**Cryptography and Network Security Lab**

**Name: Rushikesh R. Ware**

**PRN: 2020BTECS00012**

**Batch: B2**

**RSA ALGORITHM**

**Aim:**

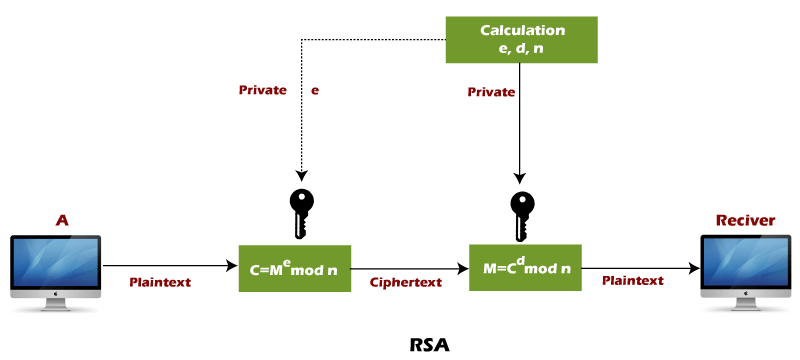
Implementation of RSA algorithm.

**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 < φ (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 = me mod 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:  
  De mod {(p - 1) x (q - 1)} = 1  
  Or  
  De mod φ (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 = cd mod 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 = me mod n**

C = 97 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:

**De 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 = cd mod n**

m = 3743 mod 77

m = 9

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

**Code (RSA for Numerical Message):**

# 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}")

        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"))

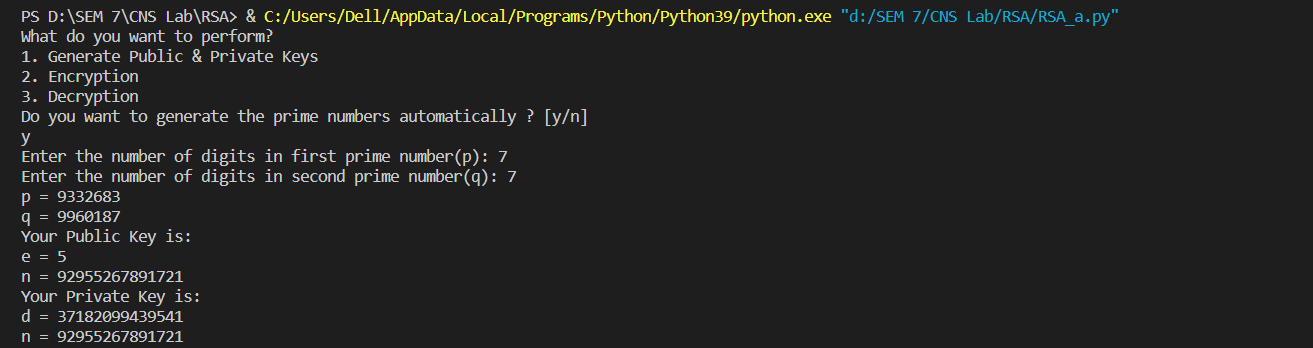
    M = decrypt(C, d, n)

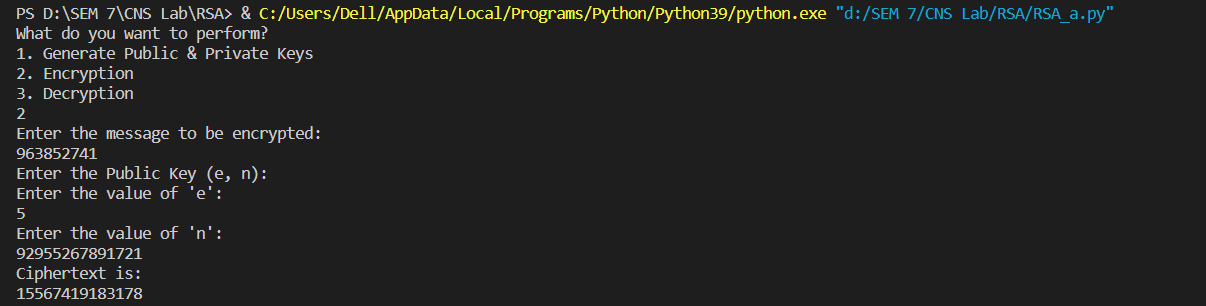
    print(f"Decrypted message is:\n{str(M)}")

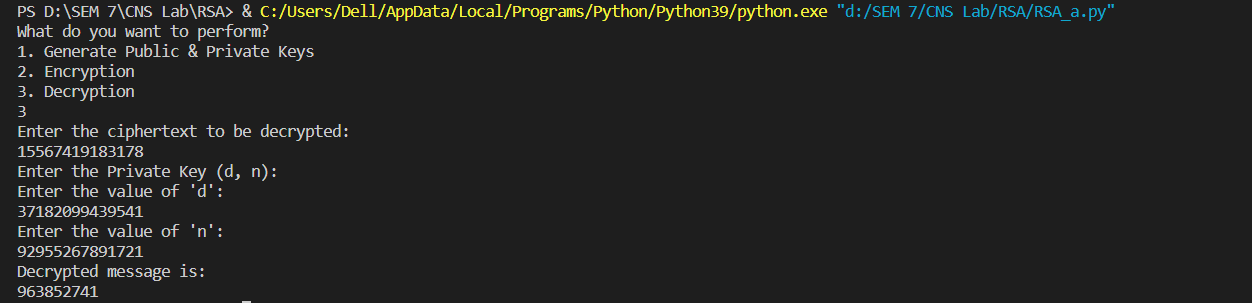
else:

    print("Invalid input!")

**Output:**







**Code (RSA for Text Message):**

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):

            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)

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

else:

    print("Invalid input!")

**Output Screenshot:**

