**Cryptography & Network Security**

**PRN - 2019BTECS00026**

**Name - Niraja Vasudev Kulkarni**

**Batch - B1**

**Assignment - 12**

**Title:** RSA Algorithm

**Aim**: To Demonstrate RSA Algorithm

**Theory:**

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission.

An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the prime numbers.

**Code:**

from math import sqrt

import random

from random import randint as rand

def gcd(a, b):

    if b == 0:

        return a

    else:

        return gcd(b, a % b)

def mod\_inverse(a, m):

    for x in range(1, m):

        if (a \* x) % m == 1:

            return x

    return -1

def isprime(n):

    if n < 2:

        return False

    elif n == 2:

        return True

    else:

        for i in range(2, int(sqrt(n)) + 1, 2):

            if n % i == 0:

                return False

    return True

p = rand(1, 1000)

q = rand(1, 1000)

def generate\_keypair(p, q, keysize):

    nMin = 1 << (keysize - 1)

    nMax = (1 << keysize) - 1

    primes = [2]

    start = 1 << (keysize // 2 - 1)

    stop = 1 << (keysize // 2 + 1)

    if start >= stop:

        return []

    for i in range(3, stop + 1, 2):

        for p in primes:

            if i % p == 0:

                break

        else:

            primes.append(i)

    while (primes and primes[0] < start):

        del primes[0]

    while primes:

        p = random.choice(primes)

        primes.remove(p)

        q\_values = [q for q in primes if nMin <= p \* q <= nMax]

        if q\_values:

            q = random.choice(q\_values)

            break

    print(p, q)

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = random.randrange(1, phi)

    g = gcd(e, phi)

    while True:

        e = random.randrange(1, phi)

        g = gcd(e, phi)

        d = mod\_inverse(e, phi)

        if g == 1 and e != d:

            break

    return ((e, n), (d, n))

def encrypt(msg\_plaintext, package):

    e, n = package

    msg\_ciphertext = [pow(ord(c), e, n) for c in msg\_plaintext]

    return msg\_ciphertext

def decrypt(msg\_ciphertext, package):

    d, n = package

    msg\_plaintext = [chr(pow(c, d, n)) for c in msg\_ciphertext]

    return (''.join(msg\_plaintext))

if \_\_name\_\_ == "\_\_main\_\_":

    bit\_length = int(input("Enter bit\_length: "))

    public, private = generate\_keypair(p, q, 2\*\*bit\_length)

    print("Public Key: ", public)

    print("Private Key: ", private)

    msg = input("Write msg: ")

    print([ord(c) for c in msg])

    encrypted\_msg = encrypt(msg, public)

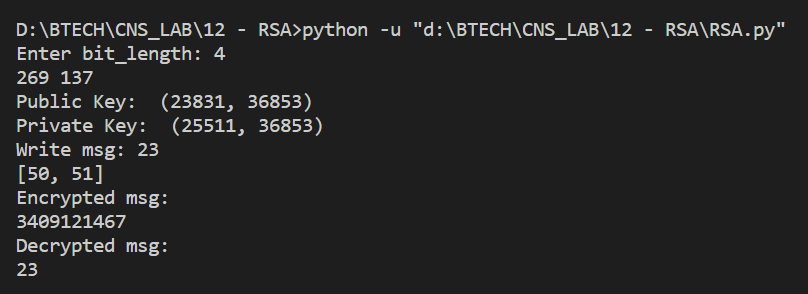
    print("Encrypted msg: ")

    print(''.join(map(lambda x: str(x), encrypted\_msg)))

    print("Decrypted msg: ")

    print(decrypt(encrypted\_msg, private))

**Output:**



**Conclusion:**

The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem".