

МИНОБРНАУКИ РОССИИ

**Федеральное государственное бюджетное образовательное учреждение
высшего образования**

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

**Кафедра теоретических основ
компьютерной безопасности и
криптографии**

Протоколы открытого распределения ключей

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Арбузова Матвея Александровича

Преподаватель

аспирант

подпись, дата

Р. А. Фарахутдинов

Саратов 2023

1 Постановка задачи

Необходимо реализовать протокол открытого распределения ключей, используя алгоритм Хьюза (Hughes).

2 Теоретические сведения

Опр. Если p – простое число, и g меньше, чем p , то g является *примитивом* по отношению к p , если для каждого числа b от 1 до $p - 1$ существует некоторое число a такое, что $g^a \equiv b \pmod{p}$.

В общем случае проверить, является ли данное число примитивом, нелегко. Однако задача упрощается, если известно разложение на множители для $p - 1$. Пусть q_1, q_2, \dots, q_n – это различные множители $p - 1$, для того чтобы g являлся примитивом по модулю p необходимо выполнение следующего условия $g^{q_i} \bmod p \neq 1$, где $q_i = p - 1$. (Другими словами, необходимо чтобы порядок g равнялся $p - 1$).

Одним из вариантов алгоритма Диффи-Хеллмана является алгоритм Хьюза (Hughes), который позволяет Алисе сначала генерировать ключ, проверить его надёжность, и уже потом послать его Бобу.

Общие параметры те же, что и в алгоритме Диффи-Хеллмана: большие простые числа n и g такие, что g является примитивом по модулю n .

Алгоритм Хьюза

Вход: целое число l – битовая длина числа n .

Выход: K – секретный ключ.

Шаг 1. Генерируется простое число n , длиной l бит;

Шаг 2. Генерируется простое число g , которое является примитивом по модулю n .

Шаг 3. Алиса генерирует случайное секретное целое число x из интервала $1 < x < n$;

Шаг 4. Алиса генерирует сеансовый ключ K ;

Шаг 5. Боб генерирует случайное секретное целое число y из интервала $1 < y < n$ с условием $(y, p - 1) = 1$. Кроме того, если p – сильное простое

число, например, вида $p = 2^k q + 1$, тогда y может быть любым большим случайным нечетным числом, кроме q .

Шаг 6. Боб вычисляет $Y = g^y \bmod n$ и посылает его Алисе;

Шаг 7. Алиса вычисляет $X = g^x \bmod n$ и посылает его Бобу;

Шаг 8. Боб вычисляет $z = y^{-1} \bmod (n - 1)$ и $K' = X^z \bmod n$.

Если все выполнено правильно, то $K = K'$.

Преимуществом описанного выше протокола над протоколом Диффи-Хеллмана состоит в том, что K можно вычислить заранее, до какого-либо взаимодействия, и Алиса может зашифровать сообщения с помощью K задолго до установления соединения с Бобом. При этом Алиса может послать сообщение сразу множеству людей, а передать ключ позднее каждому по отдельности.

3 Практическая реализация

3.1 Описание программы

Программа была написана на языке C++, и имеет множество функций.

Функция *main* является точкой старта программы и отвечает за проверку корректности введенной, при запуске программы, длины числа n .

Функция *Hughes* содержит все шаги описанного выше алгоритма, при этом для генерации чисел n и g используются функции *GenN* и *GenG* соответственно, а функции *FirstPass*, *SecondPass*, *ThirdPass*, *FourthPass*, разделяют шаги алгоритма на проходы, в каждом из которых вычисляются необходимые значения.

Проверка чисел на простоту происходит по алгоритму Соловея-Штрассена – функция *SolovSht*, данный алгоритм использует символ Якоби, подсчет значения которого реализован в функции *Jac*.

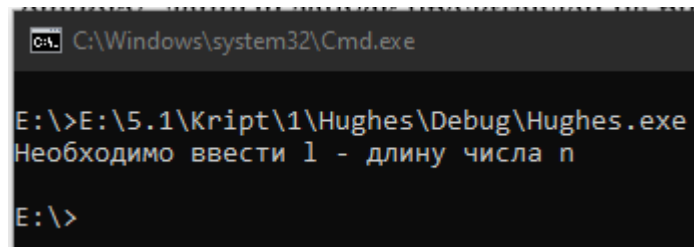
Описанный в теоретической части способ нахождения примитива g по модулю n реализован в функции *Primitive*, которая использует функцию *Divis* для нахождения делителей числа $n - 1$.

На восьмом шаге алгоритма необходимо найти обратный элемент, это позволяет сделать расширенный алгоритм Евклида – функция *ExtendedEuclid*.

В программе используются большие числа, работать с которыми позволяет подключённая библиотека boost. А для лучшей работы с возведением в степень был реализован алгоритм быстрого возведения в степень по модулю – функция *Exponentiation*.

3.2 Результаты тестирования программы

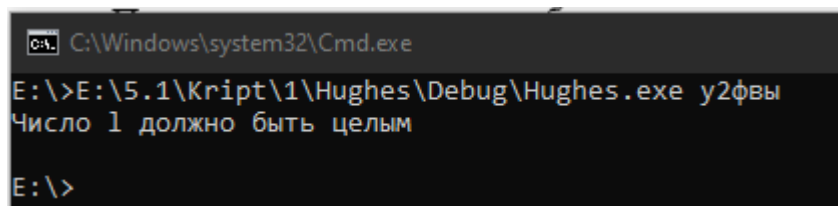
При запуске программы без параметров выведет соответствующую ошибку, данный запуск представлен на рисунке 1.



```
C:\Windows\system32\Cmd.exe
E:\>E:\5.1\Kript\1\Hughes\Debug\Hughes.exe
Необходимо ввести l - длину числа n
E:\>
```

Рисунок 1 – Запуск программы без параметров

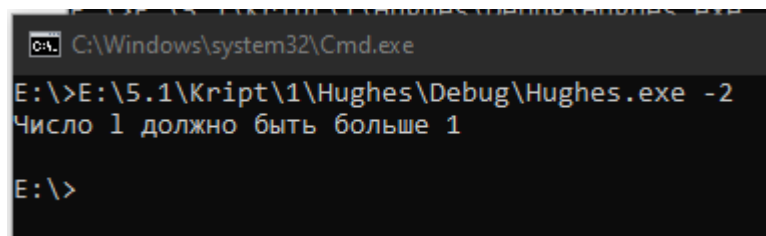
Ввод случайного набора символов, также приводит к ошибке – рисунок 2.



```
C:\Windows\system32\Cmd.exe
E:\>E:\5.1\Kript\1\Hughes\Debug\Hughes.exe y2фвы
Число l должно быть целым
E:\>
```

Рисунок 2 – Запуск программы со случайным набором символов в качестве параметра

Кроме того, *l* должен быть больше 1 – рисунок 3.



```
C:\Windows\system32\Cmd.exe
E:\>E:\5.1\Kript\1\Hughes\Debug\Hughes.exe -2
Число l должно быть больше 1
E:\>
```

Рисунок 3 – Запуск программы с параметром меньше единицы

На рисунках 4-6 представлены успешные запуски программ.

```
C:\Windows\system32\Cmd.exe

E:\>E:\5.1\Kript\1\Hughes\Debug\Hughes.exe 8
n = 227
g = 163
Алиса сгенерировала числа x = 178 и k = 131
Боб сгенерировал y = 137 и послал Алисе Y = 88
Алиса посылает Бобу X = 36
Боб вычислил z = 33 и сгенерировал k' = 131
```

Рисунок 4 – Успешный запуск программы с параметром $l = 8$

```
C:\Windows\system32\Cmd.exe

E:\>E:\5.1\Kript\1\Hughes\Debug\Hughes.exe 25
n = 28947403
g = 17726843
Алиса сгенерировала числа x = 8445054 и k = 27847916
Боб сгенерировал y = 24990079 и послал Алисе Y = 13064997
Алиса посылает Бобу X = 9785231
Боб вычислил z = 17701819 и сгенерировал k' = 27847916
```

Рисунок 5 – Успешный запуск программы с параметром $l = 25$

```
C:\Windows\system32\Cmd.exe

E:\>E:\5.1\Kript\1\Hughes\Debug\Hughes.exe 5
n = 31
g = 3
Алиса сгенерировала числа x = 18 и k = 4
Боб сгенерировал y = 17 и послал Алисе Y = 22
Алиса посылает Бобу X = 16
Боб вычислил z = 23 и сгенерировал k' = 4
```

Рисунок 6 – Успешный запуск программы с параметром $l = 5$

Листинг кода

```
#include <iostream>
#include <random>
#include <boost/multiprecision/cpp_int.hpp>
#include <boost/random/mersenne_twister.hpp>
#include <boost/random/uniform_int.hpp>
#include <boost/random/variante_generator.hpp>
#include <vector>

using namespace std;
using namespace boost::multiprecision;
using namespace boost::random;

vector <cpp_int> ExtendedEuclid(cpp_int a, cpp_int b) {
    vector <cpp_int> res(3);
    if (a == 0) {
        res = { b, 0, 1 };
        return res;
    }
    vector <cpp_int> c = ExtendedEuclid(b % a, a);
    res = { c[0], c[2] - (b / a) * c[1], c[1] };
    return res;
}

cpp_int Exponentiation(cpp_int x, cpp_int n, cpp_int m) {
    cpp_int N = n, Y = 1, Z = x % m;
    while (N != 0) {
        cpp_int lastN = N % 2;
        N = N / 2;
        if (lastN == 0) {
            Z = (Z * Z) % m;
            continue;
        }
        Y = (Y * Z) % m;
        if (N == 0)
            break;
        Z = (Z * Z) % m;
    }
    return Y % m;
}

cpp_int Jac(cpp_int a, cpp_int b) {
    if (ExtendedEuclid(a, b)[0] != 1)
        return 0;
    else {
        int r = 1;
        while (a != 0) {
            cpp_int t = 0;
            while (a % 2 == 0) {
                t = t + 1;
                a = a / 2;
            }
        }
    }
}
```

```

    }
    if (t % 2 != 0)
        if (Exponentiation(b, 1, 8) == 3 || Exponentiation(b, 1, 8)
== 5)
            r = r * (-1);
    if (Exponentiation(a, 1, 4) == 3 && Exponentiation(b, 1, 4) == 3)
        r = r * (-1);
    cpp_int c = a;
    if (c != 0)
        a = Exponentiation(b, 1, c);
    b = c;
}
return r;
}
}

