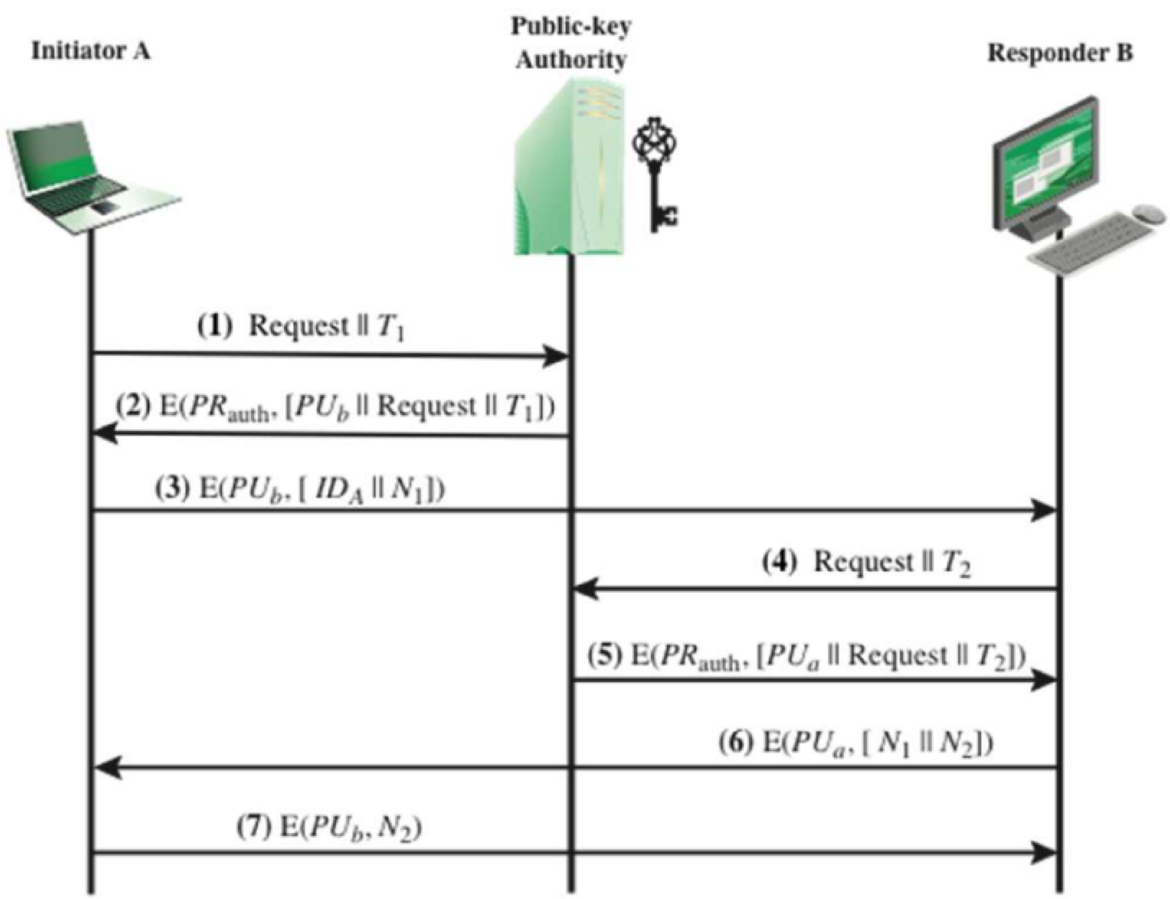
Assignment-3

Keshav Gambhir 2019249
Tanmay Rajore 2019118

Introduction:
In this assignment we have shown a simulation of public key distribution authority(PKDA) with encryption and decryption using the RSA key methods. We have also implemented the RSA algorithm using the Euclidean algorithm. In the implementation we have shown that first clients register their public keys with PKDA. Then they communicate with PKDA to get the public key of the other client with which they want to talk to. Finally they communicate with each other by encrypting the message with a public key and then decrypting the message with the private key.

