**Final Year B. Tech., Sem VII 2022-23**

**Cryptography And Network Security**

**PRN/ Roll No: 2020BTECS00206**

**Full Name: SAYALI YOGESH DESAI**

**Batch: B4**

**Assignment No. 10**

1. **Aim:**

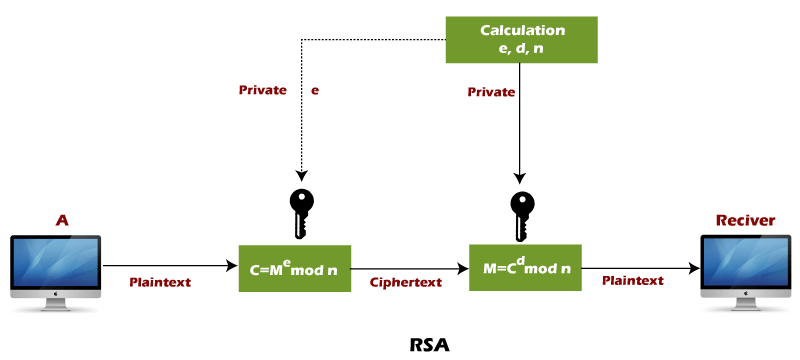
Implementation of RSA Algorithm

1. **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

1. **Code:**

#include <bits/stdc++.h>

using namespace std;

void file()

{

#ifndef ONLINE\_JUDGE

freopen("input.txt", "r", stdin);

freopen("output.txt", "w", stdout);

#endif

}

// Function for extended Euclidean Algorithm

int ansS, ansT;

int findGcdExtended(int r1, int r2, int s1, int s2, int t1, int t2)

{

// Base Case

if (r2 == 0)

{

ansS = s1;

ansT = t1;

return r1;

}

int q = r1 / r2;

int r = r1 % r2;

int s = s1 - q \* s2;

int t = t1 - q \* t2;

cout << q << " " << r1 << " " << r2 << " " << r << " " << s1 << " " << s2 << " " << s << " " << t1 << " " << t2 << " " << t << endl;

return findGcdExtended(r2, r, s2, s, t2, t);

}

int modInverse(int A, int M)

{

int x, y;

int g = findGcdExtended(A, M, 1, 0, 0, 1);

if (g != 1) {

cout << "Inverse doesn't exist";

return 0;

}

else {

// m is added to handle negative x

int res = (ansS % M + M) % M;

cout << "Inverse is: " << res << endl;

return res;

}

}

long long powM(long long a, long long b, long long n)

{

if (b == 1)

return a % n;

long long x = powM(a, b / 2, n);

x = (x \* x) % n;

if (b % 2)

x = (x \* a) % n;

return x;

}

int findGCD(int num1, int num2)

{

if (num1 == 0)

return num2;

return findGCD(num2 % num1, num1);

}

// Code to demonstrate RSA algorithm

int main()

{

file();

// Two random prime numbers

long long p, q, e, msg;

//17 31 7 2

cin >> p >> q >> e >> msg;

cout<<"Two prime Numbers are: "<<p<<" "<<q<<endl;

// First part of public key:

long long n = p \* q;

cout << "Product of two prime number n is " << n << endl;

// Finding other part of public key.

// e stands for encrypt

cout << "Taken e is " << e << endl;

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

cout << "phi is " << phi << endl;

while (e < phi) {

// e must be co-prime to phi and

// smaller than phi.

if (findGCD(e, phi) == 1)

break;

else

e++;

}

cout << "Final e value is " << e << endl;

// Private key (d stands for decrypt)

long long d = modInverse(e, phi);

cout << "d is " << d << endl;

cout << "\nSo now our public key is " << "<" << e << "," << n << ">" << endl;

cout << "\nSo now our private key is " << "<" << d << "," << n << ">" << endl << endl;

// Message to be encrypted

cout << "Message data is " << msg << endl;

// Encryption c = (msg ^ e) % n

long long c = powM(msg, e, n);

cout << "Encripted Message is " << c << endl;

