8:f4:84:a7:42:38:d2:24:9d:fc:
    b9:d5:27:dc:1f:0f:86:67:ac:8a:8f:9d:2d:51:5d:
    c6:4d:b9:15:da:0e:da:c6:97:b1:4e:1b:92:61:18:
    50:3f:ea:45:0f:2a:c8:6a:7a:ed:bc:15:14:d7:e7:
    b0:91:27:97:df
```

```
prime2:
    00:cb:41:68:16:d7:92:c5:08:d0:a7:82:55:f5:06:
    cd:64:92:c2:3d:68:34:eb:1d:bf:8b:f4:c1:4c:d7:
    db:48:2a:f7:08:00:5c:19:67:1c:f4:79:3d:83:a7:
    39:bf:12:70:9f:3b:4c:f2:82:1b:91:5f:02:0e:6d:
    83:f4:73:47:43
exponent1:
    19:66:97:8c:c4:15:f1:05:e8:b4:43:49:05:7c:ee:
    6d:29:11:df:92:cd:a0:1a:45:84:e8:84:be:05:9e:
    97:5e:fa:12:92:51:ff:f1:96:81:4a:2d:a7:91:42:
    b4:bd:cf:e7:f5:e1:ed:a5:c0:82:ff:bb:4b:4d:7d:
    81:fb:0a:0b
exponent2:
    2f:a2:59:81:98:ed:f5:ad:56:31:1b:b3:87:64:1b:
    b8:a9:48:4a:59:e4:a5:01:68:c5:a5:0b:b4:35:96:
    57:78:72:42:76:cb:1e:cc:f2:95:3c:e0:b4:bc:f9:
    24:95:6e:bd:72:89:5b:00:30:f3:c5:bf:56:a0:45:
    df:81:30:bf
coefficient:
    49:f3:fc:69:cc:04:a2:f1:01:31:15:1e:64:1e:2d:
    8e:45:7f:15:03:8c:dd:a6:77:b5:02:0d:e0:41:da:
    9e:9f:69:87:71:f8:6a:38:6e:76:e3:29:5f:2e:9f:
    f1:98:ee:8c:85:7a:bd:97:94:37:1c:dc:4c:cc:4c:
    bd:4c:28:97
```

PUBLIC KEY

```
[03/06/24]seed@VM:~$ openssl rsa -in public.pem -pubin -text -noout
RSA Public-Key: (1024 bit)
Modulus:
    00:c4:5c:7c:49:9d:6c:c4:38:ac:62:fd:25:41:61:
    08:ad:c8:3f:17:43:fd:42:ef:96:3c:1b:96:c4:e7:
    51:be:bd:21:bb:4b:b0:0a:c4:ff:8c:24:66:01:33:
    9a:7c:3a:67:6b:be:22:96:29:99:89:6d:d3:be:28:
    23:05:22:a8:85:26:22:86:35:97:97:ae:b6:98:1a:
    2b:36:15:2e:11:eb:16:e1:bd:21:7f:97:13:a8:56:
    13:02:63:2d:8d:5a:9a:e0:a9:0d:2c:05:6e:a8:2c:
    8b:e7:fb:1b:7c:db:b5:e4:88:6d:04:24:e2:df:df:
    60:e3:8d:37:1e:bd:a8:98:5d
Exponent: 65537 (0x10001)
```

QUESTION 2

ENCRYPT CODE

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

message=b'Siddharth Samber 110124156\n'

key=RSA.import_key(open('public.pem').read())


cipher=PKCS1_OAEP.new(key)

ciphertext=cipher.encrypt(message)

f=open('ciphertext.bin', 'wb')

f.write(ciphertext)

f.close()
```

