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

Схемы аутентификации

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 схему идентификации Гиллу-Кискате.

2 Теоретические сведения

Протокол Фейге-Фиата-Шамира был первым, обеспечивающим практическую идентификацию. Этот протокол минимизировал вычисления, увеличивая число итераций и аккредитаций на итерацию. Для некоторых реализаций, например для смарт-карт, такой способ вычислений не слишком подходит. Обмены с внешним миром требуют затрат времени, а хранение данных для каждой аккредитации может быстро исчерпать ограниченные возможности карточки.

Луи Гиллу (Louis Guillou) и Жан-Жак Кискате (Jean-Jacques Quisquater) разработали алгоритм идентификации с нулевым разглашением, более подходящий для подобных приложений. Обмены информацией между Пегги и Виктором, а также параллельные аккредитации в каждом обмене сведены к абсолютному минимуму: для каждого доказательства существует только один обмен, в котором выполняется только одна аккредитация. Чтобы достичь при использовании схемы Гиллу-Кискате того же уровня безопасности, что и в схеме Фейге-Фиата-Шамира, потребуется выполнить в три раза больше вычислений. Так же, как и в случае схемы Фейге-Фиата-Шамира, этот алгоритм идентификации можно превратить в алгоритм цифровой подписи.

Пусть Пегги – собирается доказать свою подлинность Виктору. Идентификация Пегги проводится по строке атрибутов J (которая, зачастую, преобразуется в число с помощью некоторой хэш-функции), при этом J аналогична открытому ключу. Другой открытой информацией, общей для всех, является показатель степени v и модуль n, где n – это произведение двух хранящихся в секрете простых чисел. Закрытым ключом служит значение B, рассчитываемое так, чтобы $JB^v \equiv 1 (mod\ n)$.

Пусть Пегги посылает Виктору свои атрибуты J. Далее Пегги хочет доказать Виктору, что это именно её атрибуты. Для этого она должна убедить

Виктора, что ей известно значение B. Вот какой протокол она для этого использует.

Протокол идентификации Гиллу-Кискате

Вход: Битовая длина простого числа n.

Выход: Виктор убеждается в том, что Пегги известно секретное значение B.

Шаг 1. Выбирается два различных случайных простых числа p и q, после чего вычисляется их произведение $n = pq$;

Шаг 2. Выбирается целое число v с условиями $1 < v < \phi(n)$ и $(\phi(n), v) = 1$, где $\phi(n)$ – функция Эйлера;

Шаг 3. Пегги генерирует $1 < J < n$;

Шаг 4. Вычисляется секрет $B = J^{-s}(mod\, n)$, где $s = v^{-1}(mod\, \phi(n))$;

Шаг 5. Пегги выбирает случайное число r, находящееся в диапазоне от 1 до $n - 1$. Она вычисляет $T = r^v(mod\, n)$ и отправляет его Виктору;

Шаг 6. Виктор выбирает случайное целое d, находящееся в диапазоне от 0 до $v - 1$ и посылает его Пегги;

Шаг 7. Пегги вычисляет $D = rB^d(mod\, n)$ и посылает его Виктору;

Шаг 8. Виктор вычисляет $T' = D^v J^d(mod\, n)$. Если $T \equiv T'(mod\, n)$, то подлинность Пегги доказана.

Используемая здесь математика не слишком сложна:

$T' = D^v J^d = (rB^d)^v J^d = r^v B^{dv} J^d = r^v (JB^v)^d = r^v \equiv T(mod\, n)$, так как B по определению удовлетворяет: $JB^v \equiv 1(mod\, n)$.

Таким образом, протокол позволяет одному участнику доказать другому участнику, что он обладает секретной информацией, не раскрывая ни единого бита этой информации.

Безопасность протокола основана на сложности извлечения квадратного корня по модулю достаточно большого составного числа.

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 введённой, при запуске программы, длинны числа n.

Функция $GuillouQuisquater$ содержит все шаги описанного выше алгоритма, при этом для генерации числа n, равного произведению двух простых чисел p и q используется функция из библиотеки $Crypto++$, генерация чисел v и J происходит с помощью функции $GenJorV$.

В программе используются большие числа, работать с которыми позволяет подключённая библиотека $boost$, кроме того, силами данной библиотеки осуществляется генерация случайных чисел из заданного диапозона в функции $Random$.

Для подсчёта НОД двух целых чисел, а также для поиска обратного элемента в поле используется расширенный алгоритм Евклида – функция $ExtendedEuclid$. Кроме того, была реализована функция быстрого возведения в степень по модулю – $Exponentiation$, часто используемая при подсчётах.

3.2 Результаты тестирования программы

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

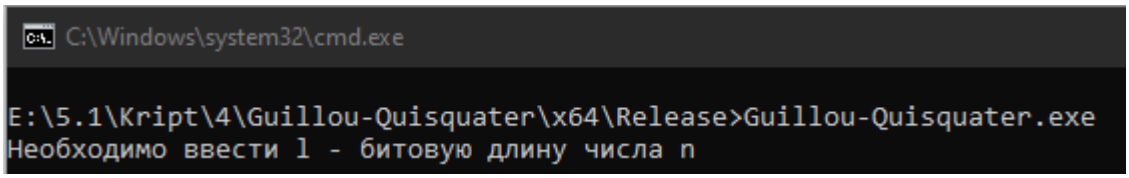

Рисунок 1 – Запуск программы без параметров

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

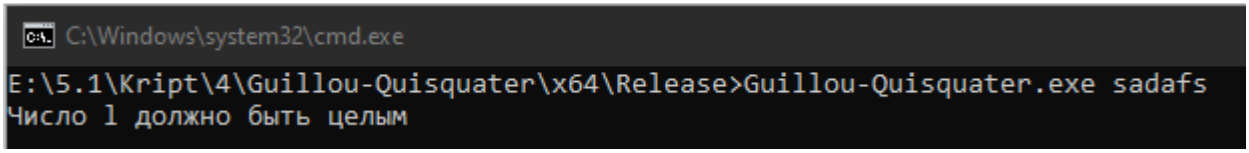

Рисунок 2 – Запуск программы со случайным набором символов в качестве параметра

Кроме того, l должен быть больше 15 – рисунок 3.

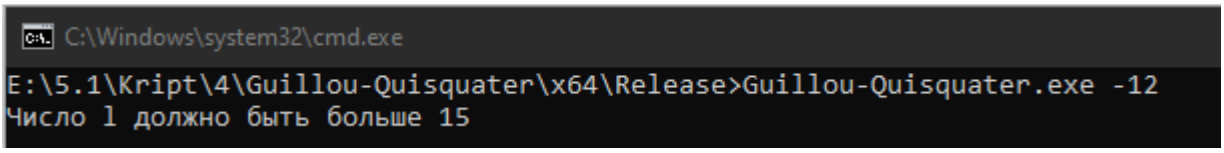

Рисунок 3 – Запуск программы с параметром меньшим семи

На рисунках 4-6 представлены успешные запуски программ.

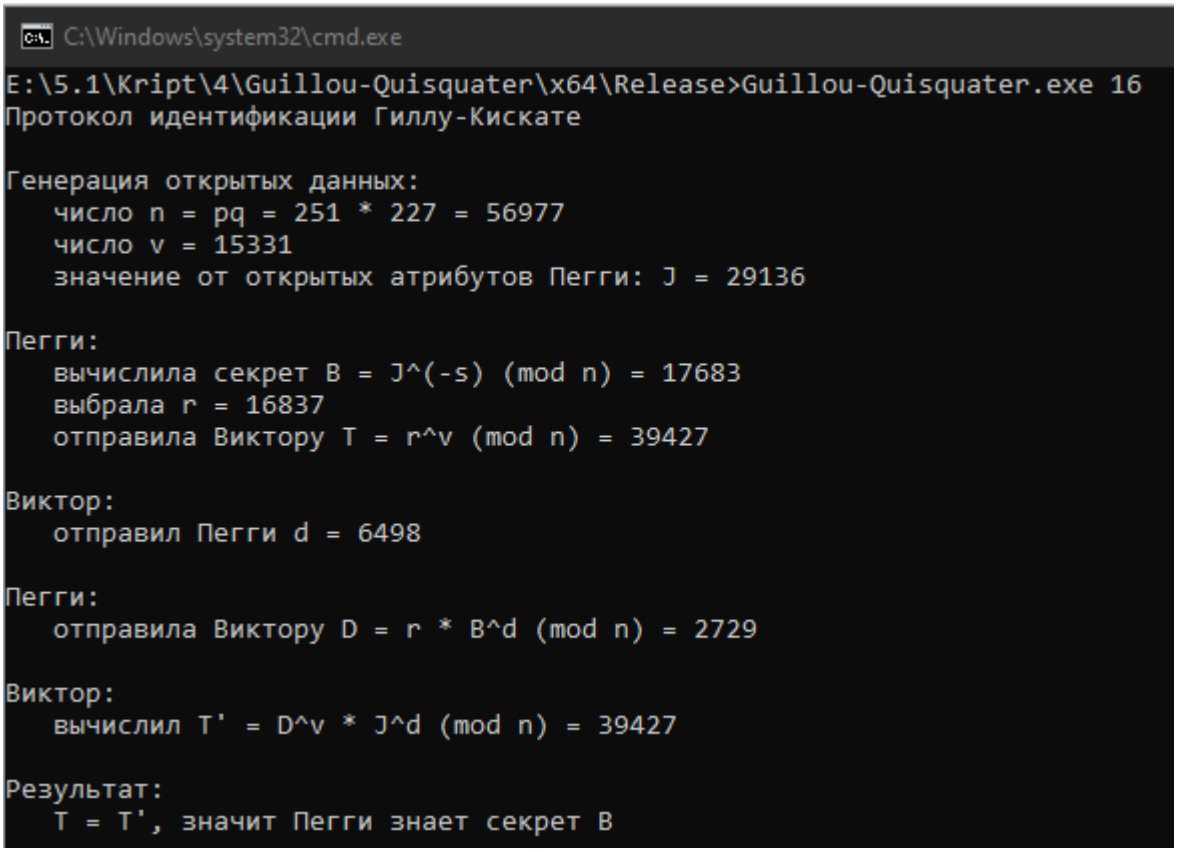

Рисунок 4 – Запуск программы с параметром $l = 16$

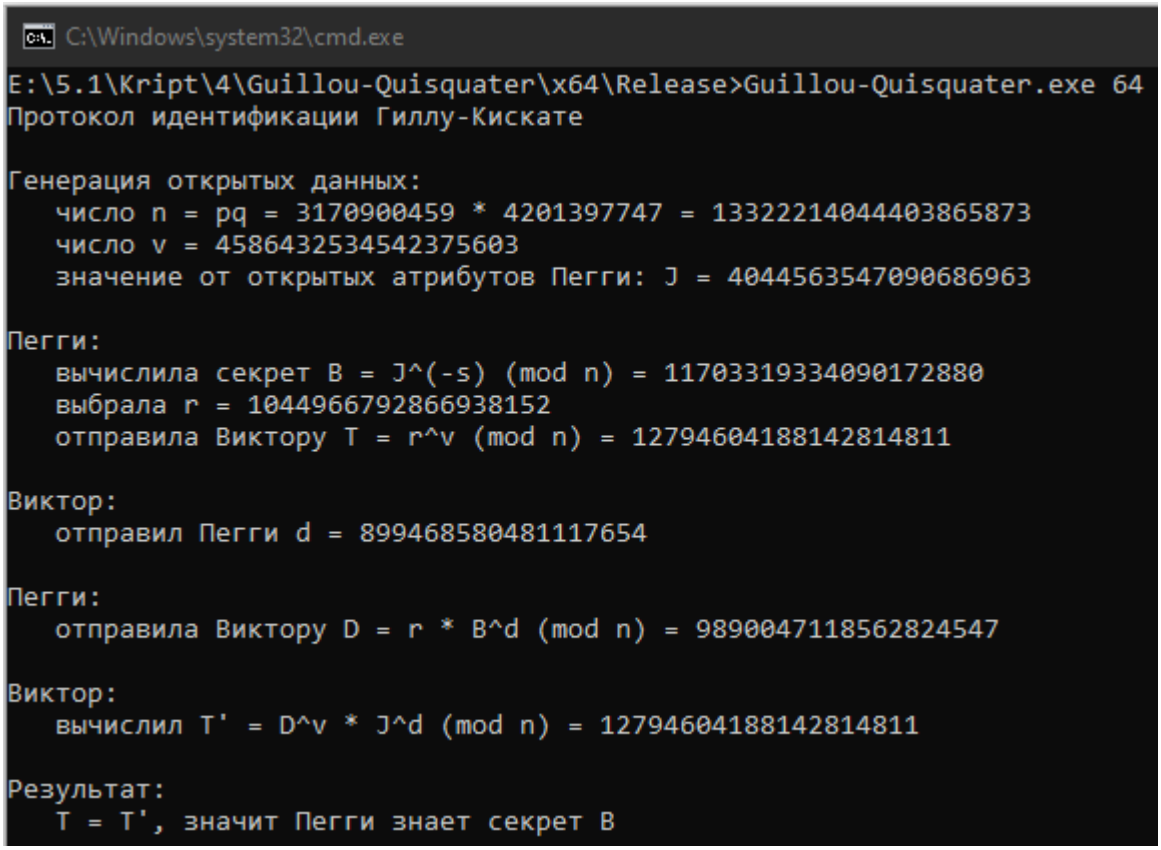

Рисунок 5 – Запуск программы с параметром $l = 64$

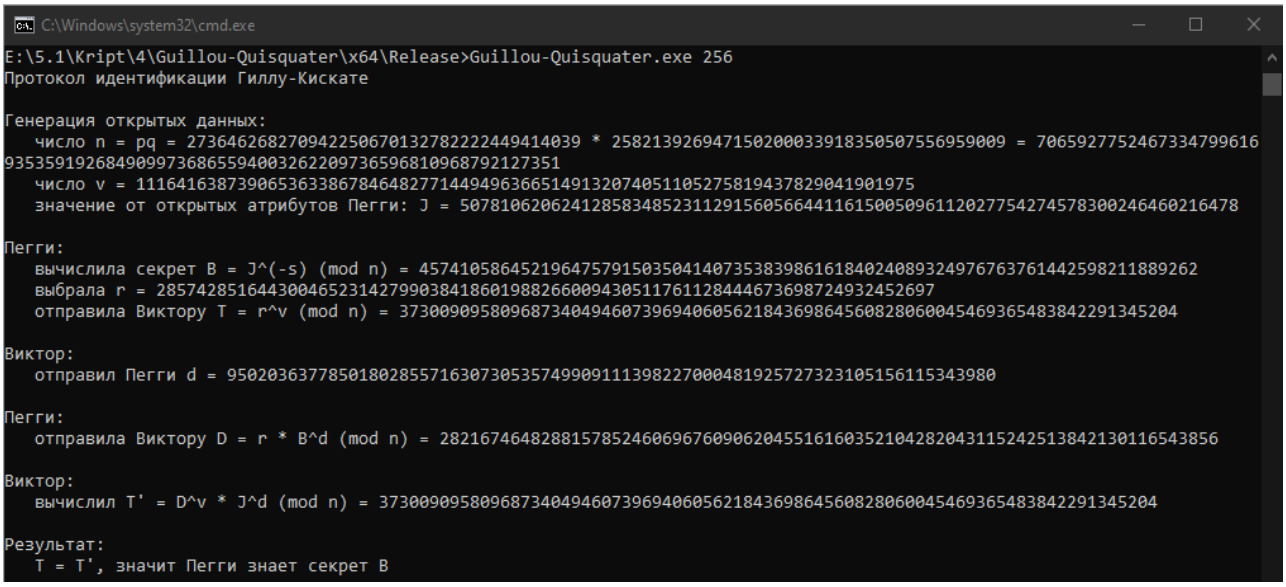

Рисунок 6 – Запуск программы с параметром $l = 256$

Листинг кода

```
#include <iostream>
#include <string>
#include "rsa.h"
#include "osrng.h"
#include <random>
#include <boost/multiprecision/cpp_int.hpp>
#include <boost/random/uniform_int.hpp>
#include <boost/random/variate_generator.hpp>


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


AutoSeededRandomPool rng;


cpp_int ModNegative(cpp_int a, cpp_int p) {
    while (a < 0)
        a = a + p;
    return a % p;
}


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 Random(cpp_int minim, cpp_int maxim) {
    random_device gen;
    boost::random::uniform_int_distribution<cpp_int> ui(minim, maxim);
    return ui(gen);
}


cpp_int IntegerToCppint(const Integer number) {
    ostringstream oss;
    oss << number;
    string str(oss.str());
    str.erase(str.size() - 1, 1);
    cpp_int res(str);
    return res;
}


cpp_int GenJorV(cpp_int m) {
    cpp_int res;
    do {
        res = Random(2, m - 1);
    } while (ExtendedEuclid(res, m)[0] != 1);
    return res;
}


void GuillouQuisquater(int l) {
    //Генерация p, q, n
    cpp_int p, q, n;
    do {
        InvertibleRSAFunction params;
        params.GenerateRandomWithKeySize(rng, l);
        p = IntegerToCppint(params.GetPrime1());
        q = IntegerToCppint(params.GetPrime2());
        n = IntegerToCppint(params.GetModulus());
    } while (p == q);
    cout << "\nГенерация открытых данных:\n   число n = pq = " << p << " * "
<< q << " = " << n << "\n";

    //Генерация v
    cpp_int fiN = (p - 1) * (q - 1);
    cpp_int v = GenJorV(fiN);
    cout << "   число v = " << v << "\n";

    //Генерация J
    cpp_int J = GenJorV(n);
    cout << "   значение от открытых атрибутов Пегги: J = " << J << "\n";

    //Вычисление B
    cpp_int s = ModNegative(ExtendedEuclid(v, fiN)[1], fiN);
    cpp_int obrJ = ModNegative(ExtendedEuclid(J, n)[1], n);
    cpp_int B = Exponentiation(obrJ, s, n);
    cout << "\nПегги:\n   вычислила секрет B = J^(-s) (mod n) = " << B <<
"\n";
```

```
    //Выбор r и вычисление T
    cpp_int r = Random(1, n - 1);
