



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-74, ФБ- 72

Каширін Євгеній, Жолоб Тетяна

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  $p_1, q_1$  довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб  $p \cdot q \leq p_1 \cdot q_1$ ;  $p$  і  $q$  – прості числа для побудови ключів абонента А,  $p_1$  і  $q_1$  – абонента В.

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 А і В – тобто, створити та зберегти для подальшого використання відкриті ключі  $(e, n)$ ,  $(e_1, n_1)$  та секретні  $d$  і  $d_1$ .

4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. 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 М і знайти криптограму для абонентів А і В, перевірити правильність розшифрування. Скласти для А і В 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().

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою <http://asymcryptwebservice.appspot.com/?section=rsa>. Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Результати:

### Абонент А

p = 319055494212607927262450575015279343881

q = 262724503321033822788539577780110531593

n = b95287e0fe0bf818690afb514ca91b5d44669d607d6ad54479b6176d6d7d51

e = 10001

d = 9b27ead54806e9ff73c876464521ceadc8edf4a58974d65bc0cb7f1447165d41

Підпис...

s = 11229443945185061444

### Абонент Б

p = 259267551073346617410231190869766514591

q = 242504443243351263104839713382779948247

n = 8b01289dcb4013ada25520d94c01ad837f031ec8ba9ae4ca618a618dc5903a89

e = 10001

d = 6450b1bf2eaf4f843f62b0122188146f810c7b3de069c2a310d05a591201ef01

Відправка ключа

**Key**  $\kappa$  = CA7D9DE769F2730DFB6A181ECA24C201

Підпис s = 11229443945185061444

```
C:\Users\Acer\Desktop\5,1\kripta.exe
Enter num of bits
256
319055494212607927262450575015279343881
262724503321033822788539577780110531593
259267551073346617410231190869766514591
242504443243351263104839713382779948247
public exponent = 10001
modulus = b95287e0fe0bf818690afb514ca91b5d44669d607d6ad54479b6176d6d7d51 private exp = 9b27ead54806e9ff73c876464521ceadc8edf4a58974d65bc0cb7f1447165d41
public exponent = 10001
modulus = 8b01289dcb4013ada25520d94c01ad837f031ec8ba9ae4ca618a618dc5903a89 private exp = 6450b1bf2eaf4f843f62b0122188146f810c7b3de069c2a310d05a591201ef01
Generated Msg = 269156294928672203297111376460261933569

-----
Process exited after 25.49 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

```
C:\Users\Acer\Desktop\5,1\rabota_s_saitom.exe
variables initialized
baluemsya crypt-decrypt sign-verify
38105404052500326343544446632219248640064563187359942384421146676659603343370
269156294928672203297111376460261933569
21842470088390800551486498454678023972189995644196619952724478445534843667229
269156294928672203297111376460261933569
Podpis = 11229443945185061444
Key = 11229443945185061444
Successsss!
Key = 175308fec5f093e93d42ce9f9abb47cec65ba687a837179a314981265f521fb3
Signature = aa61d50da2638d07f6628a1b0acbcc0b92acf78acda3afe9a9f87b3a22bab9cf
-----
Process exited after 3.23 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

