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 | \_\_\_\_\_\_\_\_\_\_ | Р. А. Фарахутдинов |
|  | подпись, дата |  |

Саратов 2023

**1 Постановка задачи**

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

**2 Теоретические сведения**

Опр. Если – простое число, и меньше, чем , то является *примитивом* по отношению к , если для каждого числа от до существует некоторое число такое, что .

В общем случае проверить, является ли данное число примитивом, нелегко. Однако задача упрощается, если известно разложение на множители для . Пусть – это различные множители , для того чтобы являлся примитивом по модулю необходимо выполнение следующего условия , где . (Другими словами, необходимо чтобы порядок равнялся ).

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 и такие, что является примитивом по модулю.

**Алгоритм Хьюза**

*Вход*: целое число – битовая длина числа .

*Выход*: – секретный ключ.

Шаг 1. Генерируется просто число , длиною бит;

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

Шаг 3. Алиса генерирует случайное секретное целое число из интервала ;

Шаг 4. Алиса генерирует сеансовый ключ ;

Шаг 5. Боб генерирует случайное секретное целое число из интервала с условием . Кроме того, если – сильное простое число, например, вида , тогда может быть любым большим случайным нечетным числом, кроме .

Шаг 6. Боб вычисляет и посылает его Алисе;

Шаг 7. Алиса вычисляет и посылает его Бобу;

Шаг 8. Боб вычисляет и

Если все выполнено правильно, то .

Преимуществом описанного выше протокола над протоколом Диффи-Хеллмана состоит в том, что можно вычислить заранее, до какого-либо взаимодействия, и Алиса может зашифровать сообщения с помощью 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 является точкой старта программы и отвечает за проверку корректности введённой, при запуске программы, длинны числа .

Функция содержит все шаги описанного выше алгоритма, при этом для генерации чисел и используются функции и соответственно, а функции , , , , разделяют шаги алгоритма на проходы, в каждом из которых вычисляются необходимые значения.

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

Описанный в теоретической части способ нахождения примитива по модулю реализован в функции , которая использует функцию для нахождения делителей числа .

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

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 .

**3.2 Результаты тестирования программы**

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

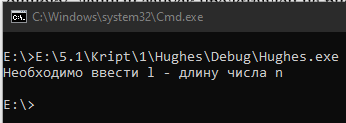


Рисунок 1 – Запуск программы без параметров

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

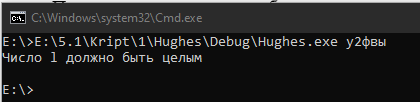


Рисунок 2 – Запуск программы со случайным набором символов в качестве параметра

Кроме того, должен быть больше – рисунок 3.

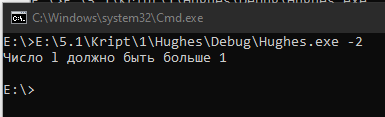


Рисунок 3 – Запуск программы с параметром меньшим единицы

На рисунках 4-6 представлены успешные запуски программ.

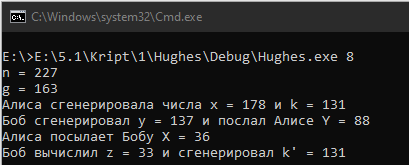


Рисунок 4 – Успешный запуск программы с параметром

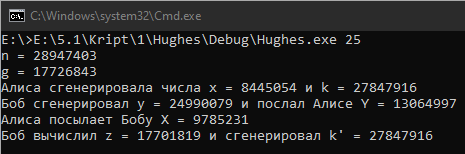


Рисунок 5 – Успешный запуск программы с параметром

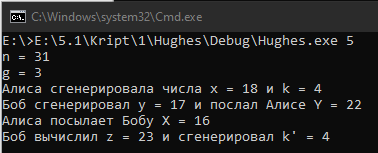


Рисунок 6 – Успешный запуск программы с параметром

**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/variate\_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 (l < 2) {

cerr << "Число l должно быть больше 1\n";

return 0;

}

Hughes(l);

return 0;

}