PART A : CIPHER TEXT HEXDUMP

```
[03/06/24]seed@VM:~$ hexdump -C ciphertext.bin
00000000  13 f3 27 70 6e 12 fd ae  8b 90 5d 60 14 61 88 1a  |..'pn.....]`.a..|
00000010  1e 44 42 69 7c 5d 3e b9  63 61 94 a3 71 d0 13 4e  |.DBi|]>.ca..q..N|
00000020  86 20 cb d7 aa e7 3c df  31 d7 07 9d c5 ce ee e8  |. ....<.1.......|
00000030  2a 9b e9 df 69 81 37 11  eb 5b 8e 7f 82 54 b3 a4  |*...i.7..[...T..|
00000040  9c f4 08 58 9f 52 6d 86  6e 25 2a 0e 38 f0 dd 33  |...X.Rm.n%*.8..3|
00000050  9a 05 68 24 f1 d2 b6 74  2c 75 83 74 5c 72 f7 b4  |..h$...t,u.t\r..|
00000060  8e 4b 52 0c 60 63 84 c4  81 4d cb e9 bb 0d cd a4  |.KR.`c...M......|
00000070  5f f5 87 a8 34 66 5b 54  4e 47 44 53 d0 d3 3b 6b  |_...4f[TNGDS..;k|
00000080
```

DECRYPT CODE

```
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

ciphertext = open('ciphertext.bin', 'rb').read()

key_str = open('private.pem').read()
prikey = RSA.import_key(key_str, passphrase='dees')
cipher = PKCS1_OAEP.new(prikey)
message = cipher.decrypt(ciphertext)
print(message)
```

PART B: DECRYPT OUTPUT

```
[03/06/24]seed@VM:~$ sudo python3 decrypt_RSA.py
b'Siddharth Samber 110124156\n'
```

QUESTION 3

PART A

```
[03/06/24]seed@VM:~$ nano sign_RSA.py
[03/06/24]seed@VM:~$ python3 sign_RSA.py
43211516ffb74dd8150d4f6ea4901bba22bd90b0c70746105bfb62fc11155127
[03/06/24]seed@VM:~$ hexdump -C signature.bin
00000000  0d f8 1f 7a 39 4c 05 d0  3d 9b 00 33 0e cb 04 92  |...z9L..=..3....|
00000010  15 e7 63 62 6b 87 ed 93  2b 59 71 2c 59 af 80 2f  |..cbk...+Yq,Y../|
00000020  9a ca 24 d7 60 3f 41 c7  d9 65 7f 88 57 f8 3a e0  |..$.`?A..e..W.:.|
00000030  85 4b 2a 1d fe 3b 91 34  2d 74 51 ab c1 6c e8 05  |.K*..;.4-tQ..l..|
00000040  c1 a8 d4 de c5 40 20 ef  db 6b 03 d6 6b e1 6b 44  |.....@ ..k..k.kD|
00000050  2e 2e dd 76 6a fa 22 9f  b5 52 a7 86 67 09 19 6d  |...vj."..R..g..m|
00000060  09 79 cb 51 ab 3f 34 4b  aa 0c f1 65 47 89 3c 36  |.y.Q.?4K...eG.<6|
00000070  66 4d 42 23 aa 8b 22 c5  6d 2c 8e e2 1a 60 60 63  |fMB#..".m,...``c|
00000080
[03/06/24]seed@VM:~$ nano sign_RSA.py
[03/06/24]seed@VM:~$ python3 sign_RSA.py
4db055d6ddb7a26c117d5baac32b0ada4cbc3601848b54da42095300c5f0ee9f
[03/06/24]seed@VM:~$ hexdump -C signature.bin
00000000  6b 0b 1b 56 a8 6f ef 1b  d1 ee 12 90 d8 2d fb 1f  |k..V.o.......-..|
00000010  ad 91 48 dc 6d 44 bd 9d  81 93 8e 78 9a fb 63 af  |..H.mD.....x..c.|
00000020  06 65 8e 6d 05 67 42 b8  e7 80 47 48 c3 8d cd ce  |.e.m.gB...GH....|
00000030  66 04 78 ec 40 e8 f3 40  9a 57 c6 d3 33 1c fd 81  |f.x.@..@.W..3...|
00000040  b9 c3 c6 3e 82 3a bc cf  66 f5 4d 69 af 8f 0c 10  |...>.:..f.Mi....|
00000050  d3 ca 15 8f 3f 8f a2 8f  0c 1f ed 96 a2 89 a2 96  |....?...........|
00000060  86 50 8e 16 4a ef ec cc  67 21 f5 b2 f6 4e f2 8c  |.P..J...g!...N..|
00000070  fc 25 db 91 1d d3 ee 13  76 9c 1c 93 bc e5 6b 5c  |.%......v.....k\|
00000080
```