## Receive key

✖ Clear

Key

175308fec5f093e93d42ce9f9abb47cec65ba687a837179a314981265f521fb3

Signature

aa61d50da2638d07f6628a1b0acbcc0b92acf78acda3afe9a9f87b3a22bab9cf

Modulus

b95287e0fef0bf818690afb514ca91b5d44669d607d6ad54479b6176d6d7d51

Public exponent

10001

Receive

Key

CA7D9DE769F2730DFB6A181ECA24C201

Verification

true

✓

## Код програми

# Файл kripa.cpp

```
#include "includes/biginteger.h"
#include <iostream>
#include <ctime>
#include <random>
#include <chrono>

using namespace std;
using namespace wide_integer::generic_template;

using bigint = uint8192_t;

unsigned int getted_bits;
unsigned int half_bits;

int default_primes[5] = {3, 5, 7, 11, 13};

bigint pow(bigint f, bigint s, bigint smth, int i)
{
    for (; ++i < s - 1;)
        smth = smth * f;
    return (smth * f);
}

void pow_mod(bigint &ret, bigint in_base, bigint power, bigint modulus)
{
    if (power == 0)
    {
        ret = 1;
        return ;
    }
    bigint y("1");
    bigint base = in_base;
    while (power - 1 > 0)
    {
        if (power % 2 == 0)
        {
            base = ((base % modulus) * (base % modulus));
            base = base % modulus;
            power = power / 2;
        }
        else
        {
            y = (y * base);
            y = y % modulus;
            base = ((base % modulus) * (base % modulus)) % modulus;
            power -= 1;
            power /= 2;
        }
    }
    ret = ((base % modulus) * (y % modulus)) % modulus;;
}

void random_bigint(bigint &gen_int, int half_bits, mt19937 &mt_rand)
{
    gen_int = gen_int + pow(2, half_bits - 1, "1", -1);
    int i = 0;
    for (; ++i < half_bits - 1;)
        if (mt_rand() % 2)
            gen_int += pow(2, half_bits - i - 1, "1", -1);
    gen_int += 1;
}

int check_default(bigint &n)
{
    for (int i = 0; i < 5; i++)
        if ((n % default_primes[i]) == 0)
            return (0);

    return (1);
}

int miller_rabin(int k, bigint n, mt19937 &mt_rand)
{
    bigint test_result;
    if (!check_default(n))
        return (0);
    bigint n_min_1 = n - 1;
    bigint s = 0;
    bigint d;
    for (d = n - 1; (d & 1) == 0; (s += 1), (d = d / 2))
        ;
    for (; --k; )
    {
        bigint a = 0;
        random_bigint(a, half_bits - 2, mt_rand);
        pow_mod(test_result, a, d, n);
        if (test_result == 1 || test_result == n_min_1)
            continue;
        unsigned int j = 0;
        for (; j < s; j++)
        {
            pow_mod(test_result, test_result, 2, n);
            if (test_result == 1)
                return 0;
            if (test_result + 1 == n)
                break;
        }
        if (j == s)
            return 0;
    }
    return 1;
}

int go_cycle(bigint n, bigint *primes, int num, bigint maximum, int i, mt19937 &mt_rand)
{
    maximum = pow(2, half_bits, "1", -1);
    for (; n < maximum;)
    {
        int ret = miller_rabin(10, n, mt_rand);
        if (ret)
        {
            primes[num] = n;
            cout << n << endl;
            return (1);
        }
        n += 2;
    }
    return (0);
}

void generate_primes(auto seed, mt19937 &mt_rand, int num, bigint *primes)
{
    bigint gen_int("0");
    random_bigint(gen_int, half_bits, mt_rand);
    if (!go_cycle(gen_int, primes, num, "0", -1, mt_rand))
    {
        generate_primes(seed, mt_rand, num, primes);
    }
}

void create_modules(bigint *modules, bigint *primes)
{
    modules[0] = primes[0] * primes[1];
    modules[1] = primes[2] * primes[3];
}

void create_pub(bigint *pub)
{
    pub[0] = 65537;
    pub[1] = 65537;
}

int extended_euclid(bigint a, bigint b, bigint &x, bigint &y, bigint &d, bigint q, bigint r, bigint x1, bigint x2, bigint y1, bigint y2)
```

```

{
    int flag = 0;
    if (!(int)b)
    {
        x = 1;
        d = a;
        y = 0;
        return 2;
    }
    x2 = 1, x1 = 0, y2 = 0, y1 = 1;
    while (b > 0)
    {
        q = a / b, r = a - q * b;
        if (flag == 0)
        {
            x = x2 - q * x1;
            flag++;
        }
        else if (flag == 1)
        {
            x = q * x1 + x2;
            flag++;
        }
        else if (flag != 0 && flag != 1)
        {
            x = q * x1 + x2;
            flag -= 1;
        }
        y = y2 - q * y1;
        a = b, b = r;
        x2 = x1, x1 = x, y2 = y1, y1 = y;
    }
    d = a, x = x2, y = y2;
    return flag;
}

void inverse_unsigned(bigint &ret, bigint a, bigint n)
{
    bigint d, x, y;
    int flag = extended_euclid(a, n, x, y, d, 0, 0, 0, 0, 0);
    if (flag == 1)
        x = n - x;
    if (d == 1)
    {
        ret = x;
        return;
    }
    ret = 0;
}

void create_priv(bigint *priv, bigint *modules, bigint *primes)
{
    bigint euler1 = primes[0] - 1;
    bigint euler2 = primes[1] - 1;
    bigint euler = euler1 * euler2;
    inverse_unsigned(priv[0], "65537", euler);
}

euler1 = primes[2] - 1;
euler2 = primes[3] - 1;
euler = euler1 * euler2;
inverse_unsigned(priv[1], "65537", euler);
}

char *itoa_base(bigint value, char *str, unsigned int size, bigint tmp)
{
    int base = 16;
    string tab;

    tab = "0123456789abcdef";
    tmp = value;
    for (;(tmp /= base) > 0;)
        size++;
    size = size + 1;
    str = new char[size + 1];
    str[size] = '\0';
    for (;size > 0;)
    {
        str[size - 1] = tab[(int)(value % base)];
        size--;
        value = value / base;
    }
    return (str);
}

int main(void)
{
    auto seed =
chrono::high_resolution_clock::now().time_since_epoch().count();
    mt19937 mt_rand(seed);//seeding mersenn twister PRNG
    cout << "enter num of bits\n";
    cin >> getted_bits;
    half_bits = getted_bits / 2;
    bigint primes[4];
    bigint modules[2];
    bigint pub[2];
    bigint priv[2];
    for (int i = 0; i < 4; i++)
        generate_primes(seed, mt_rand, i, primes);
    create_modules(modules, primes);
    create_pub(pub);
    create_priv(priv, modules, primes);
    cout << "public exponent = " << itoa_base(pub[0], NULL, 0, 0) << endl;
    cout << "modulus = " << itoa_base(modules[0], NULL, 0, 0) << " private
exp = " << itoa_base(priv[0], NULL, 0, 0) << endl;
    cout << "public exponent = " << itoa_base(pub[1], NULL, 0, 0) << endl;
    cout << "modulus = " << itoa_base(modules[1], NULL, 0, 0) << " private
exp = " << itoa_base(priv[1], NULL, 0, 0) << endl;
    bigint Msg;
    random_bigint(Msg, half_bits, mt_rand);
    cout << "Generated Msg = " << Msg << endl;
}

