Name: Tejas Redkar

Batch: C2; Roll no: 44

PRN: 1032210937

**ICS Lab Assignment 4**

**Lab A4:** Implementation of RSA asymmetric key algorithm using python or java or C++

**Objective of Lab**

1. To introduce the fundamental concepts of discrete logarithmic problem, public key encryption , man in the middle attack and implement RSA Algorithm

**Theory:**

**Example of Using RSA Algorithm:**

The RSA (Rivest–Shamir–Adleman) algorithm is a widely used public-key cryptography system for secure data transmission and encryption. It involves generating a pair of keys: a public key for encryption and a private key for decryption. Here's an example of how RSA encryption and decryption work using simple numbers:

1. Key Generation:

- Choose two large prime numbers, p and q.

- Calculate their product, n = p \* q.

- Calculate the totient (φ) of n: φ(n) = (p - 1)(q - 1).

- Choose a public exponent, e, where 1 < e < φ(n), and it is relatively prime to φ(n). Common choices include 3 or 65537.

- Calculate the private exponent, d, such that (d \* e) % φ(n) = 1.

For this example, let's choose p = 61, q = 53, and e = 17. We calculate n and d:

- n = 61 \* 53 = 3233

- φ(n) = (61 - 1)(53 - 1) = 60 \* 52 = 3120

- e = 17

- Solve for d: d \* 17 % 3120 = 1

- d ≈ 2753

2. Public Key: (e, n)

- Public Exponent (e): 17

- Modulus (n): 3233

3. Private Key: (d, n)

- Private Exponent (d): 2753

- Modulus (n): 3233

4. Encryption:

To encrypt a message M, compute C = M^e mod n.

Let's encrypt a message, "HELLO," which we'll represent as numbers (e.g., A=01, B=02, ..., Z=26).

- H: 08

- E: 05

- L: 12

- L: 12

- O: 15

Encrypt each letter individually:

- H^17 mod 3233 ≈ 2637

- E^17 mod 3233 ≈ 2228

- L^17 mod 3233 ≈ 2730

- L^17 mod 3233 ≈ 2730

- O^17 mod 3233 ≈ 1199

So, the encrypted message is: 2637 2228 2730 2730 1199

5. Decryption:

To decrypt the ciphertext C, compute M = C^d mod n.

Decrypt each block individually:

- 2637^2753 mod 3233 ≈ 08 (H)

- 2228^2753 mod 3233 ≈ 05 (E)

- 2730^2753 mod 3233 ≈ 12 (L)

- 2730^2753 mod 3233 ≈ 12 (L)

- 1199^2753 mod 3233 ≈ 15 (O)

The decrypted message is "HELLO."

In this example, we encrypted and decrypted a simple message using the RSA algorithm. Keep in mind that real-world RSA encryption involves much larger prime numbers and more complex computations to ensure security.

**Algorithm:**

1. Key Generation:

- Select two large prime numbers, typically denoted as `p` and `q`.

- Compute their product, `n = p \* q`. `n` is part of the public key and is used for encryption and as the modulus for various calculations.

- Calculate the totient (Euler's totient function) of `n`, denoted as `φ(n)`. For RSA, `φ(n) = (p - 1) \* (q - 1)`.

- Choose an integer `e` (the public exponent) such that `1 < e < φ(n)` and `e` is coprime (relatively prime) to `φ(n)`. Common choices for `e` include 3 and 65537.

- Compute the private exponent `d`, such that `(d \* e) % φ(n) = 1`. The private exponent `d` is a critical part of the private key.

The public key consists of `(e, n)`, and the private key consists of `(d, n)`.

2. Encryption:

To encrypt a plaintext message `M`, which is typically a number, perform the following:

- Obtain the recipient's public key `(e, n)`.

- Calculate the ciphertext `C` using the formula: `C = M^e mod n`.

3. Decryption:

To decrypt the ciphertext `C` and obtain the original message `M`, use the recipient's private key `(d, n)`:

- Calculate the plaintext `M` using the formula: `M = C^d mod n`.

The security of RSA relies on the difficulty of factoring the product `n = p \* q` into its prime factors. Breaking RSA encryption requires factoring `n`, which becomes increasingly difficult as `n` grows larger.

In practice, key lengths of 2048 bits or 3072 bits are commonly used for RSA encryption to ensure security. Longer key lengths provide stronger security but also require more computational effort.

RSA is widely used for secure data transmission, digital signatures, and various security protocols. However, in recent years, there has been a shift towards using even more secure algorithms, particularly for long-term security needs, due to the potential risks of quantum computers factoring large numbers efficiently.

**Code:**

import math

# step 1

p = 3

q = 7

# step 2

n = p\*q

print("n =", n)

# step 3

phi = (p-1)\*(q-1)

# step 4

e = 2

while(e<phi):

    if (math.gcd(e, phi) == 1):

        break

    else:

        e += 1

print("e =", e)

# step 5

k = 2

d = ((k\*phi)+1)/e

print("d =", d)

print(f'Public key: {e, n}')

print(f'Private key: {d, n}')

# plain text

msg = 11

print(f'Original message:{msg}')

# encryption

C = pow(msg, e)

C = math.fmod(C, n)

print(f'Encrypted message: {C}')

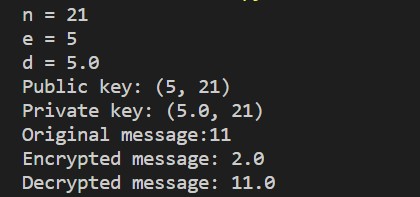
# decryption

M = pow(C, d)

M = math.fmod(M, n)

print(f'Decrypted message: {M}')

**Output Screen shots**:



**Conclusion**:

We successfully implemented RSA asymmetric key algorithm, in Python. This implementation provides a practical demonstration of how this algorithm works for encrypting messages.

# FAQs:

# What is discrete logarithmic problem?

# What is man in middle attack?

# Explain RSA algorithm.