



Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Фізико-технічний інститут

## **ЛАБОРАТОРНА РОБОТА №5**

**з дисципліни**

**«Криптографія»**

**на тему: «Вивчення криптосистеми RSA та алгоритму електронного підпису;  
ознайомлення з методами генерації параметрів для асиметричних криптосистем»**

Виконали:

студенти 3 курсу ФТІ

групи ФБ-71

Безлюдний В.

Мельник Д.

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

## Мета роботи :

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

## Порядок виконання роботи

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.
2. За допомогою цієї функції згенерувати дві пари простих чисел  $p, q$  і  $1 < p, q$  довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб  $pq \leq p_1q_1$ ;  $p, q$  – прості числа для побудови ключів абонента A,  $p_1, q_1$  – абонента B.
3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ  $(d, p, q)$  та відкритий ключ  $(n, e)$ . За допомогою цієї функції побудувати схеми RSA для абонентів A і B – тобто, створити та зберегти для подальшого використання відкриті ключі  $(e, n)$ ,  $(p, q)$  і  $n_1, e_1$  та секретні  $d$  і  $d_1$ .
4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання. За допомогою датчика випадкових чисел вибрати відкрите повідомлення  $M$  і знайти криптограму для абонентів A і B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.
5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа  $0 < k < n$ .

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція `Encrypt()`, яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: `GenerateKeyPair()`, `Encrypt()`, `Decrypt()`, `Sign()`, `Verify()`, `SendKey()`, `ReceiveKey()`.

## Труднощі та етапи розробки програмного коду

Перш за все було розроблено функцію перевірки числа на простоту. Для цього був використаний тест Мілера-Рабіна. Першою перешкодою була необхідність швидко піднести число в степінь по модулю, так як звичайна математична операція піднесення числа в степінь довго працює з великими числами. Тому ми створили функцію за схемою Горнера, яка вирішує це питання.

Далі ми отримали пари простих чисел  $p_1, q_1, p, q$  та написали функції за шифрування, розшифрування, підписання ЦП, його перевірки, та функції відправки та отримання ключів. На цих етапах проблем не було.

### Хід роботи:

#### Первый набор:

**P1: 132429342049902340100398450005047050361950972777759463984106479768426979883499**  
**Q1: 205072871744498393625691140029148606985394066532198480890968940705369434233167**

**P2: 127635264280671181083668878757629973184236670872179834607800230421965366599903**  
**Q2: 123524377319384711779658770206089701942775029872505402400461297877759423035847**

**n1: 271576654774079305790666223986845307070881902113326918400798295413680007147627**  
**85160403627839952628786442114671433027096613990184492372959977889188061811333**

**n2: 157660665442650128804171243632583694869406180037232223054414079299975062229571**  
**96417966785469458581986247801365457680947525040259320817026841521632475722841**

**e1: 65537**

**e2: 65537**

**d1: 906719682486477759129280810390491197951991531522911683709639974662668757556368**  
**3372886335131048854996564083996680465212692152189405767752750367062646187757**

**d2: 122631644439088346520613291963540998667361176641865660656145568249927960880335**  
**26732813290990462274354040226861641690486787300169133409152396460014437737089**

**M1: 78229198338062314354986704307940498777107181242510095522767953296440420516207**  
**84838222107489880022325156103375285955899111325808654130941510752264219638687**

**C1 deciphered:**

**782291983380623143549867043079404987771071812425100955227679532964404205162078483**  
**8222107489880022325156103375285955899111325808654130941510752264219638687**

**M2: 18419319197679411979536271757612642884668552176724223957565578986866417927754**  
**952904107476953117115824253429034043107594228642277749015443791362696741527617**

**C2 deciphered:**

**184193191976794119795362717576126428846685521767242239575655789868664179277549529**  
**04107476953117115824253429034043107594228642277749015443791362696741527617**

**k::122782399877127271642506539380202622071380622016478247705116671629103632329894  
31283180818486266661393986687920825352662531582558003986566301482041182221458**

**Verified S:**

**782291983380623143549867043079404987771071812425100955227679532964404205162078483  
8222107489880022325156103375285955899111325808654130941510752264219638687**