```

## Файл krypta.cpp

```

#include "includes/biginteger.h"
#include <iostream>
#include <string>

```

```

using namespace std;
using namespace wide_integer::generic_template;

```

```

using bigint = uint8192_t;

```

```

bigint msg
("87259850849553274357038739578702628995861132959698615394546622895177706547929543617301138762016158977
9165833436925620218487896656141747250888783812062043839263125039073754391630887860182438564127757481290
8733105574175092340874596480970324431215090408143202020286399337600080636800212286694407242382447047618
8040130415638052312880424989164801187794082823484099667514313500221495702347527044449401255547995632534
2260112651920492884736205150803662410661194852668837793925905954249796105214206015331919381092594977945
8218944600221023064361652014342093607565593347978063037470813896399975207688964125240400924490171931940
1227368459917330363762622905720458466274417733197625951691090820130112761293440434209874879704713735315
6944949925116248532323908591547866301942533925996296794809495005087976312093617458893399566896775885605
0108896661653723265256532601098248216178487420519282214073284167026437059288394916550039367738583859673
4781074190703221795875041482158221195285817248926434775266196747199356913139419312847929394084844518701
1005051622089090126902706694896926394910416702830140436269431932555387547164533052056069770828376204536

```

```

2496033045803966676066859517230631488616647828404631303743258822240439484972488726366761383632927783419
8390202219084800656329484714400382219898220948305080839507246548927892176928476824668512475345695177123
0294980975585198982082922893039084939857930426788351028259219801135533746409824862567783442829898243659
4976761829628982210911562326958450067378841188060568316955099597038934465510157302526225224635145432745
1491867260939086545565040239553823260396404903105464674406371895458574332701273442845217292569232239369
6220924486298486349981973473222690390166959560121624985490075362468555071155725649076989363387221456882
3671639609576696612369492982370356291612966052074808151248190190572316262209669415828717855730642471146
4373213554894806268051325466955562404114999304903906749381219087632392341876474297915962018174769392349
428380368837950161581442046128470483046181656617435377581672308350868675276393692879882031123179724563
1551784688727444775706886140239719507316785485647710036491978846622508990666311250472034304138571398469
3117013706682240738620668069844769129497902513768116815889259790948407751687135134435307164681985829403
2890570111034940454120272290376025544018718262772254837790595385622412151751728335058380800652350088521
803416130876520700347080917698571381813284322865425175272645529934900996496416890078023419236897085");
//Вставляем сюда сгенерированный меседж
string my_modulus = "0x6f5ac5f949dac9ea21fbb5e32d0c76ce30b6de479667a36b675614e108d34e15";// модуль
string private_exp = "0x26dedbeeba4e2a5f957f452dc902e9d17665757f2f16ef1274d3497743df1321";// наш приватный ключ
string site_modulus = "0x929A4E3EA753570A3DBE73951C70CA8F0A18A205B184CEC709E0EFF8FB8280DF";// модуль сайта
string Key_site1 = "0x26A0E0D5EFA6A582FE15B10EDC5F4C494E2EF824547D9116C30AA57C9032F4B7";// сгенерированный
ключ из сайта
string Signature_site = "0x6f5ac5f949dac9ea21fbb5e32d0c76ce30b6de479667a36b675614e108d34e15";// Signature из сайта

