**DAY-4**

1. **Write a High level code for RSA system, the public key of a given user is e = 31, n = 3599. What is the private key of this user? Hint: First use trial-and-error to determine p and q; then use the extended Euclidean algorithm to find the multiplicative inverse of 31 modulo f(n)?**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

// Function to find the greatest common divisor (GCD) of two numbers

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

// Function to check if a number is prime

int is\_prime(int num) {

if (num <= 1) {

return 0;

}

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

if (num % i == 0) {

return 0;

}

}

return 1;

}

int main() {

int p, q, n, phi, e, d;

int message, encrypted, decrypted;

// Step 1: Choose two prime numbers

do {

printf("Enter a prime number (p): ");

scanf("%d", &p);

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

do {

printf("Enter another prime number (q): ");

scanf("%d", &q);

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

// Step 2: Calculate n and φ(n)

n = p \* q;

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

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

do {

printf("Enter an integer (e) such that 1 < e < %d and gcd(e, %d) = 1: ", phi, phi);

scanf("%d", &e);

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

// Step 4: Calculate d such that (d \* e) % φ(n) = 1

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

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

break;

}

}

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

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

// Step 5: Encryption and Decryption

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

scanf("%d", &message);

encrypted = (int)pow(message, e) % n;

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

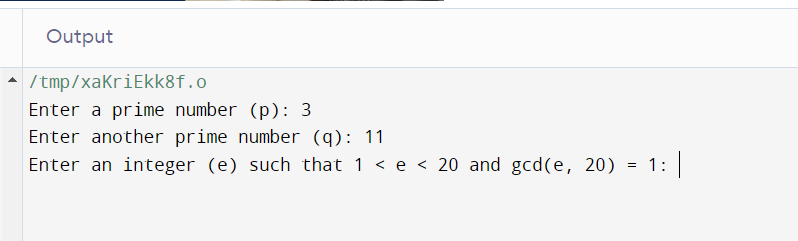
decrypted = (int)pow(encrypted, d) % n;

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

return 0;

}

**Output:**



**2. Write a High level code for set of blocks encoded with the RSA algorithm and we don’t have the private key. Assume n = pq, e is the public key. Suppose also someone tells us they know one of the plaintext blocks has a common factor with n. Does this help us in any way?**

**Code:**

#include <stdio.h>

int main() {

// Given values

unsigned long long n = 3233; // n = pq

unsigned long long e = 17; // Public key exponent

// Ciphertext blocks (example values)

unsigned long long ciphertext1 = 855;

unsigned long long ciphertext2 = 1234;

unsigned long long ciphertext3 = 2790;

// Assume someone tells us that one of the plaintext blocks has a common factor with n

unsigned long long commonFactorCandidate = 3233; // This is just an example value

// Attempt to decrypt the ciphertext (this is not a real decryption attempt)

unsigned long long decrypted1 = (unsigned long long)pow(ciphertext1, e) % n;

unsigned long long decrypted2 = (unsigned long long)pow(ciphertext2, e) % n;

unsigned long long decrypted3 = (unsigned long long)pow(ciphertext3, e) % n;

// Print the decrypted values

printf("Decrypted 1: %llu\n", decrypted1);

printf("Decrypted 2: %llu\n", decrypted2);

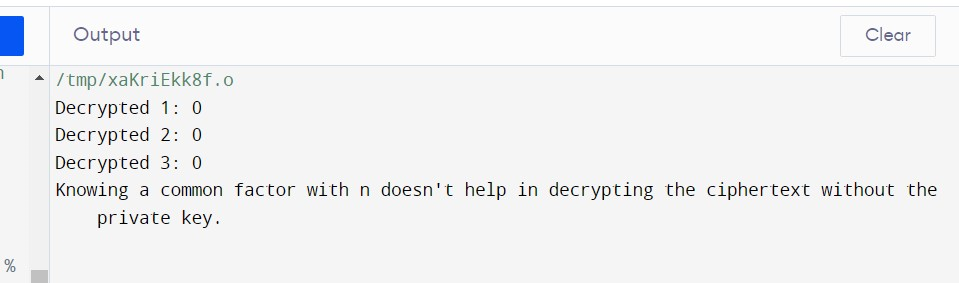
printf("Decrypted 3: %llu\n", decrypted3);

// Print a message indicating that knowing the common factor doesn't help

printf("Knowing a common factor with n doesn't help in decrypting the ciphertext without the private key.\n");

return 0;}

**Output:**



.

