RSA

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <time.h>

// Function to find the Greatest Common Divisor

int gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

// Function to find modular inverse using the Extended Euclidean Algorithm

int mod\_inverse(int e, int phi) {

int d = 0, x1 = 0, x2 = 1, y1 = 1, temp\_phi = phi;

while (e > 0) {

int temp1 = temp\_phi / e;

int temp2 = temp\_phi - temp1 \* e;

temp\_phi = e;

e = temp2;

int x = x2 - temp1 \* x1;

int y = d - temp1 \* y1;

x2 = x1;

x1 = x;

d = y1;

y1 = y;

}

if (temp\_phi == 1) {

return d + phi;

}

return -1;

}

// Function to check if a number is prime

int is\_prime(int num) {

if (num < 2) return 0;

for (int i = 2; i <= sqrt(num); i++) {

if (num % i == 0) return 0;

}

return 1;

}

// Function to generate a random prime number within a range

int generate\_prime(int lower, int upper) {

int prime;

do {

prime = rand() % (upper - lower + 1) + lower;

} while (!is\_prime(prime));

return prime;

}

// Function to perform modular exponentiation

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

long long result = 1;

while (exp > 0) {

if (exp % 2 == 1)

result = (result \* base) % mod;

exp = exp >> 1;

base = (base \* base) % mod;

}

return result;

}

int main() {

srand(time(NULL));

// Step 1: Generate two distinct prime numbers

int p = generate\_prime(50, 100);

int q = generate\_prime(50, 100);

while (p == q) {

q = generate\_prime(50, 100);

}

// Step 2: Calculate n and φ(n)

int n = p \* q;

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

// Step 3: Choose e such that 1 < e < φ(n) and gcd(e, φ(n)) = 1

int e;

do {

e = rand() % (phi - 2) + 2;

} while (gcd(e, phi) != 1);

// Step 4: Calculate d, the modular inverse of e

int d = mod\_inverse(e, phi);

printf("Public Key: (e=%d, n=%d)\n", e, n);

printf("Private Key: (d=%d, n=%d)\n", d, n);

// Step 5: Encryption

char message[128];

printf("Enter a message to encrypt: ");

fgets(message, sizeof(message), stdin);

long long encrypted\_message[128];

int len = strlen(message);

printf("Encrypted message: ");

for (int i = 0; i < len - 1; i++) { // Ignore newline character from fgets

encrypted\_message[i] = mod\_exp(message[i], e, n);

printf("%lld ", encrypted\_message[i]);

}

printf("\n");

// Step 6: Decryption

printf("Decrypted message: ");

for (int i = 0; i < len - 1; i++) {

char decrypted\_char = (char)mod\_exp(encrypted\_message[i], d, n);

printf("%c", decrypted\_char);

}

printf("\n");

return 0;

}