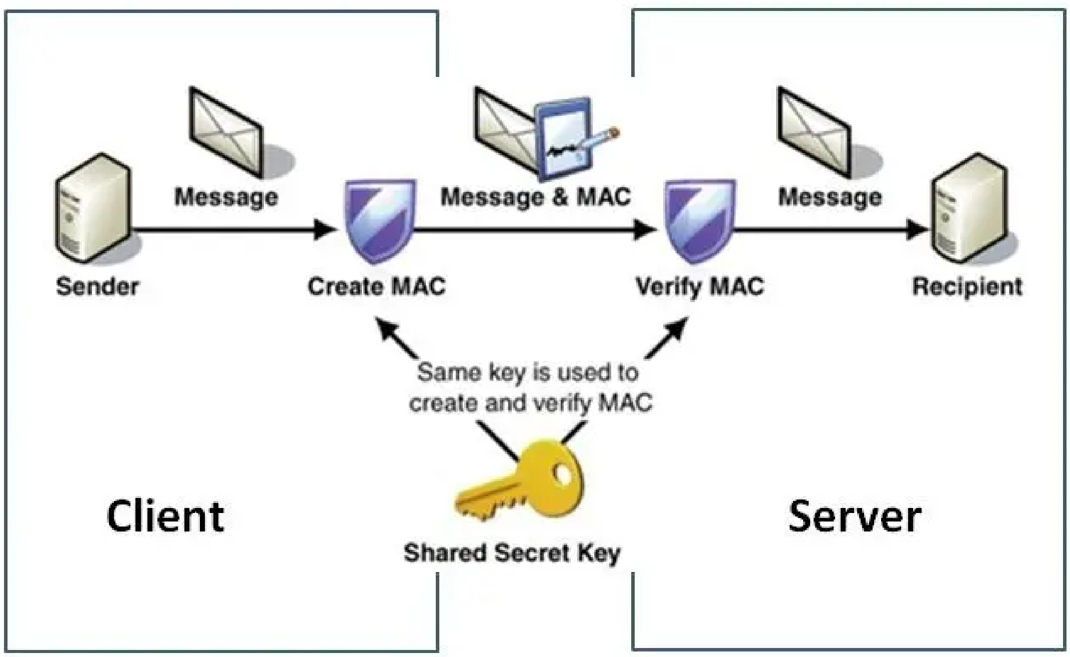
Explanation of Working
The following figure explains the working of the PKDA and the 2 clients with the messages labeled in order of their execution.



Before the execution of this, Initiator A (client A) and Responder B (client B) register their public keys with the Public-key Authority (PKDA). It is assumed that both the clients know the public key of the PKDA. Also HMACs are used for checking the message integrity sent by the PKDA and even between the 2 clients.

Explanation of HMAC

HMAC stands for hashed based message authentication code. This is a method to check message integrity which means whether the message has been tampered with or not. For our communication purposes we have used HMAC as a way to authorize the PKDA and to check the message integrity of the messages sent by the 2 clients. When either of the clients sends a request to PKDA for the public key of another client, the PKDA sends a HMAC signed by its private key along with the public keys and other params. At the receiver end, the receiver unsigns the message with the public key of the PKDA and verifies the HMAC by passing the concatenated message into a pre-negotiated hash function (for our case it is SHA256).



Explanation of RSA Algorithm:

In generating the public and private key in the RSA algorithm we used two prime numbers p and q. Then we calculated the PHI value using the 2 primes. Finally we computed e and d for public and private keys respectively.

Let p and q be 2 prime numbers

n = p * q

phi_n = (p-1)(q-1)

Now we calculated e using the following formula

e from the list{2,phi_n-1} where **gcd(e,phi_n) == 1**

To calculate the the d we used the formula

(e*d)%phi_n = 1 ⇒ modular_inverse(e,phi_n)

Implementation:

```python
def generatePublicPrivateKeysUtil(self,p,q):
        n = gmpy2.mul(p,q)
        phi_n = gmpy2.mul(gmpy2.sub(p,1),gmpy2.sub(q,1))

        e = 2
        while(True):
            if math.gcd(e,phi_n) == 1:
                break
            e += 1

        publicKey = (e,n)

        d = 2
        while(True):
            if ((d*e)%phi_n)  == 1:
                break
            d += 1

        privateKey = (d,n)
        return publicKey, privateKey
```

Encryption

```python
def encrypt(self,plainText,key):
        exponent = key[0]
        modulus = key[1]
        return gmpy2.powmod(plainText,exponent,modulus)
```

Decryption

```python
def decrypt(self,cipherText,key):
        exponent = key[0]
        modulus = key[1]
        return gmpy2.powmod(cipherText,exponent,modulus)
```

