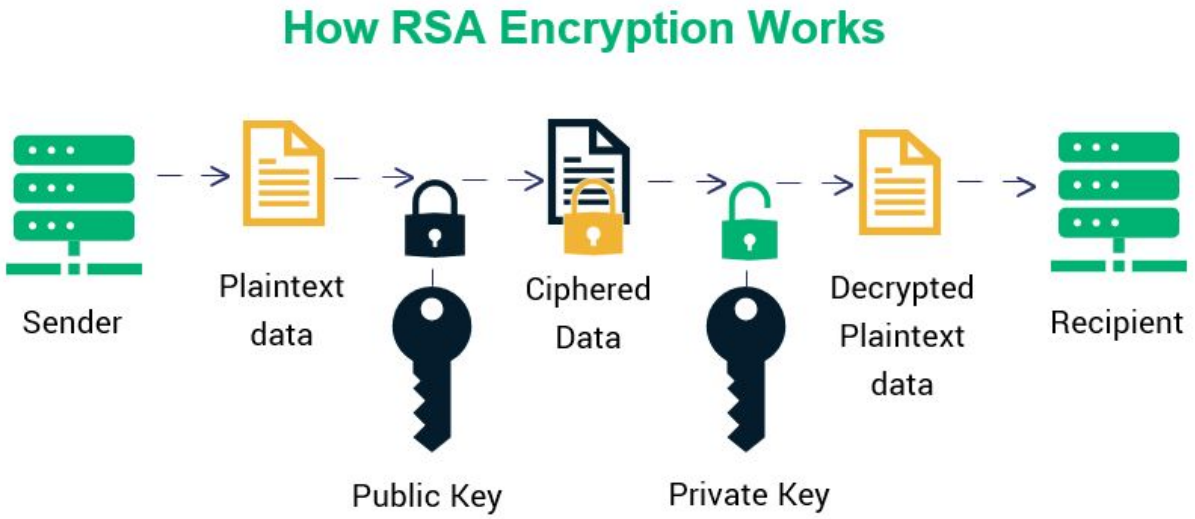
Assignment-3

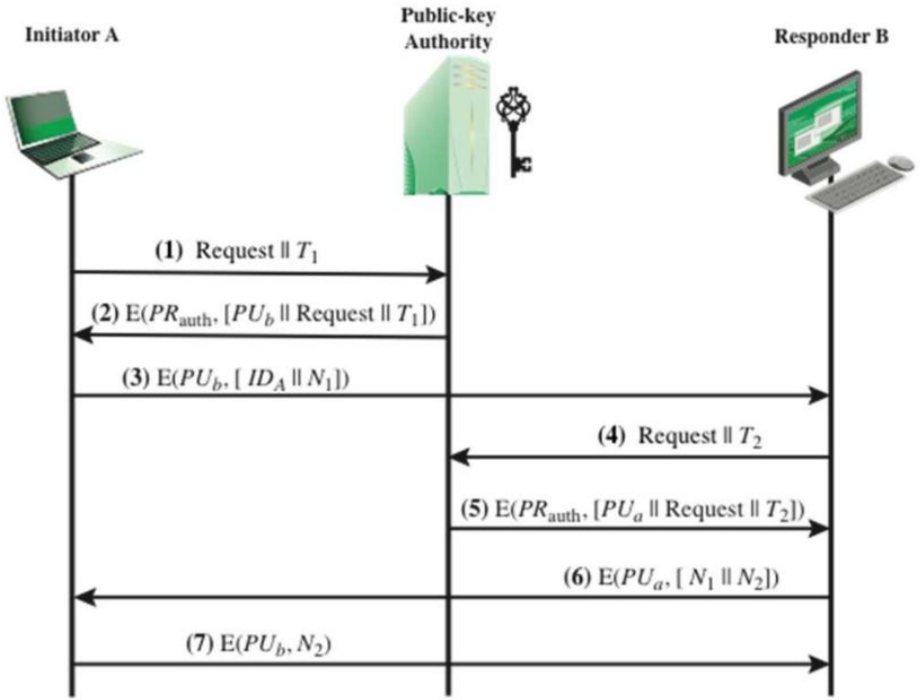
Keshav Gambhir 2019249
Tanmay Rajore 2019118

Introduction

- Implemented a Public key distribution authority to distribute public key for secure communication
- Implemented RSA algorithm for generating public key and private keys



Explanation of working



- Initially clients registers their public key with PKDA
- Initiator sends the public key request to PKDA
- It establishes a secure connection with the other client
- Communicates over the secure channel using a public private RSA keys