The Signatures are not similar for the message with “I owe you $2000” and “I owe you $3000”

Explanation: RSA encryption shows avalanche effect (property where a small change in the input data (message) results in a significantly different output (hash or signature).

PART B

```
[03/06/24]seed@VM:~$ python3 sign_RSA.py
4db055d6ddb7a26c117d5baac32b0ada4cbc3601848b54da42095300c5f0ee9f
[03/06/24]seed@VM:~$ hexdump -C signature.bin
00000000  6b 0b 1b 56 a8 6f ef 1b  d1 ee 12 90 d8 2d fb 1f  |k..V.o.......-..|
00000010  ad 91 48 dc 6d 44 bd 9d  81 93 8e 78 9a fb 63 af  |..H.mD.....x..c.|
00000020  06 65 8e 6d 05 67 42 b8  e7 80 47 48 c3 8d cd ce  |.e.m.gB...GH....|
00000030  66 04 78 ec 40 e8 f3 40  9a 57 c6 d3 33 1c fd 81  |f.x.@..@.W..3...|
00000040  b9 c3 c6 3e 82 3a bc cf  66 f5 4d 69 af 8f 0c 10  |...>.:..f.Mi....|
00000050  d3 ca 15 8f 3f 8f a2 8f  0c 1f ed 96 a2 89 a2 96  |....?...........|
00000060  86 50 8e 16 4a ef ec cc  67 21 f5 b2 f6 4e f2 8c  |.P..J...g!...N..|
00000070  fc 25 db 91 1d d3 ee 13  76 9c 1c 93 bc e5 6b 5c  |.%......v.....k\|
00000080
[03/06/24]seed@VM:~$ python3 verify_RSA.py
4db055d6ddb7a26c117d5baac32b0ada4cbc3601848b54da42095300c5f0ee9f
The signature is valid.
```

The Signature has been verified .

