Name:-Amrit Ranjan

Reg No:-21BCE1847

Cryptography And Network Security

Assignment 6

Faculty Name:-Dr Tapabarta Roy

RSA Digital Signature Algorithm and RSA Encryption algorithm



Ques:-Construct a RSA Digital Signature Algorithm as well as RSA Encryption algorithm between two entities choosing separate sets of parameters for each of them to ensure confidentiality and authenticity .

Code:-

pip install pycryptodome

pip install pycryptodomex

import random

def gcd(a, b):

    while b:

        a, b = b, a % b

    return a

def mod\_inverse(a, m):

    m0, x0, x1 = m, 0, 1

    while a > 1:

        q = a // m

        m, a = a % m, m

        x0, x1 = x1 - q \* x0, x0

    return x1 + m0 if x1 < 0 else x1

def mod\_exp(base, exp, mod):

    result = 1

    while exp > 0:

        if exp % 2 == 1:

            result = (result \* base) % mod

        base = (base \* base) % mod

        exp //= 2

    return result

def generate\_keypair(bits):

    # Step 1: Choose two large prime numbers, p and q

    p = generate\_prime(bits)

    q = generate\_prime(bits)

    # Step 2: Compute n = pq and Euler's totient function phi(n)

    n = p \* q

    phi\_n = (p - 1) \* (q - 1)

    # Step 3: Choose e such that 1 < e < phi\_n and gcd(e, phi\_n) = 1

    e = random.randrange(2, phi\_n)

    while gcd(e, phi\_n) != 1:

        e = random.randrange(2, phi\_n)

    # Step 4: Compute d, the modular multiplicative inverse of e (mod phi\_n)

    d = mod\_inverse(e, phi\_n)

    return ((n, e), (n, d))

def generate\_prime(bits):

    candidate = random.getrandbits(bits)

    while not is\_prime(candidate):

        candidate = random.getrandbits(bits)

    return candidate

def is\_prime(n, k=5):

    if n <= 1:

        return False

    if n <= 3:

        return True

    if n % 2 == 0:

        return False

    # Write n-1 as 2^r \* d

    r, d = 0, n - 1

    while d % 2 == 0:

        r += 1

        d //= 2

    # Witness loop

    for \_ in range(k):

        a = random.randint(2, n - 2)

        x = mod\_exp(a, d, n)

        if x == 1 or x == n - 1:

            continue

        for \_ in range(r - 1):

            x = mod\_exp(x, 2, n)

            if x == n - 1:

                break

        else:

            return False  # n is composite

    return True  # n is probably prime

def sign\_message(message, private\_key):

    n, d = private\_key

    signature = [mod\_exp(ord(char), d, n) for char in message]

    return signature

def verify\_signature(message, signature, public\_key):

    n, e = public\_key

    expected\_message = ''.join([chr(mod\_exp(char, e, n)) for char in signature])

    return message == expected\_message

def encrypt\_message(message, public\_key):

    n, e = public\_key

    cipher\_text = [mod\_exp(ord(char), e, n) for char in message]

    return cipher\_text

def decrypt\_message(cipher\_text, private\_key):

    n, d = private\_key

    plain\_text = ''.join([chr(mod\_exp(char, d, n)) for char in cipher\_text])

    return plain\_text

# Example usage

message = "Hello, Entity B!"

public\_key\_A, private\_key\_A = generate\_keypair(256)

public\_key\_B, private\_key\_B = generate\_keypair(256)

# Entity A signs and encrypts the message

signature\_A = sign\_message(message, private\_key\_A)

encrypted\_message = encrypt\_message(message, public\_key\_B)

# Entity B verifies the signature and decrypts the message

is\_signature\_valid = verify\_signature(message, signature\_A, public\_key\_A)

decrypted\_message = decrypt\_message(encrypted\_message, private\_key\_B)

# Display results

print(f"Original message: {message}")

print(f"Signature verification: {is\_signature\_valid}")

print(f"Encrypted message: {encrypted\_message}")

print(f"Decrypted message: {decrypted\_message}")

Using Python Library

from Crypto.PublicKey import RSA

from Crypto.Signature import pkcs1\_15

from Crypto.Hash import SHA256

from Crypto.Cipher import PKCS1\_OAEP

import base64

# Generate keys for Entity A (sender)

key\_A = RSA.generate(2048)

private\_key\_A = key\_A.export\_key()

public\_key\_A = key\_A.publickey().export\_key()