// Decryption m = (c ^ d) % n

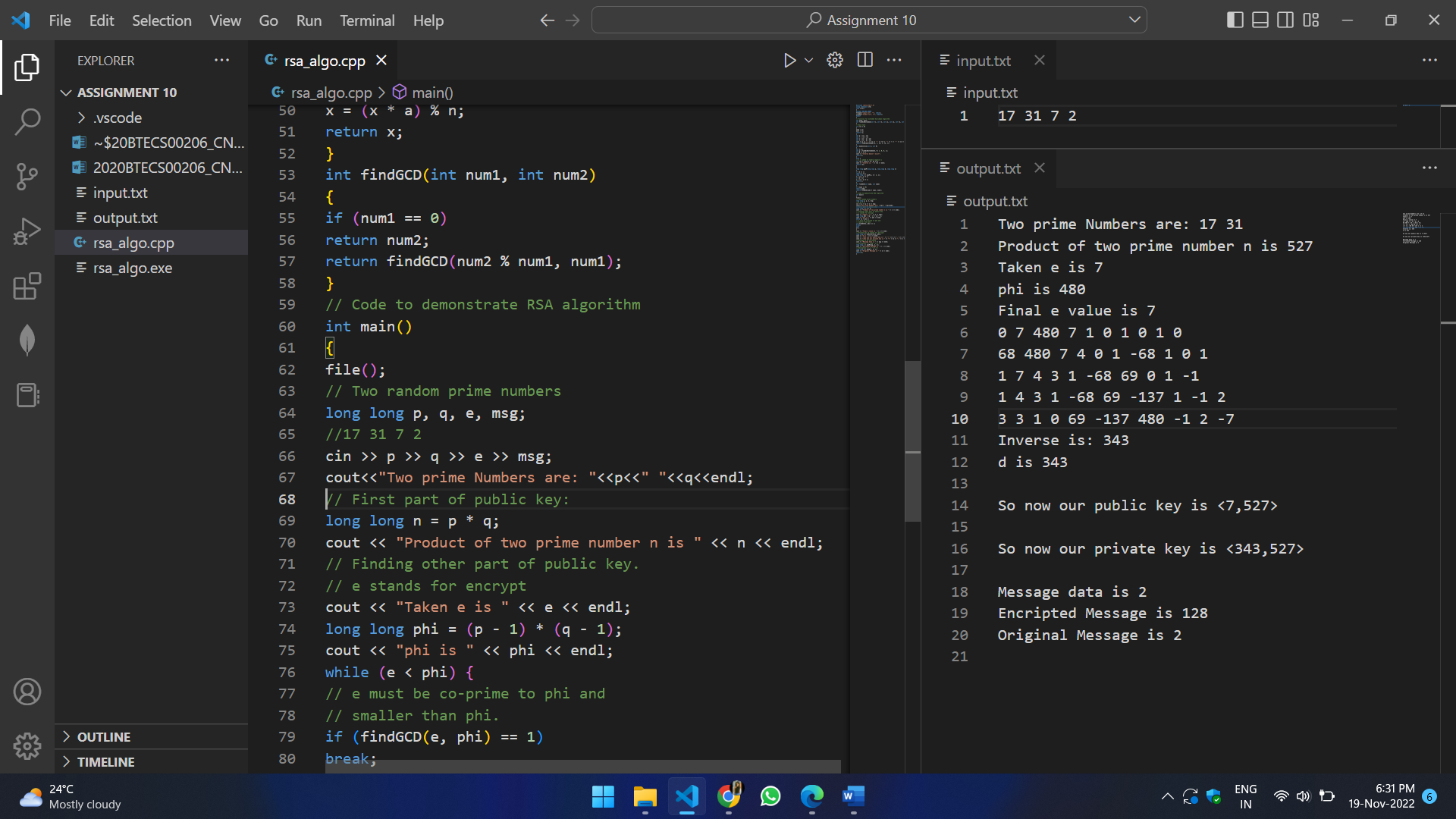
long long m = powM(c, d, n);

cout << "Original Message is " << m << endl;

return 0;

}

1. **Output:**



1. **Conclusion:**

Successfully implemented RSA Algorithm.