PRN No: 2020BTECS00006

Name: Samrat Vishwas Jadhav

Batch: B1

Assignment: 8

Title of assignment: Implementation of RSA Algorithm

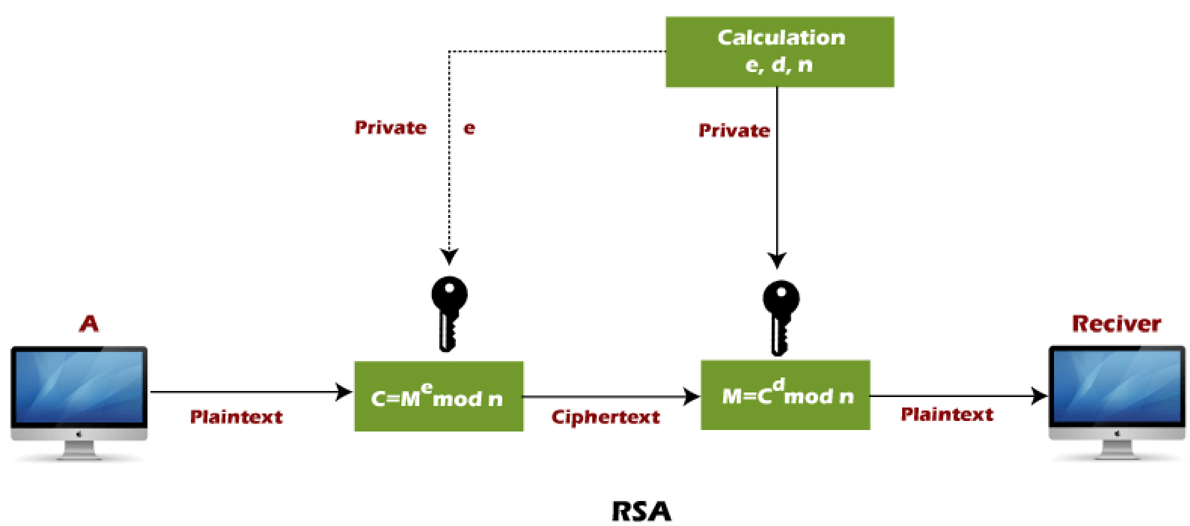
1. **Aim:**
   Implementation of RSA Algorithm

2. **Theory:**

RSA encryption algorithm is a type of public-key encryption algorithm.

RSA encryption algorithm:

RSA is the most common public-key algorithm, named after its inventors **Rivest, Shamir, and Adelman (RSA).**



RSA algorithm uses the following procedure to generate public and private keys:

- Select two large prime numbers, p and q.

- Multiply these numbers to find n = p x q, where n is called the modulus for encryption and decryption.
- Choose a number e less than n, such that n is relatively prime to (p - 1) x (q -1). It means that e and (p - 1) x (q - 1) have no common factor except 1. Choose "e" such that $1<e < \varphi (n)$, e is prime to φ (n),
  gcd (e,d(n)) =1
- If n = p x q, then the public key is <e, n>. A plaintext message m is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.
  $C = m^e \bmod n$
  Here, m must be less than n. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.
- To determine the private key, we use the following formula to calculate the d such that:
  $D_e \bmod \{(p - 1) \times (q - 1)\} = 1$
  Or
  $D_e \bmod \varphi (n) = 1$
- The private key is <d, n>. A ciphertext message c is decrypted using private key <d, n>. To calculate plain text m from the ciphertext c following formula is used to get plain text m.
  $m = c^d \bmod n$

Let's take some example of RSA encryption algorithm:

This example shows how we can encrypt plaintext 9 using the RSA public-key encryption algorithm. This example uses prime numbers 7 and 11 to generate the public and private keys.

Explanation:

Step 1: Select two large prime numbers, p, and **q**.

p = 7

q = 11

Step 2: Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.

First, we calculate

n = p x q

n = 7 x 11

n = 77

Step 3: Choose a number **e** less that **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n), gcd (e, d (n)) =1.

Second, we calculate

φ (n) = (p - 1) x (q-1)

φ (n) = (7 - 1) x (11 - 1)

φ (n) = 6 x 10

φ (n) = 60

Let us now choose relative prime e of 60 as 7.

Thus, the public key is <e, n> = (7, 77)

Step 4: A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.

To find ciphertext from the plain text following formula is used to get ciphertext C.

C = m^e mod n

C = 9^7 mod 77

C = 37

Step 5: The private key is <d, n>. To determine the private key, we use the following formula d such that:

D_e mod {(p - 1) x (q - 1)} = 1

7d mod 60 = 1, which gives d = 43

The private key is <d, n> = (43, 77)

Step 6: A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m.

m = c^d mod n

$m = 37^{43} \bmod 77$

$m = 9$

In this example, Plain text = 9 and the ciphertext = 37

Code:

Using the numbers as Plaintext

```python
# Using the numbers as plaintext

from generate_prime import generate_prime_no, is_prime

# Function to find mod: a^m mod n
def findExpoMod(a, m, n):
    # Decimal to binary conversion
    m_bin = bin(m).replace("0b", "")

    # Convert it into list (individual characters)
    m_bin_lst = [int(i) for i in m_bin]

    # Initialize the list
    a_lst = [a]

    # Functions to perform operations
    # If next value = 0
    def oneOperation(num):
        return (num*num) % n

    # If next value = 1
    def twoOperation(num):
        return (a * oneOperation(num)) % n

    for j in range(len(m_bin_lst)):
        if j+1 == len(m_bin_lst):
            break
```

```
        if(m_bin_lst[j+1] == 0):
            a_lst.append(oneOperation(a_lst[j]))
        else:
            a_lst.append(twoOperation(a_lst[j]))

    return a_lst[-1]

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 gcd(a, h):
    temp = 0
    while(1):
        temp = a % h
        if (temp == 0):
            return h
        a = h
        h = temp

def gen_keys(p, q):
    n = p*q
    phi = (p-1)*(q-1)

    e = 2
    # e must be co-prime to phi and smaller than phi.
    while (e < phi):
        if(gcd(e, phi) == 1):
            break
        else:
            e += 1

    # Private key choosing 'd' such that it satisfies
```

```
    # d*e = 1 mod (phi)
    d = mod_inverse(e, phi)

    print(f"Your Public Key is:\ne = {str(e)}\nn = {str(n)}")
    print(f"Your Private Key is:\nd = {str(d)}\nn = {str(n)}")


def encrypt(M, e, n):
    if(len(str(M))) < n:
        # Encryption: C = (M ^ e) % n
        C = findExpoMod(M, e, n)
        return C
    else:
        print("Message size should be less than 'n' !!")

def decrypt(C, d, n):
    # Decryption: M = (C ^ d) % n
    M = findExpoMod(C, d, n)

    return M

# Main Code
ch = int(input("What do you want to perform?\n1. Generate Public
& Private Keys\n2. Encryption\n3. Decryption\n"))

if (ch == 1):
    gen_r = input("Do you want to generate the prime numbers
automatically ? [y/n]\n")
    if gen_r == 'y':
        dig_p = int(input("Enter the number of digits in first
prime number(p): "))
        p = generate_prime_no(dig_p)
        dig_q = int(input("Enter the number of digits in second
prime number(q): "))
        q = generate_prime_no(dig_q)

        print(f"p = {p}")
```

```python
        print(f"q = {q}")

        gen_keys(p, q)

    elif gen_r == 'n':
        p = int(input("Enter first large prime number(p):\n"))
        if not is_prime(p):
            print(f"Entered number is not prime!")
            exit()

        q = int(input("Enter second large prime number(q):\n"))
        if not is_prime(q):
            print(f"Entered number is not prime!")
            exit()

        gen_keys(p, q)

    else:
        print("Invaild choice!")
        exit()

elif(ch == 2):
    M = int(input("Enter the message to be encrypted:\n"))
    print("Enter the Public Key (e, n):")
    e = int(input("Enter the value of 'e':\n"))
    n = int(input("Enter the value of 'n':\n"))

    C = encrypt(M, e, n)

    print(f"Ciphertext is:\n{str(C)}")

elif(ch == 3):
    C = int(input("Enter the ciphertext to be decrypted:\n"))
    print("Enter the Private Key (d, n):")
    d = int(input("Enter the value of 'd':\n"))
    n = int(input("Enter the value of 'n':\n"))
    M = decrypt(C, d, n)
```

```python
    print(f"Decrypted message is:\n{str(M)}")

else:
    print("Invalid input!")
```

Output:

```
PS D:\Final_BTech_Labs\CNS> & C:/Python310/python.exe "d:/Final_BTech_Labs/CNS/Assignment 8/RSA_a.py"
1. Generate Public & Private Keys
2. Encryption
3. Decryption
1
Do you want to generate the prime numbers automatically ? [y/n]
y
Enter the number of digits in first prime number(p): 5
Enter the number of digits in second prime number(q): 5
p = 13537
q = 21017
Your Public Key is:
e = 5
n = 284507129
Your Private Key is:
d = 227578061
n = 284507129
PS D:\Final_BTech_Labs\CNS> & C:/Python310/python.exe "d:/Final_BTech_Labs/CNS/Assignment 8/RSA_a.py"
What do you want to perform?
1. Generate Public & Private Keys
2. Encryption
3. Decryption
2
Enter the message to be encrypted:
123456
Enter the Public Key (e, n):
Enter the value of 'e':
5
Enter the value of 'n':
284507129
Ciphertext is:
132097498
```

```
PS D:\Final_BTech_Labs\CNS> & C:/Python310/python.exe "d:/Final_BTech_Labs/CNS/Assignment 8/RSA_a.py"
What do you want to perform?
1. Generate Public & Private Keys
2. Encryption
3. Decryption
3
Enter the ciphertext to be decrypted:
132097498
Enter the Private Key (d, n):
Enter the value of 'd':
227578061
Enter the value of 'n':
284507129
Decrypted message is:
123456
PS D:\Final_BTech_Labs\CNS>
```