Implementation of RSA algorithms

Let p, q be 2 prime numbers

n = p * q

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

e = x ∈ {2 …. phi_n-1} where gcd(x,phi_n) == 1

(e * d) % phi_n = 1

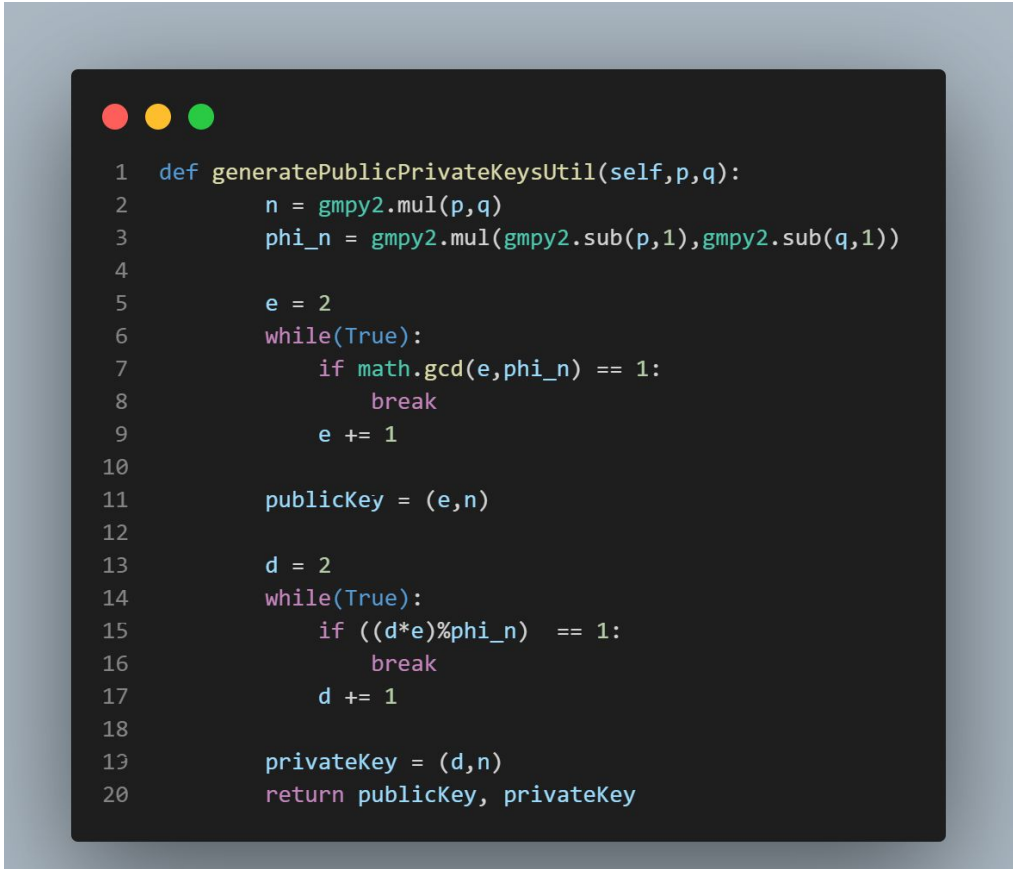
⇒ d = modular_inverse(e,phi_n)

Use of HMAC

- HMAC stands for hashed based message authentication code

- Used to check the message integrity and to verify that the message is coming from the correct authority

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
```

```
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
```

```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
```

Output of Client-1

```
[Message-1] Requesting PKDA for public key of client2
[Message-1] Generating HMAC
[Message-1] Verifying HMAC
[Message-1] HMAC verifies that the message has been sent by PKDA

[Message-2] Sending connection initiation request to client2
[Message-2] clientID || N1 encrypted with public key of client2

[Message-3] Receiving response of N1||N2 from client2
[Message-3] Verifying received nonce 1
[Message-3] Received Nonce verified

[Message-4] Sending N2 encrypted with public key of client2

Authentication Complete with client2. We can send and receive message in encrypted manner

Enter the message to send to client2
hi1

Receiving Encrypted Message
Encrypted Message: 1521073376992845057506454542102132288194899779200252264612435105161716877462922260651519478415704510662287280820500343038289879011214104980578806151062553
Verifying HMAC
HMAC Verified
Message in plain text: got-it1
```

Output of client-2

```
Client2 started at port: 6000
Listening for incomming connections
Connection Received from: ('127.0.0.1', 54139)

[Message-1] Connection initiation handshake from client
Listening for incomming connections
[Message-1] Sender Identification: client1
[Message-1] Verifying message integrity from sender
[Message-1] HMAC verified

[Message-2] Requesting PKDA for public key of client1
[Message-2] Generating HMAC
[Message-2] Verifying HMAC
[Message-2] HMAC verifies that the message has been sent by PKDA

[Message-3] Sending N1||N2 encrypted with public key of client1

[Message-4] Receiving encrypted N2
[Message-4] Verifying Nonce
[Message-4] Nonce Verified

Authentication Complete with client1. We can send and receive message in encrypted manner

Receiving Encrypted Message
Encrypted Message: 5448789098499960270412773035582977249166717185529037004059589407950425573025305861944533864449415622542619864299269507201863811495583324514585837319861255
Verifying HMAC
HMAC Verified for the message
Message in plain text: hi1

Enter a message to send to client1
got-it1
```