# Generate keys for Entity B (receiver)

key\_B = RSA.generate(2048)

private\_key\_B = key\_B.export\_key()

public\_key\_B = key\_B.publickey().export\_key()

# Entity A signs a message

def sign\_message(message, private\_key):

    hash\_value = SHA256.new(message.encode())

    signer = pkcs1\_15.new(RSA.import\_key(private\_key))

    signature = signer.sign(hash\_value)

    return signature

# Entity B verifies the signature

def verify\_signature(message, signature, public\_key):

    hash\_value = SHA256.new(message.encode())

    verifier = pkcs1\_15.new(RSA.import\_key(public\_key))

    try:

        pkcs1\_15.new(RSA.import\_key(public\_key)).verify(hash\_value, signature)

        return True

    except (ValueError, TypeError):

        return False

# Entity A encrypts a message for Entity B

def encrypt\_message(message, public\_key):

    cipher = PKCS1\_OAEP.new(RSA.import\_key(public\_key))

    ciphertext = cipher.encrypt(message.encode())

    return base64.b64encode(ciphertext).decode()

# Entity B decrypts the message from Entity A

def decrypt\_message(ciphertext, private\_key):

    cipher = PKCS1\_OAEP.new(RSA.import\_key(private\_key))

    ciphertext = base64.b64decode(ciphertext)

    decrypted\_message = cipher.decrypt(ciphertext).decode()

    return decrypted\_message

# Example usage

message = "Hello, Entity B!"

# Entity A signs and encrypts the message

signature = sign\_message(message, private\_key\_A)

encrypted\_message = encrypt\_message(message, public\_key\_B)

# Entity B verifies the signature and decrypts the message

is\_signature\_valid = verify\_signature(message, signature, public\_key\_A)

decrypted\_message = decrypt\_message(encrypted\_message, private\_key\_B)

# Display results

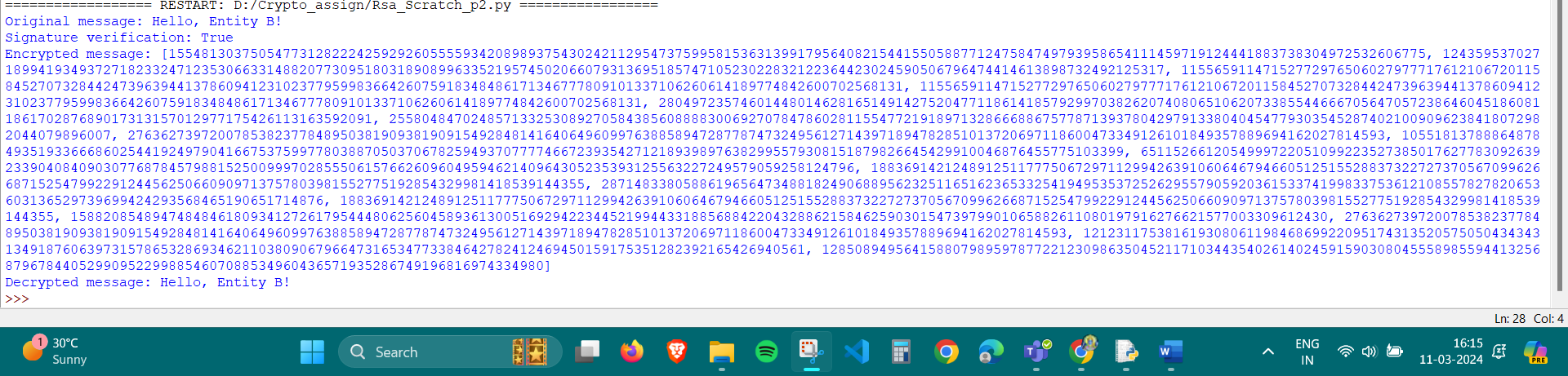
print(f"Signature verification: {is\_signature\_valid}")

print(f"Encrypted message: {encrypted\_message}")

print(f"Decrypted message: {decrypted\_message}")

Output ScreenShots:-

ScreenShots Without Using Python Library



Using Python Library

