

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 передачи секретного ключа по открытому каналу

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 трёхпроходный (трехэтапный) протокол Шамира.

2 Теоретические сведения

Этот изобретённый Ади Шамиром, но никогда не опубликованный протокол, позволяет Алисе и Бобу безопасно обмениваться информацией, не используя предварительного обмена ни секретными, ни открытыми ключами. Он предполагает использование коммутативного шифра относительно транспозиции ключей, то есть для которого:

$$E_A(E_B(P)) = E_B(E_A(P)),$$

где использование ключей Алисы обозначается индексом A , а Боба – индексом B .

Алиса хочет передать Бобу секретный ключ K . Для этого Алиса применяет следующий протокол.

Трёхпроходный протокол Шамира

Вход: Битовая длина простого числа p .

Выход: Боб получает от Алисы секретный ключ.

Шаг 1. Алиса шифрует секретный ключ K своим ключом E_A и посылает результат Бобу $C_1 = E_A(K)$;

Шаг 2. Боб шифрует C_1 своим ключом E_B и посылает результат Алисе $C_2 = E_B(C_1) = E_B(E_A(K))$;

Шаг 3. Алиса расшифровывает C_2 своим ключом D_A и посылает результат Бобу $C_3 = D_A(C_2) = D_A(E_B(E_A(K))) = D_A(E_A(E_B(K))) = E_B(K)$;

Шаг 4. Боб расшифровывает C_3 своим ключом D_B , получая $K = D_B(C_3) = D_B(E_B(K))$.

Замечание. Не всякая коммутативная криптосистема сохраняет свою устойчивость в этом протоколе. Одноразовые блокноты обладают свойством коммутативности и обеспечивают абсолютную безопасность, но с описанным

выше протоколом они работать не будут. При использовании одноразового блокнота три шифртекста будут выглядеть следующим образом:

$$C_1 = K \oplus A$$

$$C_2 = K \oplus A \oplus B$$

$$C_3 = K \oplus B$$

Ева, записав все эти три сообщения, которыми обмениваются Алиса и Боб, просто выполнить операцию XOR над всеми этими шифртекстами и восстановит сообщение:

$$C_1 \oplus C_2 \oplus C_3 = (K \oplus A) \oplus (K \oplus A \oplus B) \oplus (K \oplus B) = K.$$

Таким образом, данный способ шифрования работать не будет.

Криптосистема RSA вполне удовлетворяет требованиям этого протокола при условии, что Алиса и Боб использует один и тот же модуль p , а свои пары открытый/закрытый ключ держат секрете.

Ади Шамир и, независимо, Джим Омура, описали алгоритм шифрования, похожий на алгоритм RSA, который будет работать с описываемым протоколом.

Генерация ключей

Генерируется большое безопасное простое число p , которое может быть открытым для группы пользователей.

Функция Эйлера $\phi(p) = p - 1$. Для генерации пары секретных ключей пользователя A выбирается случайное число e с условиями $1 < e < \phi(p)$ и $(\phi(p), e) = 1$. По расширенному алгоритму Евклида вычисляется $d = e^{-1}(\text{mod } \phi(p))$. Число e условно называются *открытым ключом* (или ключом шифрования), число d – *закрытым ключом* (или ключом расшифрования) пользователя A .

Шифрование и расшифрование

Для зашифрования сообщения M вычисляется $C = M^e (\text{mod } p)$.

Для расшифрования сообщения вычисляется $M = C^d (\text{mod } p)$.

Данная криптосистема работает только при условии секретности обоих (открытый/закрытый) ключей, таким образом, она является асимметричной, но

не является криптосистемой с открытым ключом (с публичным ключом). В трёхпроходном протоколе Шамира эта криптосистема работает при дополнительном условии общего модуля p участвующих сторон. В этих условиях Ева не может получить M , не решив проблему дискретного логарифма.

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 простого числа.

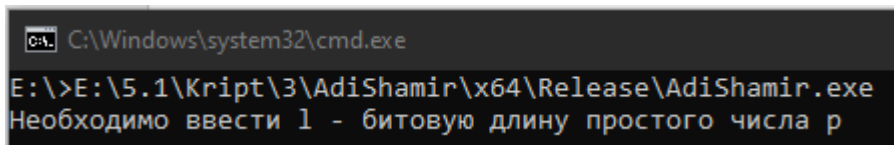
Функция *AdiShamir* содержит все шаги описанного выше алгоритма, при этом для генерации простого числа p используется функция из библиотеки *Crypto++*, генерация ключей E_A , D_A , E_B , D_B происходит с помощью функции *GenKey*.

В программе используются большие числа, работать с которыми позволяет подключённая библиотека *boost*, кроме того, силами данной библиотеки осуществляется генерация секретного ключа K 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*, которая участвует в шифровании и расшифровании.

3.2 Результаты тестирования программы

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



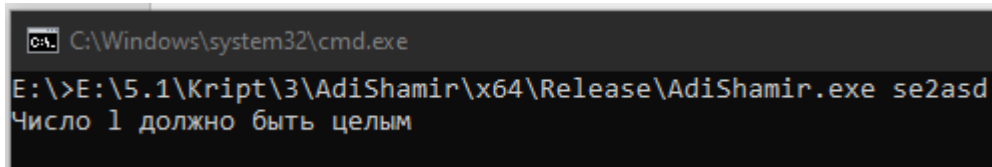
```

C:\Windows\system32\cmd.exe
E:\>E:\5.1\Kript\3\AdiShamir\x64\Release\AdiShamir.exe
Необходимо ввести l - битовую длину простого числа p

```

Рисунок 1 – Запуск программы без параметров

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



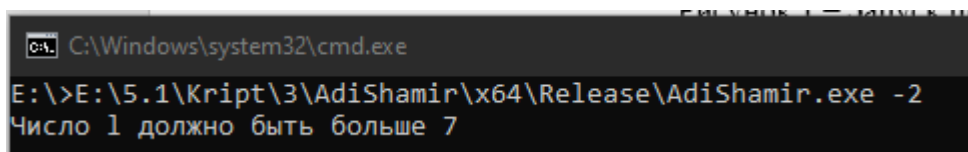
```

C:\Windows\system32\cmd.exe
E:\>E:\5.1\Kript\3\AdiShamir\x64\Release\AdiShamir.exe se2asd
Число l должно быть целым

```

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

Кроме того, l должен быть больше 7 – рисунок 3.



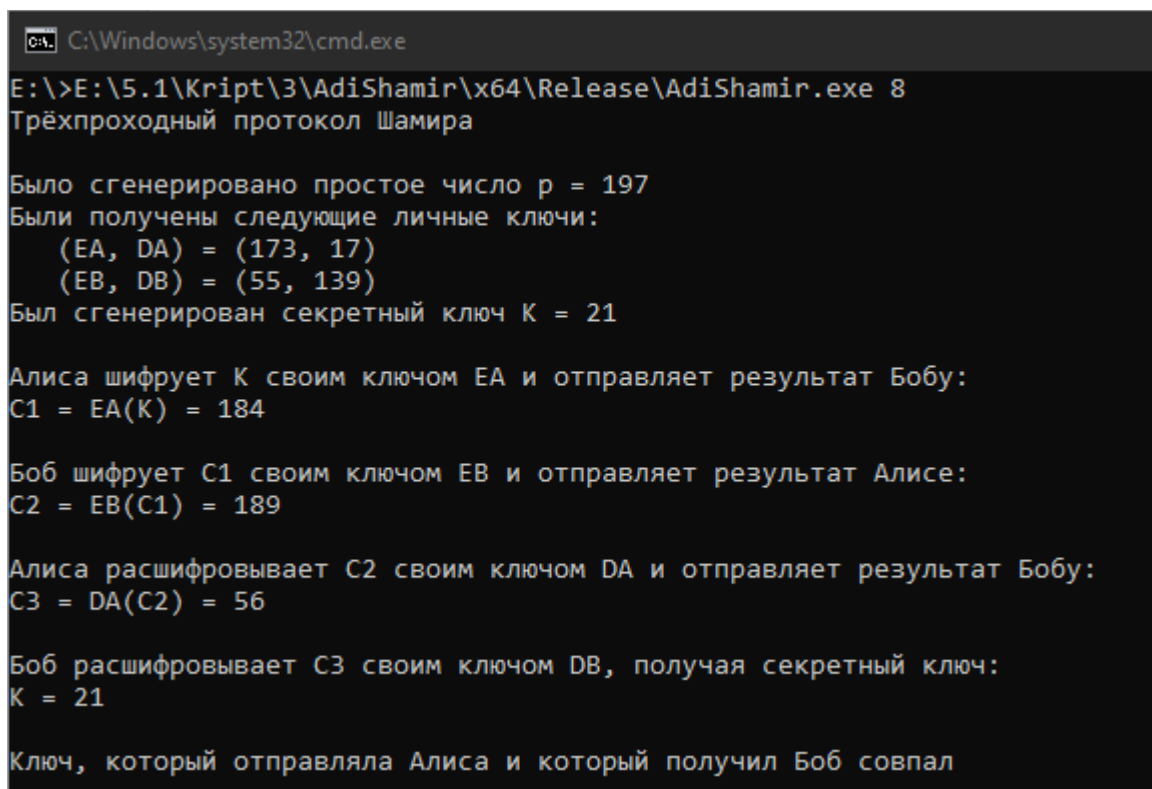
```

C:\Windows\system32\cmd.exe
E:\>E:\5.1\Kript\3\AdiShamir\x64\Release\AdiShamir.exe -2
Число l должно быть больше 7

```

Рисунок 3 – Запуск программы с параметром меньшим семи

На рисунках 4-6 представлены успешные запуски программ.



```

C:\Windows\system32\cmd.exe
E:\>E:\5.1\Kript\3\AdiShamir\x64\Release\AdiShamir.exe 8
Трёхпроходный протокол Шамира

Было сгенерировано простое число p = 197
Были получены следующие личные ключи:
    (EA, DA) = (173, 17)
    (EB, DB) = (55, 139)
Был сгенерирован секретный ключ K = 21

Алиса шифрует K своим ключом EA и отправляет результат Бобу:
C1 = EA(K) = 184

Боб шифрует C1 своим ключом EB и отправляет результат Алисе:
C2 = EB(C1) = 189

Алиса расшифровывает C2 своим ключом DA и отправляет результат Бобу:
C3 = DA(C2) = 56

Боб расшифровывает C3 своим ключом DB, получая секретный ключ:
K = 21

Ключ, который отправляла Алиса и который получил Боб совпал

```

Рисунок 4 – Запуск программы с параметром $l = 8$

```
C:\Windows\system32\cmd.exe
E:\>E:\5.1\Kript\3\AdiShamir\x64\Release\AdiShamir.exe 55
Трёхпроходный протокол Шамира

Было сгенерировано простое число p = 33200275027398403
Были получены следующие личные ключи:
    (EA, DA) = (28471652589198821, 625142217088967)
    (EB, DB) = (25576799517933535, 27385556931868249)
Был сгенерирован секретный ключ K = 7715584510767980

Алиса шифрует K своим ключом EA и отправляет результат Бобу:
C1 = EA(K) = 23830835784856647

Боб шифрует C1 своим ключом EB и отправляет результат Алисе:
C2 = EB(C1) = 29727698051704330

Алиса расшифровывает C2 своим ключом DA и отправляет результат Бобу:
C3 = DA(C2) = 10804265337780463

Боб расшифровывает C3 своим ключом DB, получая секретный ключ:
K = 7715584510767980

Ключ, который отправляла Алиса и который получил Боб совпал
```

Рисунок 5 – Запуск программы с параметром $l = 55$

```
C:\Windows\system32\cmd.exe
E:\>E:\5.1\Kript\3\AdiShamir\x64\Release\AdiShamir.exe 512
Трёхпроходный протокол Шамира

Было сгенерировано простое число p = 11888200432395570849881867726638835046693460794983585738318722561501505759767213941
048694083470671528226752652121673399728344108439016697307151638545869931
Были получены следующие личные ключи:
    (EA, DA) = (298470810584008203736597810389582759108363264374297751497459408391690115853448537272524470052760777890699
332728676454309758694354374497401358821335828927, 3870285934477925223754939152673879753800257958974726658596114276749300
046451227602383296736008864709248539262653454868432096516384425690497431985890945723)
    (EB, DB) = (707289128759182392869920099034246275632909152576882164813073872649114565364045645682227429737667376383130
5423548659138078222978066640027835844186534530991, 509795454995698430347398795929167743917382155891036100365500181921387
2795312575528415400472094101623613780414893748150989235527913823407775945030797370301)
Был сгенерирован секретный ключ K = 635681048972238014116531018738887272288537375111479562441162948775105197394160597094
237529961245501000724538277823891726752239572402173095248769670644956

Алиса шифрует K своим ключом EA и отправляет результат Бобу:
C1 = EA(K) = 42267336612598978620156171566864105072811058888336172007183754258734894169422214618756436923521662926093193
52582835852225566595037311148759390396325374477

Боб шифрует C1 своим ключом EB и отправляет результат Алисе:
C2 = EB(C1) = 1060155571383073566130549609525698204236237887324128355992114996293749101414769487450936476967676481417051
6883134140997780757397278230378747703868482355044

Алиса расшифровывает C2 своим ключом DA и отправляет результат Бобу:
C3 = DA(C2) = 1160269396930028258537245305122450214753015887730858480842315072724538806976753564962436735383155468538808
4100761652574975392633094942483624834862027977844

Боб расшифровывает C3 своим ключом DB, получая секретный ключ:
K = 63568104897223801411653101873888727228853737511147956244116294877510519739416059709423752996124550100072453827782389
17267522239572402173095248769670644956

Ключ, который отправляла Алиса и который получил Боб совпал
```

Рисунок 6 – Запуск программы с параметром $l = 512$

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/variante_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);
}

pair <cpp_int, cpp_int> GenKey(cpp_int p) {
    cpp_int publicKey = Random(2, p - 2);
    while (ExtendedEuclid(publicKey, p - 1)[0] != 1)
        publicKey = Random(2, p - 2);
    cpp_int privateKey = ModNegative(ExtendedEuclid(publicKey, p - 1)[1], p -
1);
    return make_pair(publicKey, privateKey);
}

void AdiShamir(int l) {
    //Генерация p
    InvertibleRSAFunction params;
    params.GenerateRandomWithKeySize(rng, l*2);
    const Integer& pstr = params.GetPrime1();
    ostringstream oss;
    oss << pstr;
    string s(oss.str());
    s.erase(s.size() - 1, 1);
    cpp_int p(s);
    cout << "\nБыло сгенерировано простое число p = " << p << "\n";

    //Генерация ключей для шифрования и расшифрования
    pair<cpp_int, cpp_int> keys = GenKey(p);
    cpp_int EA = keys.first;
    cpp_int DA = keys.second;
    keys = GenKey(p);
    cpp_int EB = keys.first;
    cpp_int DB = keys.second;
    cout << "Были получены следующие личные ключи:\n";
    cout << "    (EA, DA) = (" << EA << ", " << DA << ")\n";
    cout << "    (EB, DB) = (" << EB << ", " << DB << ")\n";

    // Генерация секретного ключа
    cpp_int K = Random(2, p - 1);
    cout << "Был сгенерирован секретный ключ K = " << K << "\n";

    //1
    cout << "\nАлиса шифрует K своим ключом EA и отправляет результат
Бобу:\n";
    cpp_int C1 = Exponentiation(K, EA, p);
    cout << "C1 = EA(K) = " << C1 << "\n";

    //2
    cout << "\nБоб шифрует C1 своим ключом EB и отправляет результат
Алисе:\n";
    cpp_int C2 = Exponentiation(C1, EB, p);

```



```

    cout << "C2 = EB(C1) = " << C2 << "\n";

    //3
    cout << "\nАлиса расшифровывает C2 своим ключом DA и отправляет результат
Бобу:\n";
    cpp_int C3 = Exponentiation(C2, DA, p);
    cout << "C3 = DA(C2) = " << C3 << "\n";

    //4
    cout << "\nБоб расшифровывает C3 своим ключом DB, получая секретный
ключ:\n";
    cpp_int res = Exponentiation(C3, DB, p);
    cout << "K = " << res << "\n";

    //Проверка
    if (res != K)
        cout << "\nБоб получил неверное сообщение\n";
    else
        cout << "\nКлюч, который отправляла Алиса и который получил Боб
совпал\n";
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "Russian");
    int l;
    if (argc == 1) {
        cerr << "Необходимо ввести l - битовую длину простого числа p\n";
        return 0;
    }
    try {
        l = stoi(argv[1]);
    }
    catch (std::invalid_argument) {
        cerr << "Число l должно быть целым\n";
        return 0;
    }
    if (l < 8) {
        cerr << "Число l должно быть больше 7\n";
        return 0;
    }
    cout << "Трёхпроходный протокол Шамира\n";
    AdiShamir(l);
    return 0;
}

```