|  |  |
| --- | --- |
| **Academic Year: 2025-26** | **Programme: BTECH-Cyber (CSE)** |
| **Year: 2nd** | **Semester: IV** |
| **Student Name: Jal Bafana** | **Batch: K1** |
| **Roll No: K005** | **Date of experiment: 13.02.2025** |
| **Faculty: Rejo Mathew** | **Signature with Date: 13.02.2025** |

**Experiment 6: RSA algorithm**

**Aim:** Write a program to implement RSA algorithm.

**Learning Outcomes:**

After completion of this experiment, student should be able to

1. Differentiate between symmetric and asymmetric key cryptography.
2. Describe working of RSA algorithm.
3. Understand application of RSA along with its advantage and limitations.

**Theory:**

Algorithm for RSA is given below.

Choose two large prime numbers *p, q*

Let *n = p \* q*; then *φ* (*n*) = (*p*–1) \* (*q*–1) [where *φ* is Euler's totient function]

Choose *e* <*φ* (*n*) such that *e* is relatively prime to *φ* (*n*).

Choose an integer *e* such that 1 < *e* < *φ*(*pq*), and *e* and *φ*(*pq*) share no divisors other than 1 (i.e., *e* and *φ*(*pq*) are coprime.).

* + - *e* is released as the public key exponent.

Determine *d* = e-1 mod (*φ)*

* + - *d* is kept as the private key exponent.

Compute *d*such that *e d* mod *φ* (*n*) = 1

Public key: (*e*, *n*); private key: (*d, n)*

Encipher: *c* = *me*mod *n*

Decipher: *m* = *cd*mod *n*

Example:

Take *p* = 7, *q* = 11, so *n* = 77 and*φ* (*n*) = 60

Alice chooses *e* = 17, making *d* = 53

Bob wants to send Alice secret message HELLO (07 04 11 11 14)

0717mod 77 = 28

0417mod 77 = 16

1117mod 77 = 44

1117mod 77 = 44

1417mod 77 = 42

Bob sends 28 16 44 44 42

Alice receives 28 16 44 44 42

Alice uses private key, *d* = 53, to decrypt message:

2853mod 77 = 07

1653mod 77 = 04

4453mod 77 = 11

4453mod 77 = 11

4253mod 77 = 14

Alice translates message to letters to read HELLO

**Algorithm:**

1.Accept two integer numbers from user.

2. Validate the input provided by user is a prime number. If not, ask user to re-enter prime number.

3. Generate public key and private key.

4. Display public key and private key.

5. Ask user to input message for encryption.

6. Display the cipher text.

7. Ask user to input cipher text for decryption.

8. Display the plain text.

**Code: *type or copy your completed working code here***

*Note: Code should have proper comments*

#include <iostream>

#include <string>

#include <sstream>

using namespace std;

int gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

int modInverse(int e, int phi) {

for (int d = 1; d < phi; d++) {

if ((e \* d) % phi == 1) {

return d;

}

}

return -1; // No inverse found

}

int main() {

int p, q;

cout<< "Enter the first prime number (p): ";

cin>> p;

cout<< "Enter the second prime number (q): ";

cin>> q;

// Calculate n and phi(n)

int n = p \* q;

int phi\_n = (p - 1) \* (q - 1);

// Display n and phi(n)

cout<< "n = " << n <<endl;

cout<< "phi(n) = " <<phi\_n<<endl;

int e;

cout<< "Enter a value for e (1 < e < " <<phi\_n<< "): ";

cin>> e;

// Ensure e is coprime with phi(n)

while (gcd(e, phi\_n) != 1) {

cout<< "e must be coprime with phi(n). Enter a valid value for e: ";

cin>> e;

}

// Find d such that e \* d ≡ 1 (mod phi(n))

int d = modInverse(e, phi\_n);

if (d == -1) {

cout<< "Error: No modular inverse found for e.\n";

return -1;

}

// Display Public and Private Key

cout<< "Public Key (n, e): (" << n << ", " << e << ")\n";

cout<< "Private Key (n, d): (" << n << ", " << d << ")\n";

// Encrypt the message

string message;

cin.ignore(); // Clear the newline left by the previous input

cout<< "Enter the message to encrypt (only space-separated numbers corresponding to letters): ";

getline(cin, message); // Read the input message

// Convert message from string to a list of numbers

stringstream ss(message);

int m;

cout<< "Encrypted Cipher Text: ";

while (ss >> m) {

if (m >= 0 && m <= 25) { // Only allow values between 0 and 25 (A=0, B=1, ..., Z=25)

int cipher = 1;

for (int j = 0; j < e; j++) { // Encrypt each number using c = m^e mod n

cipher = (cipher \* m) % n;

}

cout<< cipher << " "; // Output the encrypted number

}

else {

cout<< "(Invalid character) "; // Handle invalid values

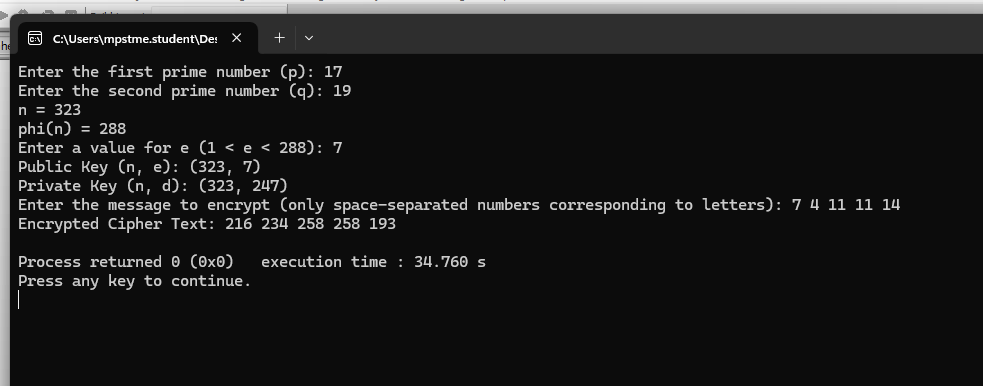
}

}

cout<<endl;

return 0;