QUESTION 4

Client Program

```
from socket import *

import Crypto.Random.random as r
from Crypto.Hash import SHA256
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad , unpad
p =
25822498780869085896559191720030118743297057928292235128306593565406476220168411946296453532801378314359031719727475597 79
g = 2
#AES functions
def encrypt_message(message, key):
    cipher = AES.new(key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(message.encode('utf-8'), AES.block_size))
    iv = cipher.iv
    return iv + ct_bytes  # Prepend IV to ciphertext for use in decryption

def decrypt_message(ciphertext, key):
    iv = ciphertext[:AES.block_size]  # Extract IV from the beginning
    ct = ciphertext[AES.block_size:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    return pt.decode('utf-8')

def compute_tag(ciphertext):
    hash_obj = SHA256.new(ciphertext)
    return hash_obj.digest()

# Diffi hellman Exchange Functions
def calc_client_private_key():
    return r.getrandbits(400)

def calc_client_public_key(client_private_key):
    return pow(g,client_private_key,p)

def calc_shared_key(server_public_key,client_priavte_key):
    return pow(server_public_key,client_private_key,p)
server_name='127.0.0.1'
server_port=55000
client_socket=socket(AF_INET,SOCK_STREAM)
client_socket.connect((server_name,server_port))
#generate client private key and calculate client public key
client_private_key=calc_client_private_key()
client_public_key=calc_client_public_key(client_private_key)

# Receive server's public key
```

```
server_public_key = int(client_socket.recv(2048).decode())

# Send client's public key to server
client_socket.send(str(client_public_key).encode())



# Compute shared secret
shared_secret = calc_shared_key(server_public_key, client_private_key)
hashed_secret = SHA256.new(str(shared_secret).encode()).digest()
print(f"SK Shared Key: {hashed_secret.hex()}")
while True:
    ######################################################
    # Client sending a message
    message = input("Client: ")
    encrypted_msg = encrypt_message(message, hashed_secret)
    final_message = encrypted_msg + compute_tag(encrypted_msg)
    client_socket.send(final_message)

    if message.lower() == "quit":
        print("Closing connection as requested.")
        client_socket.close()
        break

    # Client receiving a message
    encrypted_data = client_socket.recv(2048)
    if not encrypted_data:
        print("No data received. Closing connection.")
        client_socket.close()
        break

    encrypted_message, received_tag = encrypted_data[:-32], encrypted_data[-
32:]


    if compute_tag(encrypted_message) == received_tag:
        decrypted_message = decrypt_message(encrypted_message, hashed_secret)
        if decrypted_message == "quit":
            print("Server requested to close the connection.")
            client_socket.close()
            break
        print(f"\nServer (decrypted message): {decrypted_message}")
        print(f"Server Cipher text(C): {encrypted_message}")
        print(f"Server Tag (tag): {received_tag}\n")
    else:
        print("Message integrity check failed. Closing connection.")
        client_socket.close()
        break
```

Server Program

```
from socket import *
import Crypto.Random.random as r
from Crypto.Hash import SHA256
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad,unpad
p =
25822498780869085896559191720030118743297057928292235128306593565406476220168411946296453532801378314359031719727475597 79
g = 2
# AES Functions
def encrypt_message(message, key):
    cipher = AES.new(key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(message.encode('utf-8'), AES.block_size))
    iv = cipher.iv
    return iv + ct_bytes  # Prepend IV to ciphertext for use in decryption

def decrypt_message(ciphertext, key):
    iv = ciphertext[:AES.block_size]  # Extract IV from the beginning
    ct = ciphertext[AES.block_size:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    return pt.decode('utf-8')

def compute_tag(ciphertext):
    hash_obj = SHA256.new(ciphertext)
    return hash_obj.digest()

# Diffi Hellman Key Exchange Functions
def calc_server_private_key():
    return r.getrandbits(400)

def calc_server_public_key(server_private_key):
    return pow(g,server_private_key,p)

def calc_shared_key(client_public_key,server_priavte_key):
    return pow(client_public_key,server_private_key,p)

server_port=55000
welcome_socket=socket(AF_INET,SOCK_STREAM)
welcome_socket.bind(('',server_port))
welcome_socket.listen(1)
while True:
    connection_socket,addr=welcome_socket.accept()
    print(f"Connection established with {addr}")
    #generate server private key and calculate server public key
    server_private_key=calc_server_private_key()
    server_public_key=calc_server_public_key(server_private_key)
```

```
        # Send server's public key to client
    connection_socket.send(str(server_public_key).encode())
        # Receive client's public key
    client_public_key = int(connection_socket.recv(2048).decode())
        # Compute shared secret
    shared_secret = calc_shared_key(client_public_key, server_private_key)
    hashed_secret = SHA256.new(str(shared_secret).encode()).digest()
    print(f"SK Shared Key: {hashed_secret.hex()}")

    while True:
        ################################################
        # Server receiving a message
        #let us assume data sent is binary encoded
        encrypted_data = connection_socket.recv(2048)
        if not encrypted_data:
            print("No data received. Closing connection.")
            break

        # tag is appended at the end and is 32 bytes long
        encrypted_message, received_tag = encrypted_data[:-32],
encrypted_data[-32:]


        if compute_tag(encrypted_message) == received_tag:
            decrypted_message = decrypt_message(encrypted_message,
hashed_secret)
            if decrypted_message == "quit":
                print("Client requested to close the connection.")
                connection_socket.close()
                break
            print(f"\nClient (decrypted message): {decrypted_message}")
            print(f"Client Cipher Text (C): {encrypted_message}")
            print(f"Client Tag (tag): {received_tag}\n")
        else:
            print("Message integrity check failed. Closing connection.")
            connection_socket.close()
            break

        # Server sending a message
        #sent data is encoded
        message = input("Server: ")

        if message.lower() == "quit":
            connection_socket.send(encrypt_message(message.lower(),
hashed_secret) + compute_tag("quit"))
            connection_socket.close()
            print("Closing connection as requested.")
            break
```

```
        encrypted_response = encrypt_message(message, hashed_secret)
        response_tag = compute_tag(encrypted_response)
        connection_socket.send(encrypted_response + response_tag)
```

CLIENT AND SERVER COMMUNICATION OUTPUT

CLIENT SIDE

```
[03/08/24]seed@VM:~$ sudo python3 tcp_client.py
SK Shared Key: 22bbb47ff5fa11ae923b12afaca177b1ac443aab652cb73067ebc30e3a9b0423
Client: hello

Server (decrypted message): hello client
Server Cipher text(C): b'k\xd9\xe6\x96\xefP\xc9\xc2\xba"9\xe9\xd0h:\xe5\x8bm\xa5j\xfd{%-\xd1\xf6\x90\x9b\xd4`\xda+'
Server Tag (tag): b'\xd2\xf7\xb6\x81s\xb8\xff\x93-\xbcX\x8eR\xfd\x81\xcd\x10o\x80iC\xf0#$\\\x9c6\xe8\x06\x03\xf8i'

Client: hello server

Server (decrypted message): okay bye client
Server Cipher text(C): b"I\xc7R\x87'\xd4\xac+\xf5\x88\xf8\xad\xb1\xb0\xe5\x10\xcfm\x1e\x87\x8dL=\xf5\x8e\xf3\x8b\xbb\x1d\xaf\xec\xaf"
Server Tag (tag): b"b\x9b\xae\x9d\x8b\xc2\xa1V\xecuy\xda\xf8\x85N\\\xf8\x94\xf9\xe15\xeb'\x9fL\x10\xd58<Q\xd9%"

Client: quit
Closing connection as requested.
[03/08/24]seed@VM:~$
```

SERVER SIDE

```
[03/08/24]seed@VM:~$ sudo python3 tcp_server.py
Connection established with ('127.0.0.1', 44744)
SK Shared Key: 22bbb47ff5fa11ae923b12afaca177b1ac443aab652cb73067ebc30e3a9b0423

Client (decrypted message): hello
Client Cipher Text (C): b'\xfb\x93\xe6\xc3\x04Q\x8b\x88\x9f\xd3\xf6(\x16\xea\xf7{\xe31\r\x17U\xbe9\xa17P\xf2\xce\xa5\xc6s\xf6'
Client Tag (tag): b'\xe4~\x19\x8b$\x08\xe6\xe3\x109\x80\xd6\xf80\xd2\xcd\x1b\x8ci\x07\x1ek\x05N5\xa9(c\x9f\xc7\x9e\x90'

Server: hello client

Client (decrypted message): hello server
Client Cipher Text (C): b'\x88\xcc=6\xc4Ex\x16>\x8f\x81\xa2(\x98W\x0e\xe5\x8bC\x15n^\xac\x030\xaf\x94\x15\x08\xee\xc4\xad'
Client Tag (tag): b'\xd9\x03\xb3#\x91w\x1c\xf0\xae\xa3!\x8bQ\xb5\x8dT\x1c\x7f\xed:W\x7f\xd4[\x83[E\xccb\x1f\xef\xbe'

Server: okay bye client
Client requested to close the connection.
```