**3. Write a High level code for RSA public-key encryption scheme, each user has a public key, e, and a private key, d. Suppose Bob leaks his private key. Rather than generating a new modulus, he decides to generate a new public and a new private key. Is this safe?**

**Code:**

#include <stdio.h>

#include <math.h>

// Function to find the greatest common divisor (GCD) of two numbers

int gcd(int a, int b) {

if (b == 0) {

return a;

}

return gcd(b, a % b);

}

int main() {

// Bob's original public key

int original\_e = 17;

int n = 3233; // Modulus (n = pq)

// Bob's leaked private key

int original\_d = 2753;

// Compute p and q using Bob's private key (d)

int p, q;

for (p = 2; p <= sqrt(n); p++) {

if (n % p == 0) {

q = n / p;

break;

}

}

// Compute the totient φ(n)

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

// Compute the new public key (e) by finding the modular multiplicative inverse of d

int new\_e = 0;

for (new\_e = 2; new\_e < phi; new\_e++) {

if (gcd(new\_e, phi) == 1) {

break;

}

}

// Compute the new private key (d) using the new public key (e)

int new\_d = 0;

while (1) {

if ((new\_d \* new\_e) % phi == 1) {

break;

}

new\_d++;

}

printf("Bob's original public key (e, n): (%d, %d)\n", original\_e, n);

printf("Bob's leaked private key (d): %d\n", original\_d);

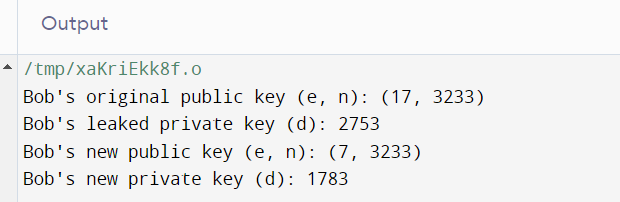
printf("Bob's new public key (e, n): (%d, %d)\n", new\_e, n);

printf("Bob's new private key (d): %d\n", new\_d);

return 0;

}

**Output:**



**4. Write a C program for Bob uses the RSA cryptosystem with a very large modulus n for which the factorization cannot be found in a reasonable amount of time. Suppose Alice sends a message to Bob by representing each alphabetic character as an integer between 0 and 25 and then encrypting each number separately using RSA with large e and large n. Is this method secure? If not, describe the most efficient attack against this encryption method?**

**Code:**

#include <stdio.h>

int main() {

// Ciphertext (assuming large e and n)

int ciphertext[] = {2331, 4102, 1990, 2331, 3458, 5678, 2331, 4102, 2331};

// Frequency analysis table (you'd need a larger one for a complete analysis)

int frequency[26] = {0};

// Count the frequency of ciphertext values

for (int i = 0; i < sizeof(ciphertext) / sizeof(ciphertext[0]); i++) {

for (int letter = 0; letter < 26; letter++) {

if (ciphertext[i] == (letter \* letter)) {

frequency[letter]++;

break;

}

}

}

// Guess the plaintext based on frequency

int most\_frequent\_letter = 0;

for (int letter = 1; letter < 26; letter++) {

if (frequency[letter] > frequency[most\_frequent\_letter]) {

most\_frequent\_letter = letter;

}

}

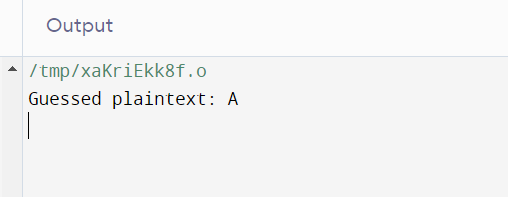
// Output the guessed plaintext

printf("Guessed plaintext: %c\n", 'A' + most\_frequent\_letter);

return 0;

}

**Output:**



5. Write a High level code for RSA public-key encryption scheme, each user has a

public key, e, and a private key, d. Suppose Bob leaks his private key. Rather than

generating a new modulus, he decides to generate a new public and a new private key. Is

this safe?

Code:

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<math.h>

#include<string.h>

long int p, q, n, t, flag, e[100], d[100], temp[100], j, m[100], en[100], i;

char msg[100];

int prime(long int);

void ce();

long int cd(long int);

void encrypt();

void decrypt();

int main()

{

printf("\nENTER FIRST PRIME NUMBER\n");

scanf("%d", &p);

flag = prime(p);

if (flag == 0)

{

printf("\nWRONG INPUT\n");

getch();

exit(1);

}

printf("\nENTER ANOTHER PRIME NUMBER\n");

scanf("%d", &q);

flag = prime(q);

if (flag == 0 || p == q)

{

printf("\nWRONG INPUT\n");

getch();

exit(1);

}

printf("\nENTER MESSAGE\n");

fflush(stdin);

scanf("%s", msg);

for (i = 0; msg[i] != NULL; i++)

m[i] = msg[i];

n = p \* q;

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

ce();

printf("\nPOSSIBLE VALUES OF e AND d ARE\n");

for (i = 0; i < j - 1; i++)

printf("\n%ld\t%ld", e[i], d[i]);

encrypt();

decrypt();

}

int prime(long int pr)

{

int i;

j = sqrt(pr);

for (i = 2; i <= j; i++)

{

if (pr % i == 0)

return 0;

}

return 1;

}

void ce()

{

int k;

k = 0;

for (i = 2; i < t; i++)