Code for registering client with PKDA

```python
ef getNewClientPublicKey(requestMessage,clientID,publicKeyExponent, publicKeyModulus,hmac):
    global clientPublicKeyMap, rsaKey
    print("[Client Public Key Registration Request] clientID: "+str(clientID))
    unsignedHMAC = decryptMessages(hmac)
    unsignedHMACAscii = rsaKey.convertNumberToText(unsignedHMAC)
    generatedHMAC = sha256((requestMessage+"_"+clientID+"_"+publicKeyExponent+"_"+publicKeyModulus).encode()).hexdigest()
    print("Verifying HMAC")
    if unsignedHMACAscii == generatedHMAC:
        print("HMAC verification is done")
        print("Client Added succesfully")
        print()
        clientPublicKeyMap[clientID] = [publicKeyExponent,publicKeyModulus]
        return True
    return False
```

Server request for Public keys of other client

```python
def serveClientRequest(clientID,clientRequestID,timestamp):
    global clientPublicKeyMap, rsaKey, publicKey, privateKey
    print("[Client Public key request] clientID: "+str(clientID)+" clientRequestedID: "+str(clientRequestID)+" timestamp: "+str(timestamp))
    requestedPublicKeyExponent = clientPublicKeyMap[clientRequestID][0]
    requestedPublicKeyModulus = clientPublicKeyMap[clientRequestID][1]

    responseString = clientRequestID+"_"+str(requestedPublicKeyExponent)+"_"+str(requestedPublicKeyModulus)+"_"+str(timestamp)

    hashedResponseString = sha256(responseString.encode('utf-8')).hexdigest()
    integralHash = rsaKey.convertTextToNumbers(hashedResponseString)
    encryptedHash = rsaKey.encrypt(mpz(integralHash),privateKey)
    sendingString = str(responseString)+"_"+str(encryptedHash)
    return sendingString
```

Client-1 Communication code

```python
def communicateWithOtherClient(requestedClientID,clientExponent,clientModulus):
    host = "127.0.0.1"
    port = 6000
    fd = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

    # connect to server on local computer
    fd.connect((host,port))

    print("[Message-2] Sending connection initiation request to "+requestedClientID)
    print("[Message-2] clientID || N1 encrypted with public key of "+requestedClientID)
    nonce = random.randint(1,100)
    message = clientID+"_"+str(nonce)
    hmac = sha256(message.encode()).hexdigest()
    messageIntegers = rsaKeys.convertTextToNumbers(message)
    encryptedMessage = rsaKeys.encrypt(messageIntegers,(clientExponent,clientModulus))
    sendingInitiationMessage = str(encryptedMessage)+"_"+str(hmac)
    fd.send(sendingInitiationMessage.encode('utf-8'))
    print()

    print("[Message-3] Receiving response of N1||N2 from "+requestedClientID)
    receivedMessage = fd.recv(6144)
    encryptedNonce = mpz(receivedMessage.decode('utf-8'))
    decryptedNonce = rsaKeys.decrypt(encryptedNonce,clientPrivateKey)
    decryptedNonceAscii = rsaKeys.convertNumberToText(decryptedNonce)
    receivedNonce1 = decryptedNonceAscii.split("_")[0]
    receivedNonce2 = decryptedNonceAscii.split("_")[1]

    print("[Message-3] Verifying received nonce 1")
    if str(receivedNonce1) == str(nonce):
        print("[Message-3] Received Nonce verified")
    else:
        print("[Message-3] Nonce verification failed")
        return
    print()

    print("[Message-4] Sending N2 encrypted with public key of "+requestedClientID)
    encryptedReceivedNonce2 = str(rsaKeys.encrypt(mpz(rsaKeys.convertTextToNumbers(str(receivedNonce2))),(clientExponent,clientModulus)))
    fd.send(encryptedReceivedNonce2.encode('utf-8'))

    print()
    print("Authentication Complete with "+requestedClientID+". We can send and receive message in encrypted manner")
    print()

    while(True):

        print("Enter the message to send to "+requestedClientID)
        tmp = input()
        messageHMAC = sha256(tmp.encode()).hexdigest()
        sendingMessage = str(rsaKeys.encrypt(mpz(rsaKeys.convertTextToNumbers(tmp)),(clientExponent,clientModulus)))
        sendingMessage = sendingMessage +"_"+messageHMAC
        fd.send(sendingMessage.encode('utf-8'))
        print()

        print("Receiving Encrypted Message")
        receivedMessage = fd.recv(6144)
        receivedMessage = receivedMessage.decode('utf-8')
        receivedMessageHMAC = receivedMessage.split("_")[1]
        receivedMessage = receivedMessage.split("_")[0]
        print("Encrypted Message: "+str(receivedMessage))
        decryptedMessage = rsaKeys.convertNumberToText(rsaKeys.decrypt(mpz(receivedMessage),clientPrivateKey))
        print("Verifying HMAC")
        if sha256(decryptedMessage.encode()).hexdigest() == receivedMessageHMAC:
            print("HMAC Verified")
            print("Message in plain text: "+str(decryptedMessage))
        else:
            print("HMAC verification failed")
            print("Message has been tampered with")
        print()
```