**Key exchange:**

**S:6072127498930097319705438393182909880242600268495284359125050473006364452442146  
300826942532677079318397685209209726482331075472622729581638792441673027422**

**S1:732743993595702455457040140932629211211406828641659152449025975936392785609249  
2344763292483488511425457928958744182505809604595459698074175535016374787542**

**k1:152020142334773997230935105133213812819486339687209394049102938196071965220974  
13522544115833969354253543197571869303230102788496864277381330363545570588705**

```
C:\Users\Дима\Downloads\WinNTL-11_4_3\WinNTL-11_4_3\src\Debug\ConsoleApplication4.exe
13242934204990234010039845000504705036195097277759463984106479768426979883499
PRIME(p)!!
20507287174449839362569114002914860698539406653219848089096984070536943423167
PRIME!!(q)
127635264280671181083668878757629973184236670872179834607800230421965366599903
PRIME!!(p1)
12352437731938471177965877020608970194277502987250540240046129787759423035847
PRIME!!(q1)

n1:: 271576654774079305790666223986845307088190211332691840079829541368000714762785160403627839952628786442114671433027096613990184492372959977889188061811333
n2:: 15766066544265012880417124363258369486940618003723222305441407929997506222957196417966785469458581986247801365457680947525040259320817026841521632475722841
e1:: 65537
e2:: 65537
d1:: 9067196824864777591292808103904911979519915315229116837096399746626687575563683372886335131048854996564083996680465212692152189405767752750367062646187757
d2:: 12263164443908834652061329196354099866736117664186566065614556824992796088033526732813290990462274354040226861641690486787300169133409152396460014437737089
M1: 7822919833806231435498670430794049877710718124251009552276795329644042051620784838222107489880022325156103375285955899111325808654130941510752264219638687
C1 deciphered: 7822919833806231435498670430794049877710718124251009552276795329644042051620784838222107489880022325156103375285955899111325808654130941510752264219638687
7
M2: 18419319197679411979536271757612642884668552176724223957565578986866417927754952904107476953117115824253429034043107594228642277749015443791362696741527617
C2 deciphered: 18419319197679411979536271757612642884668552176724223957565578986866417927754952904107476953117115824253429034043107594228642277749015443791362696741527617
17-----
k::12278239987712727164250653938020262207138062201647824770511667162910363232989431283180818486266661393986687920825352662531582558003986566301482041182221458
Verified
7822919833806231435498670430794049877710718124251009552276795329644042051620784838222107489880022325156103375285955899111325808654130941510752264219638687

n1::271576654774079305790666223986845307088190211332691840079829541368000714762785160403627839952628786442114671433027096613990184492372959977889188061811333
n2::15766066544265012880417124363258369486940618003723222305441407929997506222957196417966785469458581986247801365457680947525040259320817026841521632475722841

Send..
S:6072127498930097319705438393182909880242600268495284359125050473006364452442146300826942532677079318397685209209726482331075472622729581638792441673027422
S1:7327439935957024554570401409326292112114068286416591524490259759363927856092492344763292483488511425457928958744182505809604595459698074175535016374787542
k1:15202014233477399723093510513321381281948633968720939404910293819607196522097413522544115833969354253543197571869303230102788496864277381330363545570588705

Recieve..
Ok
k:: 12278239987712727164250653938020262207138062201647824770511667162910363232989431283180818486266661393986687920825352662531582558003986566301482041182221458
Для продолжения нажмите любую клавишу . . .
```

# RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

## Verify

Clear

Message

012412

Bytes

Signature

83EE470F4101B36DB08408517D9E73BEDE774D558CC80909C00BD0941E230C7B

Modulus

AAA237EEB1E7C2A8CBC700056653411323DF5FD77235D5B0D64B9829378CB085

Public exponent

10001

Verify

Verification

true

## Код:

```
#include <NTL/ZZ.h>
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <iomanip>
#include <cmath>
NTL_CLIENT
```

```
using namespace std;
```

```
ZZ e = pow(2, 16) + (ZZ)1;
long o = 0;
```

```
ZZ pow(int a, int b) {
    ZZ temp;
    temp = a;
    for (int i = 1; i < b; i++)
        temp = temp * a;
    return temp;
}
```

```
ZZ pow(ZZ a, ZZ b) {
    ZZ temp;
    temp = a;
    for (ZZ i = (ZZ)1; i < b; i++)
        temp = temp * a;
    return temp;
}
```

```
ZZ rando(ZZ min) {
```

```

        return RandomBnd(min) + min;
    }

ZZ mod(ZZ k, ZZ m)
{
    if (k < m)
    {
        if (k < 0) { for (;;) { k += m; if (k > 0) return k; } }
        return k;
    }
    else {
        for (;;)
        {
            k = k - m;
            if (k < m) return k;
        }
        return k;
    }
}

```

```

ZZ gcdExtended(ZZ a, ZZ b, ZZ* x, ZZ* y)
{
    if (a == 0)
    {
        *x = 0, *y = 1;
        return b;
    }
    /*Change all data types*/
    ZZ x1, y1;
    ZZ gcd = gcdExtended(b % a, a, &x1, &y1);

    *x = y1 - (b / a) * x1;
    *y = x1;

    return gcd;
}

```

```

ZZ modInverse(ZZ a, ZZ m)
{
    ZZ x, y;
    ZZ g = gcdExtended(a, m, &x, &y);
    if (g != 1)
    {
        return (ZZ)0;
    }
    else
    {
        ZZ res = (x % m + m) % m;
        return res;
    }
}

```

```

bool test(/*Change data type*/ZZ p)
{
    int count = 0;
    /*Change data type*/ZZ temp = p - 1, x, x1, x2, x0;
    x = 2;

```

```

while (1)
{
    if (temp % 2 == 0) { temp = temp / 2; count++; }
    else break;
}

```

```

x = RandomBnd(p);

```

```

if (gcdExtended(x, p, &x1, &x2) != 1) return 0;
ZZ power;

```

```

x0 = PowerMod(x, temp, p);

```

```

if (x0 == 1 || x0 == -1) return 1;
for (int i = 1; i < count; i++)
{

```

```

    x0 = PowerMod(x0, (ZZ)2, p);
    if (x0 == -1) return 1;
    if (x0 == 1) return 0;
}
return 0;

```

```

}

```

```

bool find(ZZ &p) {
    ZZ min;
    bool chk = 0;

```

```

    min = pow(2, 256);
    p = rando(min);

```

```

//cout << p << "\n";

```

```

int i = 0, k = 4; //your value

```

```

ZZ arr[5] = { (ZZ) 2, (ZZ)3, (ZZ)5, (ZZ)7, (ZZ)11 };
for (int i = 0; i < 5; i++)
    if (p % arr[i] == 0) { return 0; }

```

```

while (i < k) {
    if (test(p) == 1) i++;
    else break;
}

```

```

if (i >= k) chk = 1;

```

```

if (chk) return 1;
else return 0;

```

```

}

ZZ GenerateKeyPair(ZZ p, ZZ q, ZZ &n) {
    ZZ d, e;

    n = p * q;
    e = pow(2, 16) + 1;
    d = modInverse(e, (p - 1) * (q - 1));
    return d;
}

ZZ Encrypt(ZZ e, ZZ n, ZZ M)
{
    ZZ C = PowerMod(M, e, n);
    return C;
}

ZZ Decrypt(ZZ C, ZZ d, ZZ n)
{
    ZZ M = PowerMod(C, d, n);
    return M;
}

ZZ Sign(ZZ M, ZZ d, ZZ n)
{
    ZZ S = PowerMod(M, d, n);
    return S;
}

ZZ Verify(ZZ S, ZZ e, ZZ n)
{
    ZZ M;
    M = PowerMod(S, e, n);
    return M;
}

void Send(ZZ &S, ZZ &S1, ZZ &k1, ZZ k, ZZ d, ZZ n, ZZ n1, ZZ e1)
{
    S = PowerMod(k, d, n);
    S1 = PowerMod(S, e1, n1);
    k1 = PowerMod(k, e1, n1);
    cout << "S:" << S << endl;
    cout << "S1:" << S1 << endl;
    cout << "k1:" << k1 << endl;
}

void Recieve(ZZ &S, ZZ &S1, ZZ k1, ZZ d1, ZZ n1, ZZ k, ZZ e, ZZ n)
{
    S = PowerMod(S1, d1, n1);
    k = PowerMod(k1, d1, n1);
    if(k == PowerMod(S, e, n)) cout << "\nOk";
    else cout << "\nNot ok";
    cout << "\nk:: " << k << endl;
}

int main()
{
    ZZ p, q, p1, q1, e1, e2, n1, n2, d1, d2, C1, M1, C2, M2, k1, S1, k, S;
    while (1) {
        if (find(p)) {
            cout << endl << p << "\nPRIME(p)!!!"; break;

