

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

## Лабораторна робота 3

Реалізація основних асиметричних криптосистем

*Підгрупа 2А*

Виконали:

Галіца О.О.

Паршин О.Ю.

Литвиненко Ю.С.

ФІ-22мн

Перевірила:

Байденко П.В.

# 1 Теоретичні відомості

## 1.1 Криптосистема шифрування Ель-Гамала

Розглянемо криптографічну схему шифрування, яка побудована на односторонній функції Діффі-Хеллмана. Цей алгоритм запропоновано в 1985 році, пізніше найбільш відомої та поширеної схеми асиметричної криптографії RSA. Алгоритм шифрування Ель-Гамала має загальну структуру систем шифрування з відкритими ключами: систему шифрування з відкритим і секретним ключами будує один абонент, секретний ключ знає тільки і саме секретний ключ використовується для розшифрування шифрованого повідомлення, яке зашифровано «загальнодоступним» ключем будь-яким іншим абонентом.

### Побудова схеми шифрування Ель-Гамала користувачем А.

Для цього абонент А виконує наступні кроки:

1. Обирає велике просте число  $p$  та  $\alpha$  — примітивний елемент поля  $F_p$ .
2. Обирає випадкове число  $k$ ,  $1 < k < p$  — секретний ключ А, який використовується для розшифрування.
3. Обчислює  $y = \alpha^k \bmod p$ . Параметри  $(p, \alpha, y)$  складають відкритий ключ, що застосовується для шифрування повідомлень, призначених для абонента А.
4. Забезпечує доступність відкритого ключа для всіх абонентів, які бажають відіслати А зашифроване повідомлення.

Відкритий ключ може бути з підтвердженням автентичності (зазвичай, центрами сертифікації ключів ЦСК).

### Шифрування повідомлення М, $1 < M < p$ в системі шифрування Ель-Гамала користувачем В для користувача А, який побудував систему і має секретний ключ для розшифрування.

Користувач В, який знає алгоритм шифрування і відкритий ключ хоче переслати користувачу А зашифроване повідомлення, що кодується і є відповідним цілим числом  $M$ ,  $1 < M < p$ . Для цього він виконує наступні дії:

1. Вибирає випадкове число  $x_M$ ,  $1 < x_M < p - 1$
2. Обчислює  $C_1 = \alpha^{x_M} \bmod p$  та  $C_2 = y^{x_M} \bmod p$
3. Формує шифрований текст, що є впорядкованою парою чисел (складається з двох частин):  $(C_1, C_2)$

## Розшифрування зашифрованого повідомлення в системі шифрування Ель-Гамалія користувачем А.

Користувач А обчислює:

$$C_1^{-k} C_2 \bmod p = \alpha^{-x_M k} \cdot \alpha^{k x_M} \bmod p = M$$

Значення  $C_1^{-k}$  можна обчислити як  $(C_1^{-1})^k \bmod p$ , або як  $(C_1^k)^{-1} \bmod p$ , де обернений за модулем швидко знаходиться алгоритмом Евкліда.

Стійкість схеми шифрування Ель-Гамалія обумовлена складністю розв'язання задачі дискретного логарифмування.

### Зауваження

Для різних повідомлень  $M$  та  $M'$  числа  $X_M$  та  $X_{M'}$  мають бути різними. Таким чином, неякисний генератор випадкових чисел, використаний для вибору чисел  $x_M$ , та порушення процедури шифрування може призвести до рівності  $x_M = x_{M'}$  і, відповідно, до успішної атаки на систему.

## 1.2 Цифровий підпис Ель-Гамалія

### Побудова криптосистеми ЦП Ель-Гамалія користувачем А.

Користувач А виконує наступні кроки:

1. Обирає велике просте число  $p$  та  $\alpha$  — примітивний елемент поля  $F_p$ .
2. Обирає випадкове число  $k$ ,  $1 < k < p$  — секретний ключ для формування ЦП.
3. Обчислює  $y = \alpha^k \bmod p$ . Параметри  $(p, \alpha, y)$  складають відкритий ключ, що застосовується для перевірки електронного підпису користувача А.

### Формування ЦП для повідомлення М, $1 < M < p$ в системі, ЦП Ель-Гамалія користувачем А.

1. Вибирає випадкове число  $x_M$ ,  $1 < x_M < p - 1$ ,  $(x_M, p - 1) = 1$
2. Обчислює  $r = \alpha^{x_M} \bmod p$
3. Розв'язує відносно  $s$  рівняння  $M \equiv (kr + x_M s) \bmod (p - 1)$ , тобто  $s = (M - kr)x_M^{-1} \bmod (p - 1)$
4. Формує повідомлення з цифровим підписом:  $(M, r, s)$ , де пара чисел  $(r, s)$  є цифровим підписом.

## Перевірка цифрового підпису Ель-Гамалія

Користувачем В перевіряється рівність:

$$y^r r^s \bmod p = \alpha^M \bmod p$$

Якщо ця рівність виконується, то підпис вірний, підтверджена цілісність повідомлення і справжність автора. Інакше мало місце навмисне чи випадкове спотворення повідомлення або підпису.

### Зауваження

Для різних  $M_1 \neq M_2$  параметри ЦП  $x_{M_1}$  і  $x_{M_2}$  повинні бути різними. Тобто,  $x_{M_i}$  не повинно ніколи повторюватися. Інакше криптоаналітик Е знаходить секретний ключ для підписування повідомлень і зламує систему цифрового підпису Ель-Гамалія.

Стійкість криптосистеми цифрового підпису Ель-Гамалія як і криптосистеми шифрування обумовлена складністю розв'язання задачі дискретного логарифмування.

## 2 Результати

В ході лабораторної роботи була реалізована криптосистема Ель-Гамалія з параметризованою довжиною ключа. В якості допоміжної бібліотеки для роботи з довгою арифметикою була використана бібліотека OpenSSL.

Тестове повідомлення: D4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D379AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4CED2B41A0BD38BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C320A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910DD17B6766A7EFEE215A98CAC300F2827DB



```

Microsoft Visual Studio Debu  X  +  -
Input message that will be encrypted:
D4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D379AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4CED2B41A0BD38
BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C320A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910DD17B6766A7EFEE215A98CAC
300F2827DB

c1:
BA77C00A34E3DE16DED833B9658779991C7DF01E25F2B2E00A985701754804655A51918FD59359FD103DC8F700E8C8B2386F97800D0B84C5AEEC521B443
374FF10BD51B2B7F02E912E65F3C28E2421847BF5B3D6DCBE6997810AD7AD6CE8682BEBACE5E1B4644879C428FAEED322960258BFAAFC987DA6D06C8CC6
E6B42012C8DBB7F5FD90514F579EBE407AB76547B8356A6A05B350374287C8682F65BA3CBFBF8C2A57DD4B9750DCE55BA805F697EF0273FCCA6AAB6513B
65E05AAEDF6E2DC9FF6185327A942B8F74E1A3EEC1984B16AB9883D26F2277AFFBE2173C662DAD72F2A511031A737BC901160B631907184E580CBD73331
F874A01622DCB37F1D0C

c2:
030A4FEF8FA403AEDC06083F79990A6003B02910A76092A69990DFD9487BE7EB8BF8D0D6C1FD44B4EB77932D66B3FE2BCC31A8DC6A062FA36CAB8A7FB5D
CBE4A6E13497BDF4735F9E50150F0A7CDD15938A05401CCFFC54D6661DAFF727A9A72962F03AC8CA0C1AF690141014BD39711E1D2C3EBBA5F1ACA03C2E2
F4376CC5C0B6635DC6347DD65E6E3F9BCD72FD9B712F3EEDEF3A57C10B4BB7F6A2BF0EAB417D7CBDC3634FEDE2B5369B1569BB288B50552B6A0F13B2621
7AD5A3AC8ACD617C87C9AA2549B31A7885603CF493C547DEDE2773C1C94A835AAB0C76952BDEDC9B5997CDD1E32FBD57B6AFE186320361ACF5D20CFA6E1
8E3468341BE6C275891A

-----
Input message that will be decrypted:
c1: BA77C00A34E3DE16DED833B9658779991C7DF01E25F2B2E00A985701754804655A51918FD59359FD103DC8F700E8C8B2386F97800D0B84C5AEEC521
B443374FF10BD51B2B7F02E912E65F3C28E2421847BF5B3D6DCBE6997810AD7AD6CE8682BEBACE5E1B4644879C428FAEED322960258BFAAFC987DA6D06C
8CC6E6B42012C8DBB7F5FD90514F579EBE407AB76547B8356A6A05B350374287C8682F65BA3CBFBF8C2A57DD4B9750DCE55BA805F697EF0273FCCA6AAB6
513B65E05AAEDF6E2DC9FF6185327A942B8F74E1A3EEC1984B16AB9883D26F2277AFFBE2173C662DAD72F2A511031A737BC901160B631907184E580CBD7
3331F874A01622DCB37F1D0C

c2: 030A4FEF8FA403AEDC06083F79990A6003B02910A76092A69990DFD9487BE7EB8BF8D0D6C1FD44B4EB77932D66B3FE2BCC31A8DC6A062FA36CAB8A7
FB5DCBE4A6E13497BDF4735F9E50150F0A7CDD15938A05401CCFFC54D6661DAFF727A9A72962F03AC8CA0C1AF690141014BD39711E1D2C3EBBA5F1ACA03
C2E2F4376CC5C0B6635DC6347DD65E6E3F9BCD72FD9B712F3EEDEF3A57C10B4BB7F6A2BF0EAB417D7CBDC3634FEDE2B5369B1569BB288B50552B6A0F13B
26217AD5A3AC8ACD617C87C9AA2549B31A7885603CF493C547DEDE2773C1C94A835AAB0C76952BDEDC9B5997CDD1E32FBD57B6AFE186320361ACF5D20CF
A6E18E3468341BE6C275891A

m:
D4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D379AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4CED2B41A0BD38
BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C320A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910DD17B6766A7EFEE215A98CAC
300F2827DB
  
```

Рис. 1: Шифрування та розшифрування

```

Microsoft Visual Studio Debu  X  +  -
-----
Input message that will be signed:
D4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D379AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4CED2B41A0BD38
BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C320A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910DD17B6766A7EFEE215A98CAC
300F2827DB

r:
D12FBF4D90A93E642BC586851417318F9926BA2E595464CE28CA06B3DB73096433B1E8CE8C25E984E56B615DAB2248EE43D7114049D6AE6575D2FF8EA8D
8B8109A4BE7DFBEC3B233F18EFC26A446E3FCBE570D0807AC70593C82182402E9581C8214AAF0E9E99C4218A9277C48F1570CD5C97E4D54C89F0BB928A9
944713B80E70182B53E8FE7E5F11BD143AB582CC14B0DC8FCCC043237266CA879F931A3806D5F1232F219FE170DB4A97A1A319D1A1DD39D7D5F3039E22E
A39EF0B213EF894053F1EBD07937F8F8222AB3649A498448BE11F8FF7C03F901BE25CAC803F88F1377917A5AADEF243D9FDF2B48D698A9F3696607B545
542F1AE3450DD749236A

s:
158564B3861653734F1DD67E611BC2D55E9819387E92EF6674639CA02518648D3A3762E843C7C2CBE0EC9A46BDD4714367BF9261C2BFB703545D338C1C1
AA926BAE2BF34B81980225DD8D6A30E1FAF44BECF32778F1278ACF15C114B819DABB065BEE52FA5D0C7F0A95EA8E62CB3CBFC907499FC9B8C8052BF613
CB91CC303D7E8DD7765CFE7F8BE9046597CF8ACB7A454515F16CC47A6B26C3816AD20799DDF938A72D3B206ADB6709ECFE31E78EFA9143B396F87FECE57F
633AE7EFAFC9F60EB0F561094AEF077BF07A29B1E70769B6B8E65BD9E138630BC1CF07026CC6422B2D60296E6E1D41ADB8E4CB22C7B9BC01ABD945B6B
77CD9AB469CF0EE2E597

-----
Input message and sign that will be verified:
m: D4D2110984907B5625309D956521BAB4157B8B1ECE04043249A3D379AC112E5B9AF44E721E148D88A942744CF56A06B92D28A0DB950FE4CED2B41A0B
D38BCE7D0BE1055CF5DE38F2A588C2C9A79A75011058C320A7B661C6CE1C36C7D870758307E5D2CF07D9B6E8D529779B6B2910DD17B6766A7EFEE215A98
CAC300F2827DB

r: D12FBF4D90A93E642BC586851417318F9926BA2E595464CE28CA06B3DB73096433B1E8CE8C25E984E56B615DAB2248EE43D7114049D6AE6575D2FF8E
A8D8B8109A4BE7DFBEC3B233F18EFC26A446E3FCBE570D0807AC70593C82182402E9581C8214AAF0E9E99C4218A9277C48F1570CD5C97E4D54C89F0BB92
8A9944713B80E70182B53E8FE7E5F11BD143AB582CC14B0DC8FCCC043237266CA879F931A3806D5F1232F219FE170DB4A97A1A319D1A1DD39D7D5F3039E
22EA39EF0B213EF894053F1EBD07937F8F8222AB3649A498448BE11F8FF7C03F901BE25CAC803F88F1377917A5AADEF243D9FDF2B48D698A9F3696607B
545542F1AE3450DD749236A

s: 158564B3861653734F1DD67E611BC2D55E9819387E92EF6674639CA02518648D3A3762E843C7C2CBE0EC9A46BDD4714367BF9261C2BFB703545D338C
1C1AA926BAE2BF34B81980225DD8D6A30E1FAF44BECF32778F1278ACF15C114B819DABB065BEE52FA5D0C7F0A95EA8E62CB3CBFC907499FC9B8C8052BF
613CB91CC303D7E8DD7765CFE7F8BE9046597CF8ACB7A454515F16CC47A6B26C3816AD20799DDF938A72D3B206ADB6709ECFE31E78EFA9143B396F87FECE
57F633AE7EFAFC9F60EB0F561094AEF077BF07A29B1E70769B6B8E65BD9E138630BC1CF07026CC6422B2D60296E6E1D41ADB8E4CB22C7B9BC01ABD945
B6B77CD9AB469CF0EE2E597

Sign is correct: 1
-----

D:\Documents\Projects\cryptographic-mechanisms\lab-3\x64\Release\lab-3.exe (process 17500) exited with code 1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console
when debugging stops.
Press any key to close this window . . .|

```

Рис. 2: Підписування та перевірка підпису

## 3 Лістинг програми

### 3.1 el-gamal.cpp

```
#include "el-gamal.hpp";
```

```

ElGamal::ElGamal(const uint64_t& keyLength)
{
    this->keyLength = keyLength;

    p = rand_prime_big_number(keyLength);
    q = (p - 1) / 2;
    g = BigNumber::sqr_mod(rand_big_number(keyLength - 1, -1), p);
    x = rand_big_number(keyLength - 2, 0);
    y = BigNumber::exp_mod(g, x, p);
}

```

```

std::pair<BigNumber, BigNumber> ElGamal::encrypt(const BigNumber& m)
{
    BigNumber k = rand_big_number(keyLength - 2, 0);

    auto c = std::pair<BigNumber, BigNumber>();
    c.first = BigNumber::exp_mod(g, k, p);
    c.second = (m * BigNumber::exp_mod(y, k, p)) % p;

    return c;
}

BigNumber ElGamal::decrypt(const std::pair<BigNumber, BigNumber>& c)
{
    return (c.second * BigNumber::exp_mod(c.first, -x % (p - 1), p)) % p;
}

std::pair<BigNumber, BigNumber> ElGamal::sign(const BigNumber& m)
{
    BigNumber k = rand_big_number(keyLength - 2, 0);
    auto sign = std::pair<BigNumber, BigNumber>();

    sign.first = BigNumber::exp_mod(g, k, p);
    sign.second = (m - sign.first * x) * BigNumber::exp_mod(k, q - 2, q) % q;

    return sign;
}

bool ElGamal::verify(const BigNumber& m, const std::pair<BigNumber, BigNumber>& sign)
{
    BigNumber leftPart = BigNumber::exp_mod(y, sign.first, p) * BigNumber::exp_mod(
        sign.first,
        sign.second,
        p
    ) % p;

    BigNumber rightPart = BigNumber::exp_mod(g, m, p);

    return leftPart == rightPart;
}

```

## 3.2 big-number.cpp

```
#include "./big-number.hpp";
```

```

BigNumber::BigNumber(): value(initEmpty(), ::BN_free) {}
BigNumber::BigNumber(const int64_t number): value(initDecimal(number), ::BN_free) {}
BigNumber::BigNumber(const std::string& str): value(initHex(str), ::BN_free) {}
BigNumber::BigNumber(const BigNumber& obj): value(copy(obj.value), ::BN_free) {}

BigNumber& BigNumber::operator=(const BigNumber& obj)
{
    if (this != &obj) value.reset(BN_dup(obj.value.get()));

    return *this;
}

BigNumber BigNumber::add(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    BigNumber result;

    BN_add(result.value.get(), firstOperand.value.get(), secondOperand.value.get());

    return result;
}

BigNumber BigNumber::sub(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    BigNumber result;

    BN_sub(result.value.get(), firstOperand.value.get(), secondOperand.value.get());

    return result;
}

BigNumber BigNumber::mod(const BigNumber& number, const BigNumber& modulo)
{
    BigNumber result;
    bigNumberContext ctx(BN_CTX_new(), ::BN_CTX_free);

    BN_nnmod(result.value.get(), number.value.get(), modulo.value.get(), ctx.get());

```



```

    return result;
}

BigNumber BigNumber::add_mod(
    const BigNumber& firstOperand,
    const BigNumber& secondOperand,
    const BigNumber& modulo
)
{
    BigNumber result;
    bigNumberContext ctx(BN_CTX_new(), ::BN_CTX_free);

    BN_mod_add(
        result.value.get(),
        firstOperand.value.get(),
        secondOperand.value.get(),
        modulo.value.get(),
        ctx.get()
    );

    return result;
}

BigNumber BigNumber::mul(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    BigNumber result;
    bigNumberContext ctx(BN_CTX_new(), ::BN_CTX_free);

    BN_mul(
        result.value.get(),
        firstOperand.value.get(),
        secondOperand.value.get(),
        ctx.get()
    );

    return result;
}

BigNumber BigNumber::mul_mod(

```

```

    const BigNumber& firstOperand,
    const BigNumber& secondOperand,
    const BigNumber& modulo
)
{
    BigNumber result;
    bigNumberContext ctx(BN_CTX_new(), ::BN_CTX_free);

    BN_mod_mul(
        result.value.get(),
        firstOperand.value.get(),
        secondOperand.value.get(),
        modulo.value.get(), ctx.get()
    );

    return result;
}

BigNumber BigNumber::div(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    BigNumber result;
    bigNumberContext ctx(BN_CTX_new(), ::BN_CTX_free);

    BN_div(
        result.value.get(),
        nullptr,
        firstOperand.value.get(),
        secondOperand.value.get(),
        ctx.get()
    );

    return result;
}

BigNumber BigNumber::sqr_mod(const BigNumber& number, const BigNumber& modulo)
{
    return BigNumber::mul_mod(number, number, modulo);
}

BigNumber BigNumber::exp_mod(

```

```

    const BigNumber& number,
    const BigNumber& power,
    const BigNumber& modulo
)
{
    BigNumber result;
    bigNumberContext ctx(BN_CTX_new(), ::BN_CTX_free);

    BN_mod_exp(
        result.value.get(),
        number.value.get(),
        power.value.get(),
        modulo.value.get(),
        ctx.get()
    );

    return result;
}

BIGNUM* BigNumber::initEmpty()
{
    auto bigNumber = BN_new();
    BN_zero(bigNumber);

    return bigNumber;
}

BIGNUM* BigNumber::initDecimal(const int64_t& number)
{
    auto bigNumber = BN_new();
    BN_dec2bn(&bigNumber, std::to_string(number).c_str());

    return bigNumber;
}

BIGNUM* BigNumber::initHex(const std::string& hexString)
{
    auto bigNumber = BN_new();
    BN_hex2bn(&bigNumber, hexString.c_str());

```

```

    return bigNumber;
}

BIGNUM* BigNumber::copy(const BigNumberPointer& arg)
{
    return BN_dup(arg.get());
}

BigNumber rand_big_number(const uint64_t& bitsCount, const int8_t& firstSignificantBit)
{
    BigNumber randomNumber;
    BN_rand(randomNumber.value.get(), bitsCount - 1, firstSignificantBit, false);

    return randomNumber;
}

BigNumber rand_prime_big_number(const uint64_t& bitsCount)
{
    BigNumber randomPrime;
    BN_generate_prime_ex(randomPrime.value.get(), bitsCount, 1, NULL, NULL, NULL);

    return randomPrime;
}

BigNumber operator+(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::add(firstOperand, secondOperand);
}

BigNumber operator+(const int64_t& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::add(firstOperand, secondOperand);
}

BigNumber operator+(const BigNumber& firstOperand, const int64_t& secondOperand)
{
    return BigNumber::add(firstOperand, secondOperand);
}

BigNumber operator-(const BigNumber& number)

```

```

{
    return -1 * number;
}

BigNumber operator-(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::sub(firstOperand, secondOperand);
}

BigNumber operator-(const int64_t& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::sub(firstOperand, secondOperand);
}

BigNumber operator-(const BigNumber& firstOperand, const int64_t& secondOperand)
{
    return BigNumber::sub(firstOperand, secondOperand);
}

BigNumber operator*(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::mul(firstOperand, secondOperand);
}

BigNumber operator*(const int64_t& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::mul(firstOperand, secondOperand);
}

BigNumber operator*(const BigNumber& firstOperand, const int64_t& secondOperand)
{
    return BigNumber::mul(firstOperand, secondOperand);
}

BigNumber operator/(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::div(firstOperand, secondOperand);
}

BigNumber operator/(const int64_t& firstOperand, const BigNumber& secondOperand)

```

```

{
    return BigNumber::div(firstOperand, secondOperand);
}

BigNumber operator/(const BigNumber& firstOperand, const int64_t& secondOperand)
{
    return BigNumber::div(firstOperand, secondOperand);
}

BigNumber operator%(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::mod(firstOperand, secondOperand);
}

BigNumber operator%(const int64_t& firstOperand, const BigNumber& secondOperand)
{
    return BigNumber::mod(firstOperand, secondOperand);
}

BigNumber operator%(const BigNumber& firstOperand, const int64_t& secondOperand)
{
    return BigNumber::mod(firstOperand, secondOperand);
}

bool operator==(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    return BN_cmp(firstOperand.value.get(), secondOperand.value.get()) == 0;
}

bool operator<(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    return BN_cmp(firstOperand.value.get(), secondOperand.value.get()) == -1;
}

bool operator>(const BigNumber& firstOperand, const BigNumber& secondOperand)
{
    return BN_cmp(firstOperand.value.get(), secondOperand.value.get()) == 1;
}

std::ostream& operator<<(std::ostream& out, const BigNumber& number)

```

```
{  
    const long f = out.flags() & std::ios::basefield;  
    char* ptr = nullptr;  
  
    ptr = BN_bn2hex(number.value.get());  
    out << ptr;  
  
    return out;  
}
```