```

```

bigint pub("65537");

```

```

return sign;

```

```

void pow_mod(bigint &ret, bigint in_base, bigint power, bigint
modulus)
{

```

```

int receive_key(bigint &K, bigint &sign, bigint &my_d, bigint &my_n,
bigint &pub_site, bigint &site_n)
{

```

```

    if (power == 0)
    {

```

```

        bigint copy_s = sign;
        decrypt(K, my_d, my_n);
        if (K == 0)

```

```

            return (0);
        decrypt(sign, my_d, my_n);
        verify(sign, pub_site, site_n);
        cout << "Podpis = " << sign << endl << "Key = " << K << endl;
        return ((int)sign == (int)K);
    }
}

```

```

        bigint y("1");
        bigint base = in_base;
        while (power - 1 > 0)
        {

```

```

            if (power % 2 == 0)
            {

```

```

                *itoa_base(bigint value, char *str, unsigned int size, bigint
char tmp)
{

```

```

                base = ((base % modulus) * (base % modulus));
                base = base % modulus;
                power = power / 2;
            }

```

```

            int base = 16;
            string tab;

```

```

        }
        else
        {

```

```

            tab = "0123456789abcdef";
            tmp = value;
            for (;(tmp /= base) > 0;)

```

```

            y = (y * base);
            y = y % modulus;
            base = ((base % modulus) * (base % modulus)) % modulus;
            power -= 1;
            power /= 2;
        }
    }
    ret = ((base % modulus) * (y % modulus)) % modulus;
}

```

```

            size++;
            size = size + 1;
            str = new char[size + 1];
            str[size] = '\0';
            for (;size > 0;)
            {

```

```

void encrypt(bigint &message, bigint e, bigint n)
{

```

```

                str[size - 1] = tab[(int)(value % base)];
                size--;
                value = value / base;
            }
        }
        return (str);
    }
}

```

```

    bigint ret;
    pow_mod(ret, message, e, n);
    message = ret;
}

```

```

bigint decrypt(bigint &encrypted, bigint d, bigint n)
{

```

```

void sendkey(bigint &mymod, bigint &priv, bigint &e2, bigint &n2)
{

```

```

    bigint ret;
    pow_mod(ret, encrypted, d, n);
    encrypted = ret;
    return encrypted;
}

```

```

    bigint Sign = msg;
    sign(Sign, priv, mymod);
    encrypt(Sign, e2, n2);
    encrypt(msg, e2, n2);
    cout << "Key = " << itoa_base(msg, NULL, 0, 0) <<
"\nSignature = " << itoa_base(Sign, NULL, 0, 0);
}

```

```

bigint sign(bigint &message, bigint d, bigint n)
{

```

```

int main()
{

```

```

    bigint ret;
    pow_mod(ret, message, d, n);
    message = ret;
    return message;
}

```

```

    bigint my_module(my_modulus.c_str());
    bigint my_priv(private_exp.c_str());
    bigint site_module(site_modulus.c_str());
    bigint Key_site(Key_site1.c_str());
    bigint Sign_site(Signature_site.c_str());
    cout << "variables initialized\n";
    cout << "baluemsya crypt-decrypt sign-verify\n";
    encrypt(msg, pub, my_module);
    cout << msg << endl;
    decrypt(msg, my_priv, my_module);
}

```

```

bigint verify(bigint &sign, bigint e, bigint n)
{

```

```

    bigint ret;
    pow_mod(ret, sign, e, n);
    sign = ret;
}

```

```

cout << msg << endl;
sign(msg, my_priv, my_module);
cout << msg << endl;
verify(msg, pub, my_module);
cout << msg << endl;
if (receive_key(Key_site, Sign_site, my_priv, my_module,
pub, site_module))
    cout << "Successss!\n";
else
    cout << "Failed\n";
sendkey(my_module, my_priv, pub, site_module);
}

```

---

### **Висновки:**

Під час данного комп'ютерного практикуму, ми ознайомились з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA. Також, використовуючи криптосистему типу RSA, організували канал засекреченого зв'язку й електронний підпис, ознайомились із протоколом розсилання ключів.