Using the numbers as Plaintext

Code:

```
from generate_prime import generate_prime_no, is_prime

# Function to find mod: a^m mod n


def findExpoMod(a, m, n):
    return pow(a, m, n)


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 gcd(a, h):
    temp = 0
    while (1):
        temp = a % h
        if (temp == 0):
            return h
        a = h
        h = temp


def gen_keys(p, q):
    n = p*q
    phi = (p-1)*(q-1)

    e = 2
    while (e < phi):
        if (gcd(e, phi) == 1):
```

```python
            break
        else:
            e += 1

    d = mod_inverse(e, phi)

    print(f"Your Public Key is:\ne = {str(e)}\nn = {str(n)}")
    print(f"Your Private Key is:\nd = {str(d)}\nn = {str(n)}")


def encrypt(message, e, n):
    # Convert alphabetic input to numerical values
    numerical_message = [ord(char) - ord('A') for char in
message.upper()]

    # Encryption: C = (M ^ e) % n
    encrypted_message = [findExpoMod(char, e, n) for char in
numerical_message]
    return encrypted_message


def decrypt(encrypted_message, d, n):
    # Decryption: M = (C ^ d) % n
    decrypted_numerical_message = [findExpoMod(
        char, d, n) for char in encrypted_message]

    # Convert back to alphabetic characters
    decrypted_message = ''.join(chr(char + ord('A'))
                                for char in
decrypted_numerical_message)
    return decrypted_message


# Main Code
ch = int(input("What do you want to perform?\n1. Generate Public
& Private Keys\n2. Encryption\n3. Decryption\n"))

if (ch == 1):
```

```
    gen_r = input(
        "Do you want to generate the prime numbers automatically
? [y/n]\n")
    if gen_r == 'y':
        dig_p = int(
            input("Enter the number of digits in first prime
number(p): "))
        p = generate_prime_no(dig_p)
        dig_q = int(
            input("Enter the number of digits in second prime
number(q): "))
        q = generate_prime_no(dig_q)

        print(f"p = {p}")
        print(f"q = {q}")

        gen_keys(p, q)

    elif gen_r == 'n':
        p = int(input("Enter first large prime number(p):\n"))
        if not is_prime(p):
            print(f"Entered number is not prime!")
            exit()

        q = int(input("Enter second large prime number(q):\n"))
        if not is_prime(q):
            print(f"Entered number is not prime!")
            exit()

        gen_keys(p, q)

    else:
        print("Invaild choice!")
        exit()

elif (ch == 2):
    message = input("Enter the message to be encrypted:\n")
```

```
    print("Enter the Public Key (e, n):")
    e = int(input("Enter the value of 'e':\n"))
    n = int(input("Enter the value of 'n':\n"))

    encrypted_message = encrypt(message, e, n)
    print(f"Encrypted message is:\n{' '.join(map(str,
encrypted_message))}")

elif (ch == 3):
    encrypted_message = list(map(int, input(
        "Enter the list of encrypted values separated by
space:\n").split()))
    print("Enter the Private Key (d, n):")
    d = int(input("Enter the value of 'd':\n"))
    n = int(input("Enter the value of 'n':\n"))

    decrypted_message = decrypt(encrypted_message, d, n)

    ans = ""
    for a in decrypted_message:
        if (a < 'A' or a > 'Z'):
            ans += " "
        else:
            ans += a

    print(f"Decrypted message is:\n{ans}")

else:
    print("Invalid input!")
```

Output:

```
PS D:\Final_BTech_Labs\CNS> python -u "d:\Final_BTech_Labs\CNS\Assignment 8\RSA_b.py"
What do you want to perform?
1. Generate Public & Private Keys
2. Encryption
3. Decryption
2
Enter the message to be encrypted:
virat kohli is chase master
Enter the Public Key (e, n):
Enter the value of 'e':
5
Enter the value of 'n':
949
Encrypted message is:
554 502 153 0 158 418 355 690 674 670 502 418 502 109 418 32 674 0 109 75 418 194 0 109 158 75 153
PS D:\Final_BTech_Labs\CNS> python -u "d:\Final_BTech_Labs\CNS\Assignment 8\RSA_b.py"
What do you want to perform?
1. Generate Public & Private Keys
2. Encryption
3. Decryption
3
Enter the list of encrypted values separated by space:
554 502 153 0 158 418 355 690 674 670 502 418 502 109 418 32 674 0 109 75 418 194 0 109 158 75 153
Enter the Private Key (d, n):
Enter the value of 'd':
173
Enter the value of 'n':
949
Decrypted message is:
VIRAT KOHLI IS CHASE MASTER
```