***SAVEETHA SCHOOL OF ENGINEERING***

***SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCE***

**EXP NO 8: Encrypt and decrypt a text using RSA encryption algorithm**

**AIM**

To write a c program to Encrypt and decrypt a text using RSA encryption algorithm

**PROCEDURE**

* Generates an RSA key pair with a specified key length.
* Writes the public and private keys to separate PEM files (**public.pem** and **private.pem**).
* Encrypts the plaintext using the public key.
* Decrypts the ciphertext using the private key.
* Prints the original plaintext and the decrypted message

**PROGRAM**

#include <stdio.h>  
  
// Function to calculate modular exponentiation  
unsigned long long mod\_exp(unsigned long long base, unsigned long long exp, unsigned long long mod) {  
    unsigned 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 encrypt plaintext using RSA public key  
unsigned long long rsa\_encrypt(unsigned long long plaintext, unsigned long long e, unsigned long long n) {  
    return mod\_exp(plaintext, e, n);  
}  
  
// Function to decrypt ciphertext using RSA private key  
unsigned long long rsa\_decrypt(unsigned long long ciphertext, unsigned long long d, unsigned long long n) {  
    return mod\_exp(ciphertext, d, n);  
}  
  
int main() {  
    // RSA parameters (usually large prime numbers)  
    unsigned long long p = 61;  
    unsigned long long q = 53;  
    unsigned long long n = p \* q;  
    unsigned long long phi = (p - 1) \* (q - 1);  
    unsigned long long e = 65537; // commonly used value for e  
    unsigned long long d = 0;  
  
    // Find the modular inverse of e mod phi  
    for (unsigned long long i = 1; i < phi; ++i) {  
        if ((e \* i) % phi == 1) {  
            d = i;  
            break;  
        }  
    }  
  
    // Message to be encrypted  
    unsigned long long plaintext = 1234;  
  
    // Encrypt plaintext  
    unsigned long long encrypted = rsa\_encrypt(plaintext, e, n);  
  
    // Decrypt ciphertext  
    unsigned long long decrypted = rsa\_decrypt(encrypted, d, n);  
  
    // Output results  
    printf("Plaintext: %llu\n", plaintext);  
    printf("Encrypted: %llu\n", encrypted);  
    printf("Decrypted: %llu\n", decrypted);  
  
    return 0;  
}

**(OR)**

#include <stdio.h>

// Function to calculate modular exponentiation

unsigned long long mod\_exp(unsigned long long base, unsigned long long exp, unsigned long long mod) {

unsigned 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 encrypt plaintext using RSA public key

unsigned long long rsa\_encrypt(unsigned long long plaintext, unsigned long long e, unsigned long long n) {

return mod\_exp(plaintext, e, n);

}

// Function to decrypt ciphertext using RSA private key

unsigned long long rsa\_decrypt(unsigned long long ciphertext, unsigned long long d, unsigned long long n) {

return mod\_exp(ciphertext, d, n);

}

int main() {

unsigned long long p, q, n, phi, e, d;

unsigned long long plaintext, encrypted, decrypted;

// Get user input for RSA parameters

printf("Enter value for p (prime number): ");

scanf("%llu", &p);

printf("Enter value for q (prime number): ");

scanf("%llu", &q);

printf("Enter value for e (public exponent): ");

scanf("%llu", &e);

// Calculate n, phi, and d

n = p \* q;

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

// Find the modular inverse of e mod phi

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

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

break;

}

}

// Get user input for plaintext

printf("Enter plaintext to be encrypted: ");

scanf("%llu", &plaintext);

// Encrypt plaintext

encrypted = rsa\_encrypt(plaintext, e, n);

// Decrypt ciphertext

decrypted = rsa\_decrypt(encrypted, d, n);

// Output results

printf("Plaintext: %llu\n", plaintext);

printf("Encrypted: %llu\n", encrypted);

printf("Decrypted: %llu\n", decrypted);

return 0;

}

**OUTPUT**

