**Cryptography, Network and Security**

Assignment 7

Implementation of RSA Algorithm

Code:

#include <iostream>

#include <cmath>

#include <cstdlib>

#include <ctime>

using namespace std;

*// Function to compute the greatest common divisor (GCD)*

int gcd(int a, int b)

{

    while (b != 0)

    {

        int temp = b;

        b = a % b;

        a = temp;

    }

    return a;

}

*// Function to compute the modular inverse using the Extended Euclidean Algorithm*

int modInverse(int e, int phi)

{

    int t = 0, newT = 1;

    int r = phi, newR = e;

    while (newR != 0)

    {

        int quotient = r / newR;

        int temp = newT;

        newT = t - quotient \* newT;

        t = temp;

        temp = newR;

        newR = r - quotient \* newR;

        r = temp;

    }

    if (r > 1)

    {

        return -1; *// No modular inverse exists*

    }

    if (t < 0)

    {

        t += phi;

    }

    return t;

}

*// Function to perform modular exponentiation: (base^exp) % mod*

long long modPow(long long base, long long exp, long long mod)

{

    long long result = 1;

    base = base % mod;

    while (exp > 0)

    {

        if (exp % 2 == 1)

        {

            result = (result \* base) % mod;

        }

        exp = exp >> 1;

        base = (base \* base) % mod;

    }

    return result;

}

*// Function to generate a large prime number (simple and naive approach)*

bool isPrime(int n)

{

    if (n <= 1)

        return false;

    if (n == 2 || n == 3)

        return true;

    if (n % 2 == 0 || n % 3 == 0)

        return false;

    for (int i = 5; i \* i <= n; i += 6)

    {

        if (n % i == 0 || n % (i + 2) == 0)

            return false;

    }

    return true;

}

*// RSA Encryption Function*

long long encrypt(int message, int e, int n)

{

    return modPow(message, e, n);

}

*// RSA Decryption Function*

long long decrypt(long long ciphertext, int d, int n)

{

    return modPow(ciphertext, d, n);

}

int main()

{

    srand(time(0)); *// Seed for random number generation*

*// Step 1: Select two distinct large prime numbers p and q*

    int p = 61; *// Example prime (use a prime generator or larger primes for real-world applications)*

    int q = 53; *// Example prime*

*// Step 2: Compute n = p \* q*

    int n = p \* q;

*// Step 3: Compute φ(n) = (p-1) \* (q-1)*

    int phi = (p - 1) \* (q - 1);

*// Step 4: Choose an integer e such that 1 < e < φ(n) and gcd(e, φ(n)) = 1*

    int e = 17; *// Example public exponent (small and commonly used value)*

*// Step 5: Compute the private key d, where d is the modular inverse of e modulo φ(n)*

    int d = modInverse(e, phi);

    if (d == -1)

    {

        cout << "Error: No modular inverse found for e and φ(n)." << endl;

        return -1;

    }

*// Public key (e, n) and Private key (d, n) are now generated*

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

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

*// Step 6: Encrypt the message using the public key*

    int message;

    cout << "Enter a message (integer) to encrypt: ";

    cin >> message;

    long long ciphertext = encrypt(message, e, n);

    cout << "Encrypted message (ciphertext): " << ciphertext << endl;

*// Step 7: Decrypt the ciphertext using the private key*

    long long decryptedMessage = decrypt(ciphertext, d, n);

    cout << "Decrypted message: " << decryptedMessage << endl;

    return 0;

}