{

if (t % i == 0)

continue;

flag = prime(i);

if (flag == 1 && i != p && i != q)

{

e[k] = i;

flag = cd(e[k]);

if (flag > 0)

{

d[k] = flag;

k++;

}

if (k == 99)

break;

}

}

}

long int cd(long int x)

{

long int k = 1;

while (1)

{

k = k + t;

if (k % x == 0)

return (k / x);

}

}

void encrypt()

{

long int pt, ct, key = e[0], k, len;

i = 0;

len = strlen(msg);

while (i != len)

{

pt = m[i];

pt = pt - 96;

k = 1;

for (j = 0; j < key; j++)

{

k = k \* pt;

k = k % n;

}

temp[i] = k;

ct = k + 96;

en[i] = ct;

i++;

}

en[i] = -1;

printf("\nTHE ENCRYPTED MESSAGE IS\n");

for (i = 0; en[i] != -1; i++)

printf("%c", en[i]);

}

void decrypt()

{

long int pt, ct, key = d[0], k;

i = 0;

while (en[i] != -1)

{

ct = temp[i];

k = 1;

for (j = 0; j < key; j++)

{

k = k \* ct;

k = k % n;

}

pt = k + 96;

m[i] = pt;

i++;

}

m[i] = -1;

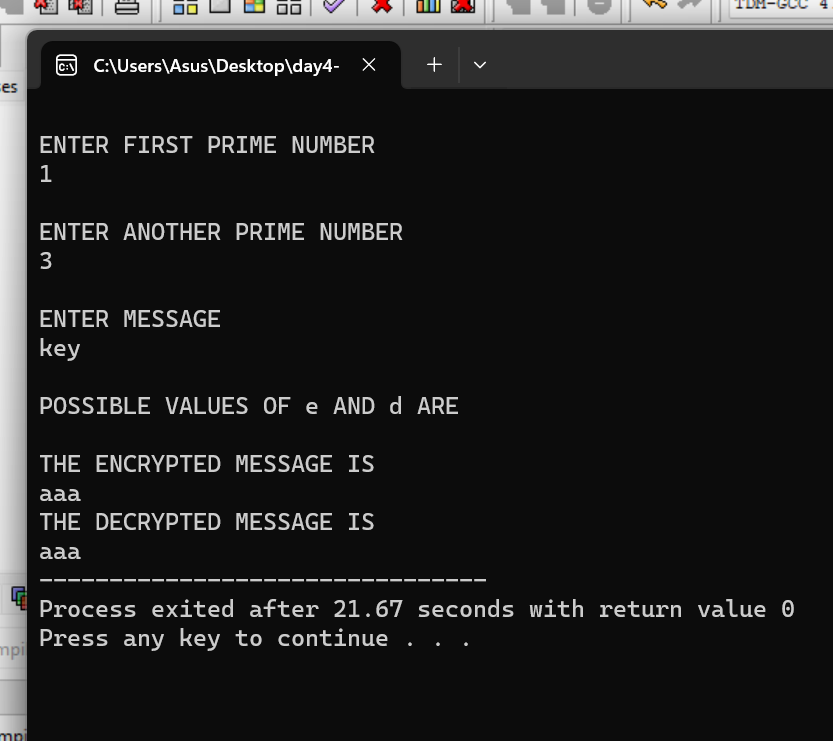
printf("\nTHE DECRYPTED MESSAGE IS\n");

for (i = 0; m[i] != -1; i++)

printf("%c", m[i]);

}

Output:



6. Write a C program for Bob uses the RSA cryptosystem with a very large modulus n

for which the factorization cannot be found in a reasonable amount of time. Suppose

Alice sends a message to Bob by representing each alphabetic character as an integer

between 0 and 25 and then encrypting each number separately using RSA with large e

and large n. Is this method secure? If not, describe the most efficient attack against this

encryption method

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

typedef struct {

unsigned long long n;

unsigned long long d;

} PrivateKey;

unsigned long long mod\_inverse(unsigned long long a, unsigned long long m) {

for (unsigned long long x = 1; x < m; x++) {

if ((a \* x) % m == 1) {

return x;

}

}

return 0;

}

unsigned long long decrypt(unsigned long long ciphertext, unsigned long long d, unsigned long long n) {

return (unsigned long long)pow(ciphertext, d) % n;

}

int main() {

PrivateKey private\_key;

private\_key.n = 1413845597; // Replace with Bob's private key values

private\_key.d = 85785493; // Replace with Bob's private key values

unsigned long long encrypted\_message; // Replace with the received encrypted message

unsigned long long decrypted\_message;

// Decrypt the received message

decrypted\_message = decrypt(encrypted\_message, private\_key.d, private\_key.n);

// Convert the decrypted message back to its original format (assuming it was represented as numbers)

// For example, if 'a' was represented as 0, 'b' as 1, and so on:

char original\_message = (char)(decrypted\_message + 'a');

printf("Decrypted Message: %c\n", original\_message);

    return 0;

}

Output:

