

Министерство образования и науки Российской Федерации  
Новосибирский государственный технический университет  
Кафедра прикладной математики

Основы криптографии  
Лабораторная работа №2

Факультет:           прикладной математики и информатики  
Группа:               ПМ-63  
Студенты:           Шепрут И.М.  
                              Кожекин М.В.  
                              Утюганов Д.С.  
Преподаватель:   Ступаков И.М.

Новосибирск

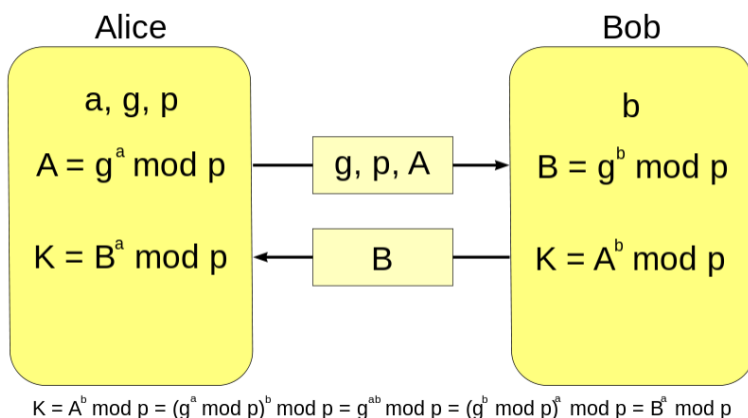
2018

## 1. Цель работы

При помощи алгоритма Диффи-Хеллмана произвести обмен публичными ключами и получить общий секретный ключ.

## 2. Анализ задачи

Криптографическая стойкость алгоритма Диффи — Хеллмана (то есть сложность вычисления  $K = g^{ab} \bmod p$  по известным  $p, g, A = g^a \bmod p$  и  $B = g^b \bmod p$ ), основана на **предполагаемой** сложности задачи дискретного логарифмирования.



## 3. Текст программы

Для удобства программа была разбита на 3 файла:

DH.h – описание класса алгоритма Диффи-Хеллмана

DH.cpp – реализация методов класса

main.cpp – основная программа

### Часть 1. Обычная 64-битная арифметика (uint64\_t)

```
#include <iostream>
#include <random>
#include <cmath>

using std::cin;
using std::cout;
using std::endl;

// Алгоритм быстрого возведения в степень в поле вычета mod
uint64_t fastPow(uint64_t num, uint64_t deg, uint64_t mod) {

    uint64_t result = 1;
    uint64_t bit = num % mod;

    while (deg > 0) {

        if ((deg & 1) == 1) {
            result *= bit;
            result %= mod;
        }

        bit *= bit;
        bit %= mod;
        deg >>= 1;
    }
}
```

```

    }
    return result;
}

// Проверка числа на простоту при помощи решета Эратосфена
bool isPrime(uint64_t num) {
    if (num <= 3) { // 2 и 3 простые
        return num > 1; // а 1 - нет
    }
    else if (num % 2 == 0 || num % 3 == 0) {
        return false;
    }
    else {
        for (int i = 5; i * i <= num; i += 6) {
            if (num % i == 0 || num % (i + 2) == 0) {
                return false;
            }
        }
        return true;
    }
}

// Генерация случайного числа
uint64_t getRandomNumber() {
    static std::mt19937 generator(917401);
    static std::uniform_int_distribution<> distribution(0, INT32_MAX);
    return distribution(generator);
}

// Генерация простого числа длиной LENGTH бит
uint64_t getPrimeNumber() {
    uint64_t n;
    do {
        n = getRandomNumber();
    } while (!isPrime(n));
    return n;
}

// Поиск p и k
void calc_p_and_k(uint64_t &p, uint64_t &k) {
    int i = 0;
    k = 1, p = 4;
    while (!isPrime(p)) {
        k = getPrimeNumber();
        p = 2 * k + 1;
    }
}

// Поиск примитивного корня g
uint64_t calcPrimitiveRoot(uint64_t p, uint64_t k) {
    uint64_t g;
    while (true) {
        g = getPrimeNumber();
        if (fastPow(g, 2, p) == 1 || fastPow(g, k, p) == 1)

```

```

        continue;
    return g;
}

void main() {
    data p, g, B;
    cout << "Enter p: ";
    cin >> p;
    cout << "Enter g: ";
    cin >> g;

    data a = getRandomNumber();
    cout << "a = " << a << endl;
    data A = fastPow(g, a, p);
    cout << "A = " << A << endl;

    cout << "Enter B: ";
    cin >> B;

    data K = fastPow(B, a, p);
    cout << "K = " << K << endl;
}

```

## Часть 2. Длинная арифметика (cpp\_int)

```

#include <iostream>
#include <random>
#include <boost/multiprecision/cpp_int.hpp>
#include <boost/multiprecision/miller_rabin.hpp>
#include <boost/random/mercenne_twister.hpp>
#include <boost/random.hpp>

using namespace boost::multiprecision;
using std::cin;
using std::cout;
using std::endl;

const int ITER = 25;
const int LENGTH = 80;

// Алгоритм быстрого возведения в степень в поле вычета mod
cpp_int fastPow(cpp_int &num, const cpp_int &deg_param, const cpp_int &mod) {
    cpp_int result = 1;
    cpp_int bit = num % mod;
    cpp_int deg = deg_param;

    while (deg > 0) {
        if ((deg & 1) == 1) {
            result *= bit;
            result %= mod;
        }

        bit *= bit;
        bit %= mod;
        deg >>= 1;
    }
    return result;
}

```

```

}

// Генерация случайного числа
cpp_int getRandomNumber() {

    static boost::random::mt19937 generator(917401);
    static boost::random::uniform_int_distribution<cpp_int> distribution(0, cpp_int(1)
<< LENGTH);
    return distribution(generator);
}

// Генерация простого числа длиной LENGTH бит
cpp_int getPrimeNumber() {

    cpp_int n;
    do {
        n = getRandomNumber();
    } while (!miller_rabin_test(n, ITER));
    return n;
}

// Поиск p и k
void calc_p_and_k(cpp_int &p, cpp_int &k) {

    int i = 0;
    k = 1, p = 4;
    while (!miller_rabin_test(p, ITER)) {
        k = getPrimeNumber();
        p = 2 * k + 1;
        cout << ++i << "\r";
    }
    cout << "Iterations count:" << i << endl;
}

// Поиск примитивного корня g
cpp_int calcPrimitiveRoot(cpp_int p, cpp_int k) {

    cpp_int g;
    while (true) {
        g = getPrimeNumber();
        if (fastPow(g, 2, p) == 1 || fastPow(g, k, p) == 1)
            continue;
        return g;
    }
}

void main() {

    data p, g, B;
    cout << "Enter p:" << endl;
    cin >> p;
    cout << "Enter g:" << endl;
    cin >> g;

    data a = getRandomNumber();
    cout << "a:" << endl << a << endl;
    data A = fastPow(g, a, p);
    cout << "A:" << endl << A << endl;
    cout << "Enter B:" << endl;
}

```

```

    cin >> B;

    data K = fastPow(B, a, p);
    cout << "K:" << endl << K << endl;
}

```

## 4. Результат работы программы

### Часть 1. Обычная 64-битная арифметика (uint64\_t)

```

C:\Windows\system32\cmd.exe
Enter p: 3411261299
Enter g: 940277333
a = 354739622
A = 1295378262
Enter B: 2443799809
K = 2603712841
Для продолжения нажмите любую клавишу . . .

```

```

C:\Windows\system32\cmd.exe
Enter p: 2070681707
Enter g: 995195969
a = 799900591
A = 518516593
Enter B: 492818084
K = 1961949497
Для продолжения нажмите любую клавишу . . .

```

```

C:\Windows\system32\cmd.exe
Enter p: 4017671459
Enter g: 1176357661
a = 1173215765
A = 1885957851
Enter B: 500805385
K = 1036142726
Для продолжения нажмите любую клавишу . . .

```

### Часть 2. Длинная арифметика (cpp\_int)

```

C:\Windows\system32\cmd.exe
Enter p:
956803045417312141966376964950512424387431803933463615743129196763030660388748180629144478
0302952997869469284540004322501718974905749003477323779864416763
Enter g:
385954491163178425995015853479388905524797941794928848779334204037640454602301007189904270
0895835968943238295343420056032821904639974681423423898190007233
a:
477858969055741492822238722180413280524613910900774369673625469859480422039605256880102950
582363982183151031872477144055048732338667388134784183794974655996452474067281522135947752
795047588015906575221365818534499988970471270655925310106418686091352486694531968667001990
8912384781875161510905908156144954097
A:
628305180438515738224405573307756694912127875370347993276951584118148303414761586772342306
4360663509439108225696845081210023677864125745455241757455038660
Enter B:
587824669993827902514342629449557891458348432950076212372977368146101294640032141968209931
360778443218488741736854992567954431799087034019814630157453236
K:
768739657828164787517856677414736032127419102801242458226531235200470967703299170022593309
9984672893273261007293385324828133539149097419450175540168413655
Для продолжения нажмите любую клавишу . . .

```

Единственным недостатком оказалось то, что если искать примитивный корень самому, то на это уйдёт 15 минут.

```
C:\Windows\system32\cmd.exe
Iterations count:653
p = 26216760080936952932710979273796884404881020977849428588907337702718364246516757634021
991331163958141390811503602940138042551431995498682320894727938462481808290433919454336437
163994254446187731387597301010712189264183700056115963784521913885060113804838334027667187
7875928583891012619546174594802189215085547
k = 13108380040468476466355489636898442202440510488924714294453668851359182123258378817010
995665581979070695405751801470069021275715997749341160447363969231240904145216959727168218
581997127223093865693798650505356094632091850028057981892260956942530056902419167013833593
8937964291945506309773087297401094607542773
g = 54846190862605656555195313110493575156475076684314099072112484780377002788907313673776
100887341954421552924952891189002607542872187668748797056086866336071698460282400004201443
916635635038011971270956714639218028800797784439685812179310626344503123263479651290101440
019210348792742214518098426973277055694213
883031Ms
Для продолжения нажмите любую клавишу . . .
```

Однако нам может повезти, и тогда мы получим их быстро:

```
C:\Windows\system32\cmd.exe
Iterations count:35
p = 14033601509919599724861245694146770129215673009203846372125715119497895024226513224627
289567674314349730172716698488589332902548945084638225171577174180112234901912280133686740
465293840720550111284696953838159069891563634546782753783339772128501047807586172932719453
3144112342504156419930479696772529742686039
k = 70168007549597998624306228470733850646078365046019231860628575597489475121132566123136
447838371571748650863583492442946664512744725423191125857885870900561174509561400668433702
326469203602750556423484769190795349457818172733913768916698860642505239037930864663597266
572056171252078209965239848386264871343019
g = 12965794539047362235657073033137455643050424967618321162257008786563768926439191522950
914341026934152915186296384191195317163598816631167375349100701697071181103857604622800298
753736812180393468939392740146545960063684623623619414643596352838261849015846307997117180
1334552778773900693415187557263092340844937
46188Ms
Для продолжения нажмите любую клавишу . . .
```

## 5. Выводы

В ходе выполнения 2 лабораторной работы был реализован алгоритм Диффи-Хелмана для генерации общего секретного ключа при помощи библиотеки boost версии 1.68, а точнее модулей длинной арифметики multiprecision и работы со случайными числами random:

[https://www.boost.org/doc/libs/1\\_68\\_0/libs/multiprecision/doc/html/index.html](https://www.boost.org/doc/libs/1_68_0/libs/multiprecision/doc/html/index.html)

[https://www.boost.org/doc/libs/1\\_68\\_0/doc/html/boost\\_random.html](https://www.boost.org/doc/libs/1_68_0/doc/html/boost_random.html)