    cout << "   выбрала r = " << r << "\n";
    cpp_int T = Exponentiation(r, v, n);
    cout << "   отправила Виктору T = r^v (mod n) = " << T << "\n";

    //Выбор d
    cpp_int d = Random(0, v - 1);
    cout << "\nВиктор:\n   отправил Пегги d = " << d << "\n";

    //Вычисление D
    cpp_int D = r * Exponentiation(B, d, n) % n;
    cout << "\nПегги:\n   отправила Виктору D = r * B^d (mod n) = " << D <<
"\n";

    //Вычисление T' и проверка T'=D^v J^d (mod n)
    cpp_int Tsh = Exponentiation(D, v, n) * Exponentiation(J, d, n) % n;
    cout << "\nВиктор:\n   вычислил T' = D^v * J^d (mod n) = " << Tsh <<
"\n";
    if (T == Tsh)
        cout << "\nРезультат:\n   T = T', значит Пегги знает секрет B\n";
    else
        cout << "\nРезультат:\n   T != T', значит Пегги не знает секрет B\n";
}


int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "Russian");
    int l;
    if (argc == 1) {
        cerr << "Необходимо ввести l - битовую 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 < 16) {
        cerr << "Число l должно быть больше 15\n";
        return 0;
    }
    cout << "Протокол идентификации Гиллу-Кискате\n";
    GuillouQuisquater(l);
    return 0;
}
```