Client-2 communication code

```python
def serveClient(clientFileDescriptor,clientAddress):
    print("Connection Received from: "+str(clientAddress))
    print()
    print("[Message-1] Connection initiation handshake from client")

    message = clientFileDescriptor.recv(6144)
    message = message.decode('utf-8')
    initiationMessageList = message.split("_")
    encryptedMessage = mpz(initiationMessageList[0])
    hmac = initiationMessageList[1]
    decryptedMessage = rsaKeys.decrypt(encryptedMessage,clientPrivateKey)

    decryptedMessageAscii = rsaKeys.convertNumberToText(decryptedMessage)
    generatedHMAC = sha256(decryptedMessageAscii.encode()).hexdigest()

    senderIdentifier = decryptedMessageAscii.split("_")[0]
    senderNonce = decryptedMessageAscii.split("_")[1]
    print("[Message-1] Sender Identification: "+str(senderIdentifier))
    print("[Message-1] Verifying message integrity from sender")
    if generatedHMAC == hmac:
        print("[Message-1] HMAC verified")

    elif generatedHMAC != hmac:
        print("[Message-1] Message has been tampered")
        return False

    print()
    print("[Message-2] Requesting PKDA for public key of "+str(senderIdentifier))
    publicKeyClientExponent,publicKeyClientModulus = requestForKey(senderIdentifier,"Message-2")
    print()

    print("[Message-3] Sending N1||N2 encrypted with public key of "+senderIdentifier)
    nonce = random.randint(1,100)
    sendingMessageNonce = str(senderNonce)+"_"+str(nonce)
    encryptedSendingNonce = str(rsaKeys.encrypt(rsaKeys.convertTextToNumbers(sendingMessageNonce),(publicKeyClientExponent,publicKeyClientModulus)))
    clientFileDescriptor.send(encryptedSendingNonce.encode('utf-8'))
    print()

    print("[Message-4] Receiving encrypted N2")
    receivedNonce2 = clientFileDescriptor.recv(6144)
    decryptedNonce2Ascii = rsaKeys.convertNumberToText(rsaKeys.decrypt(mpz(receivedNonce2.decode('utf-8')),clientPrivateKey))
    print("[Message-4] Verifying Nonce")
    if str(decryptedNonce2Ascii) == str(nonce):
        print("[Message-4] Nonce Verified")
    else:
        print("[Message-4] Nonce not verified")
        return

    print()
    print("Authentication Complete with "+senderIdentifier+". We can send and receive message in encrypted manner")
    print()

    while(True):

        print("Receiving Encrypted Message")
        receivedMessage = clientFileDescriptor.recv(6144)
        receivedMessage = receivedMessage.decode('utf-8')
        receivedMessageHMAC = receivedMessage.split("_")[1]
        receivedMessage = receivedMessage.split("_")[0]
        print("Encrypted Message: "+str(receivedMessage))
        decryptedMessage = rsaKeys.convertNumberToText(rsaKeys.decrypt(mpz(receivedMessage),clientPrivateKey))
        print("Verifying HMAC")
        if sha256(decryptedMessage.encode()).hexdigest() == receivedMessageHMAC:
            print("HMAC Verified for the message")
            print("Message in plain text: "+str(decryptedMessage))
        else:
            print("HMAC verification failed")
            print("Message has been tampered with")
        print()

        print("Enter a message to send to "+senderIdentifier)
        tmp = input()
        sendingMessageHMAC = sha256(tmp.encode()).hexdigest()
        sendingMessage = str(rsaKeys.encrypt(mpz(rsaKeys.convertTextToNumbers(tmp)),(publicKeyClientExponent,publicKeyClientModulus)))
        sendingMessage = sendingMessage + "_" + sendingMessageHMAC
        clientFileDescriptor.send(sendingMessage.encode('utf-8'))
        print()
```