}

**

#include <iostream>

#include <sstream>

using namespace std;

int modInverse(int e, int phi) {

for (int d = 1; d < phi; d++) {

if ((e \* d) % phi == 1) {

return d;

}

}

return -1; // No modular inverse found

}

// Efficient modular exponentiation

int modExp(int base, int exp, int mod) {

int result = 1;

base = base % mod; // To handle base larger than mod

while (exp > 0) {

if (exp % 2 == 1) {

result = (result \* base) % mod;

}

exp = exp >> 1; // Exp = Exp / 2

base = (base \* base) % mod;

}

return result;

}

int main() {

int p, q;

// Get n and d (Private Key)

cout<< "Enter the first prime number (p): ";

cin>> p;

cout<< "Enter the second prime number (q): ";

cin>> q;

// Calculate n and phi(n)

int n = p \* q;

int phi\_n = (p - 1) \* (q - 1);

// Display n and phi(n)

cout<< "n = " << n <<endl;

cout<< "phi(n) = " <<phi\_n<<endl;

int e;

cout<< "Enter the value for e (public key exponent): ";

cin>> e;

// Calculate d (private key exponent)

int d = modInverse(e, phi\_n);

if (d == -1) {

cout<< "Error: No modular inverse found for e.\n";

return -1;

}

// Display the Private Key (n, d)

cout<< "Private Key (n, d): (" << n << ", " << d << ")\n";

// Decrypt the cipher

string message;

cin.ignore(); // Clear the newline left by the previous input

cout<< "Enter the cipher text to decrypt (space-separated numbers): ";

getline(cin, message); // Read the input message

// Convert cipher text from string to a list of numbers

stringstream ss(message);

int cipher;

cout<< "Decrypted Plain Text: ";

while (ss >> cipher) {

if (cipher >= 0 && cipher < n) { // Ensure valid cipher text

int decrypted = modExp(cipher, d, n); // Decrypt using m = c^d mod n

cout<< decrypted << " "; // Output the decrypted number (converted to letter below)

} else {

cout<< "(Invalid cipher) "; // Handle invalid cipher values

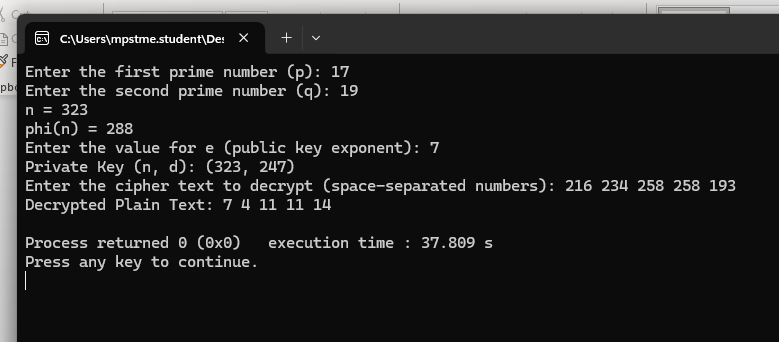
}

}

cout<<endl;

return 0;

}

**

**Questions:**

1. Why is it computationally difficult to derive the private key from the public key in RSA?

The security of RSA relies on the fact that factoring a large number nnn (which is the product of two large prime numbers ppp and qqq) is computationally difficult. Given nnn and eee (public key), finding ddd (private key) requires computing ϕ(n)\phi(n)ϕ(n),

which in turn requires knowing ppp and qqq.

Since integer factorization of large numbers is an NP-hard problem, it makes deriving the private key infeasible with current computational power.

1. What happens if two different users select the same prime numbers for their RSA key pairs

If two users choose the same prime numbers ppp and qqq, they will have the same modulus nnn but potentially different public exponents eee. If an attacker discovers this, they can compute ϕ(n)\phi(n)ϕ(n) and derive the private keys of both users, breaking their encryption.

1. How does RSA handle message sizes larger than the modulus **n**?

RSA cannot directly encrypt messages larger than nnn. Instead, large messages are divided into smaller blocks, each smaller than nnn, and then encrypted separately. Alternatively, hybrid encryption is used, where RSA encrypts a symmetric key, and the symmetric key encrypts the actual message.

1. How does the **Chinese Remainder Theorem (CRT)** optimize RSA decryption?

The Chinese Remainder Theorem (CRT) allows RSA decryption to be performed using modular arithmetic separately with ppp and qqq, rather than using nnn directly. This speeds up decryption significantly, as computations with smaller numbers are faster.

1. Why does RSA encryption work as a one-way function, and how does the difficulty of **integer factorization** contribute to its security?

RSA encryption is a one-way function because computing -

c=memod  nc = m^e \mod nc=memodn is easy,

but reversing it to find mmm without the private key is computationally infeasible. The difficulty of integer factorization ensures that an attacker cannot efficiently derive ppp and qqq from nnn, making it nearly impossible to compute ddd and decrypt the message.

**Conclusion:** *In this lab, I successfully implemented the RSA algorithm, generating key pairs, encrypting messages, and decrypting cipher text. This experiment deepened my understanding of asymmetric encryption and its real-world security implications.*