**COE451 Mini Project phase 3**

**Khalid Albar**

**201753170**

**For Phase 3 we were asked to authenticate the client and the server to each other, then establishes a session key to be used by them to encrypt/decrypt the exchanged data between them.**

**This report includes all the project phases in addition with phase 3.**

**First the network:**

**I used a VM running Linux ubuntu. The VM has Server deployed on it. The client was deployed in my Main machine.**

**I changed the VM Network adapter to bridged**

**Graphical user interface, text, application, email

Description automatically generated**

**And the VM IP address was:**

**Text

Description automatically generated**

**The Laptop IP Address:**

****

**Testing:**

**Server Bind Socket (Socket used 9000) with address:**

****

**The client connection to server port and address:**

**Text

Description automatically generated**

**Test:**

**In this phase pycryptodome library will be used for providing the secure connection and encryption.**

**First, run the server:**

**After that we run the client:**

**Text

Description automatically generated**

**As shown above the cipher and plain text are shown**

**Here we can start the game:**

**Text

Description automatically generated**

**After the guess is correct, we display the menu again as shown above.**

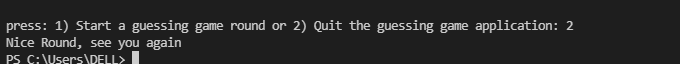
**Now let’s go for another round:**

Text

Description automatically generated

**In the second round a new IV is generated and sent to the server**

**The second round is above. Now we can either quit or go for another round. Now we quit the game by selecting 2:**

****

**As shown above this was a test for the program from the client side.**

**The output in the server will show and print the guess entered by the user as shown below:**

**Text

Description automatically generated**

**Text

Description automatically generated**

**The second round a new IV is sent**

**Text

Description automatically generated**

**As shown the cipher and plaintext are shown**

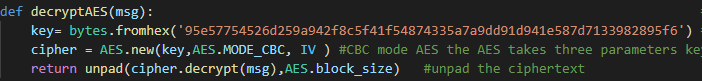
**Function used to encrypt:**

**Text

Description automatically generated**

**The key is shown here and hardcoded**

**Function used to decrypt:**

****

**SHA-256 Hash: for the hashed key I used this website** [**https://emn178.github.io/online-tools/sha256.html**](https://emn178.github.io/online-tools/sha256.html)

**Graphical user interface, text, application, email

Description automatically generated**

**verification with an online tool:**

<https://cryptii.com/pipes/aes-encryption>

I used this online tool to validate my cryptography.

Test 1:

Text

Description automatically generated

Text

Description automatically generated

The plaint text is “Guess game Please”

The generated ciphertext in hex: d82834c76e614e1fc7eb70d50060a74014f7e4cedbb9bec79c97168890da4291

The IV for this round: bf67f91a47ab53993bb354b561fa007e

Graphical user interface, application

Description automatically generated

As shown above the result is validated using the online tool

Test 2:

The plaintext entered from user is “50”

The generated ciphertext in hex: 7b1de37b8b4eeed0ce9970c6d0596b07

The IV for this round: 97f0427bcf2b39488f495c5d605ab4fe

Text

Description automatically generatedGraphical user interface, application

Description automatically generated

As shown above the number exchange between server and client is also ciphered and validated.

**Phase 3 Authentication:**

**Test case 1 – Normal test case:**

**Client side:**

Text

Description automatically generated

As shown above for the a, IV, Ra, Rb, K are printed also a message verifies that bob is authenticated.

Text

Description automatically generated

Text

Description automatically generated

As shown above the game is running with key generated from the authentication process.

**Server side:**

Text

Description automatically generated

As shown above for the a, IV, Ra, Rb, K are printed also a message verifies that Alice is authenticated.

Text

Description automatically generated

Text

Description automatically generated

As shown above the game is running on the server side with key generated from the authentication process. Also, the encryption is working as shown.

Round 2:

**Client side:**

Text

Description automatically generated

As shown above for the a, IV, Ra, Rb, K are printed also a message verifies that bob is authenticated for round 2.

Text

Description automatically generated

Text

Description automatically generated

As shown above the encryption is done and the game is running in the client.

**Server Side:**

**Text

Description automatically generated**

As shown above for the a, IV, Ra, Rb, K are printed also a message verifies that Alice is authenticated for round 2.

Text

Description automatically generated

Text

Description automatically generated

As shown above the game is running on the server side with. Also, the encryption is working as shown for round 2.

**Test case 2 – Trudy posing as Bob:**

**Client side:**

**Text

Description automatically generated**

**Server side:**

**Text

Description automatically generated**

**In this case Trudy is posing as bob and as we can see a message is shown that bob is not authenticated, and the program terminates.**

**Test case 3 – Trudy posing as Alice:**

**Server side:**

**Text

Description automatically generated**

**Client side:**

**Text

Description automatically generated**

**In this case Trudy is posing as Alice and as we can see a message is shown that Alice is not authenticated, and the program terminates.**

Appendix:

New Code for phase 3

def get\_key():

    # N and e client

    Nb = 34777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777790931233333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333336811110763333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333201798777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777777743

    eb = 65537

    m=int

    g=2

    # a generation

    a=int.from\_bytes(get\_random\_bytes(256), byteorder="big")

    print("a = "+str(a))

    # ra generation

    ra = int.from\_bytes(get\_random\_bytes(32), byteorder="big")

    print("ra = "+str(ra))

    ga=pow(g, a, m)

    s.send(int\_bytes(ra))

    s.send(int\_bytes(ga))

    gb=s.recv(10000)

    gb=int.from\_bytes(gb, byteorder='big')

    rb=s.recv(10000)

    rb=int.from\_bytes(rb, byteorder='big')

    print("rb = "+str(rb))

    gab = pow(gb, a, m)

    #print("gab= "+str(gab))

    K = hashlib.sha256(str(gab).encode()).hexdigest()

    #print("K= "+K)

    K=bytes(K,encoding='utf8')

    #print("K="+str(K))

    K=K[:32] # only took the first 32 bytes to to be AES compatiable

    print("K = "+str(K))

    # Alice and Bob IDs

    Alice=1111

    Bob=2222

    H=Alice+Bob+ra+rb+ga+gb+gab #Hash

    Hstr=hashlib.sha256(str(H).encode()).hexdigest()

    H\_auth=int.from\_bytes(bytes(Hstr,encoding='utf8'),byteorder="big")+Bob

    H\_Alice=int.from\_bytes(bytes(Hstr,encoding='utf8'),byteorder="big")+Alice

    #print("K bytes"+str(H\_auth))

    Sb=s.recv(10000)

    Sb=int.from\_bytes(Sb, byteorder='big')

    #print("Sb encrypted= "+str(Sb))

    Sb=pow(Sb,eb,Nb) # public key verfivation

    #print("Sb plain= "+str(Sb))

    if Sb!=H\_auth:

        print("Bob is not authenticated")

        print("Program terminated")# if the recived Sb is not eqaul to the computed hash terminiate the program

        sys.exit()

    else:

        print("Bob is authenticated")

    #print("Sa plain= "+str(H+Alice))

    Sa=pow(H\_Alice,d,N)

    s.send(int\_bytes(Sa))

    #print("Sa encrypted ="+str(Sa))

    Sa\_Alice=str(Sa+Alice)

    Sa\_encry=encryptAES\_auth(Sa\_Alice,K)

    #s.send(Sa\_encry)

    #Destroy a

    a=0

    return K

def int\_bytes(integer):

    int\_str = str(hex(integer))[2:]

    if len(int\_str) % 2 == 1:

        int\_str = '0' + int\_str

    result = bytes.fromhex(int\_str)

    return result

def egcd(a, b):

    if a == 0:

        return (b, 0, 1)

    else:

        g, y, x = egcd(b % a, a)

        return (g, x - (b // a) \* y, y)

def modinv(a, m):

    g, x, y = egcd(a, m)

    if g != 1:

        raise Exception('modular inverse does not exist')

    else:

        return x % m

def encryptAES\_auth(msg,k):                           #The encryption function it takes a string and returns an encrypted bytes

    msg = bytes(msg,encoding='utf8')

    key=k

    #key = bytes.fromhex('95e57754526d259a942f8c5f41f54874335a7a9dd91d941e587d7133982895f6') # KEY 201753170 hashed using sha-256

    cipher = AES.new(key, AES.MODE\_ECB) #CBC mode AES the AES takes three parameters key, mode , IV

    return cipher.encrypt(pad(msg,AES.block\_size))