```



```

    }
}
while (1) {
    if (find(q)) {
        cout << endl << q << "\nPRIME!!(q)"; break;
    }
}
while (1) {
    if (find(p1)) {
        cout << endl << p1 << "\nPRIME!!(p1)"; break;
    }
}
while (1) {
    if (find(q1)) {
        cout << endl << q1 << "\nPRIME!!(q1)\n"; break;
    }
}
e1 = ::e;
e2 = ::e;

```

```

d1 = GenerateKeyPair(p, q, n1);
d2 = GenerateKeyPair(p1, q1, n2);

```

```

cout << "\nn1:: " << n1;
cout << "\nn2:: " << n2;
cout << "\ne1:: " << e1;
cout << "\ne2:: " << e2;
cout << "\nd1:: " << d1;
cout << "\nd2:: " << d2;

```

```

M1 = (ZZ)RandomBnd(n2 - 1);
C1 = Encrypt(e2, n2, M1);
M2 = (ZZ)RandomBnd(n1 - 1);
C2 = Encrypt(e1, n1, M2);

```

```

//User2
cout << "\nM1: " << M1 << endl;
cout << "C1 deciphered: " << Decrypt(C1, d2, n2);

```

```

//User1
cout << "\nM2: " << M2 << endl;
cout << "C2 deciphered: " << Decrypt(C2, d1, n1);

```

```

//-----

```

```

while (n2 > n1)
{

    while (1) {
        if (find(p)) {
            cout << endl << p << "\nPRIME(p)!!"; break;
        }
    }
    while (1) {
        if (find(q)) {
            cout << endl << q << "\nPRIME!!(q)"; break;
        }
    }
    while (1) {
        if (find(p1)) {
            cout << endl << p1 << "\nPRIME!!(p1)"; break;
        }
    }
}
}

```

```

        while (1) {
            if (find(q1)) {
                cout << endl << q1 << "\nPRIME!!(q1)\n"; break;
            }
        }
        d1 = GenerateKeyPair(p, q, n1);
        d2 = GenerateKeyPair(p1, q1, n2);
    }
    cout << "-----" << endl;
    k = RandomBnd(n1 - 2) + (ZZ)1; // Hexай
    cout << endl << "k::" << k << endl;

    S = Sign(M1, d1, n1);

    if (M1 == Verify(S, e1, n1)) {cout << "Verified\n"; cout << Verify(S, e1, n1) << endl;}
    else{ cout << "Not verified\n" << Verify(S, e1, n1) << endl;}

    /////
    cout << "\nn1::" << n1 << endl;
    cout << "\nn2::" << n2 << endl;

    //k1 = (ZZ)44;
    cout << "\nSend.\n";
    Send(S, S1, k1, k, d1, n1, n2, e2);
    cout << "\nRecieve..";
    Recieve(S, S1, k1, d2, n2, k, e1, n1);

    system("pause");
    return 0;
}

```

### Висновок:

У ході комп'ютерного практикуму було набуто навичок роботи з числами великої розрядності, написання тестів перевірки чисел на простоту та методів генерації ключів для асиметричної криптосистеми RSA. Набуто навичок побудови цифрового підпису на основі криптосистеми RSA.