```

```

bool SolovSht(cpp_int p, int k) {
    if (p > 0 && p < 4)
        return true;
    if (p % 2 == 0)
        return false;
    for (int i = 0; i < k; i++) {
        cpp_int a = rand() % (p - 2) + 2;
        if (ExtendedEuclid(p, a)[0] > 1)
            return false;
        cpp_int t = (p - 1) / 2;
        cpp_int newa = Exponentiation(a, t, p);
        cpp_int l = Jac(a, p);
        if (l == -1)
            l = p - 1;
        if (newa != l)
            return false;
    }
    return true;
}

```

```

cpp_int GenN(int l) {
    for (;;) {
        srand(time(0));
        cpp_int n = 1;
        cpp_int deg = 2;
        for (int i = 1; i < l - 1; i++) {
            n = n + (deg * (rand() % 2));
            deg = deg * 2;
        }
        n = n + deg;
        if (SolovSht(n, 10))
            return n;
    }
}

```

```

cpp_int random(cpp_int start, cpp_int p) {
    random_device gen;

```

```

        boost::random::uniform_int_distribution<cpp_int> ui(start, p - 1);
        return ui(gen);
    }

```

```

vector <cpp_int> Divis(cpp_int n){
    vector <cpp_int> res;
    for (cpp_int i = 1; i < n / 2 + 1; i++)
        if (n % i == 0)
            res.push_back(i);
    res.push_back(n);
    return res;
}

```

```

bool Primitive(cpp_int n, cpp_int g) {
    vector <cpp_int> divisors = Divis(n - 1);
    for (int i = 0; i < divisors.size(); i++)
        if (Exponentiation(g, divisors[i], n) == 1)
            if (divisors[i] == n - 1)
                return true;
            else
                break;
    return false;
}

```

```

cpp_int GenG(cpp_int n) {
    cpp_int g;
    for (;;) {
        g = random(0, n);
        if (SolovSht(g, 10) && Primitive(n, g))
            return g;
    }
}

```

```

pair <cpp_int, cpp_int> FirstPass(cpp_int n, cpp_int g) {
    pair <cpp_int, cpp_int> res;
    cpp_int x = random(2, n);
    cpp_int k = Exponentiation(g, x, n);
    res = make_pair(x, k);
    return res;
}

```

```

pair <cpp_int, cpp_int> SecondPass(cpp_int n, cpp_int g) {
    pair <cpp_int, cpp_int> res;
    cpp_int y, q = n - 1;
    while (q % 2 == 0)
        q = q / 2;
    for (;;) {
        y = random(2, n);
        if (ExtendedEuclid(y, n - 1)[0] == 1)
            if (y != q)
                break;
    }
}

```



```

    }
    cpp_int Y = Exponentiation(g, y, n);
    res = make_pair(y, Y);
    return res;
}

cpp_int ThirdPass(cpp_int Y, cpp_int x, cpp_int n) {
    cpp_int X = Exponentiation(Y, x, n);
    return X;
}

pair <cpp_int, cpp_int> FourthPass(cpp_int y, cpp_int X, cpp_int n) {
    pair <cpp_int, cpp_int> res;
    cpp_int z = ExtendedEuclid(y, n - 1)[1];
    while (z < 0)
        z = (z + n - 1);
    cpp_int k = Exponentiation(X, z, n);
    res = make_pair(z, k);
    return res;
}

void Hughes(int l) {
    cpp_int n = GenN(l);
    cout << "n = " << n << "\n";
    cpp_int g = GenG(n);
    cout << "g = " << g << "\n";
    pair <cpp_int, cpp_int> xk = FirstPass(n, g);
    cout << "Алиса сгенерировала числа x = " << xk.first << " и k = " <<
xk.second << "\n";
    pair <cpp_int, cpp_int> yY = SecondPass(n, g);
    cout << "Боб сгенерировал y = " << yY.first << " и послал Алисе Y = " <<
yY.second << "\n";
    cpp_int X = ThirdPass(yY.second, xk.first, n);
    cout << "Алиса посылает Бобу X = " << X << "\n";
    pair <cpp_int, cpp_int> zksh = FourthPass(yY.first, X, n);
    cout << "Боб вычислил z = " << zksh.first << " и сгенерировал k' = " <<
zksh.second << "\n";
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "Russian");
    int l;
    if (argc == 1){
        cerr << "Необходимо ввести l - длину числа n\n";
        return 0;
    }
    try {
        l = stoi(argv[1]);
    }
    catch (std::invalid_argument) {
        cerr << "Число l должно быть целым\n";
        return 0;
    }
}

```

```
    }  
    if (1 < 2) {  
        cerr << "Число 1 должно быть больше 1\n";  
        return 0;  
    }  
    Hughes(1);  
    return 0;  
}
```