

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  
Яхина Шамиля Илдусовича

Преподаватель

аспирант

\_\_\_\_\_

Р. А. Фарахутдинов

подпись, дата

Саратов 2023

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 Теоретические сведения .....	4
1.1 Описание алгоритма протокола «станция-станция» .....	4
2 Практическая реализация .....	5
2.1 Описание программы .....	5
2.2 Тестирование программы.....	5

## **ВВЕДЕНИЕ**

Цель работы - реализация протокола открытого распределения ключей «станция-станция».

## 1 Теоретические сведения

Обмен ключами Диффи-Хеллмана, так же как и протокол Хьюза, чувствителен к атаке «человек посередине». Одним из способов предотвратить такую атаку является подпись Алисой и Бобом сообщений, которые они посылают друг другу. Применяемый при этом протокол предполагает, что у Алисы есть открытый ключ Боба, а у Боба есть открытый ключ Алисы. Протокол имеет графическую схему обычного трёхпроходного протокола.

### 1.1 Описание алгоритма протокола «станция-станция»

Алгоритм протокола «станция-станция».

Вход: Целые числа  $p_l, a_l, b_l$ , где  $p_l$  - битовая длина модуля  $p$ , по которому создается циклическая группа,  $a_l$  - битовая длина ключа Алисы для подписи,  $b_l$  - битовая длина ключа Боба для подписи.

Выход: Секретный ключ  $K$ .

1. Алиса передает Бобу  $\{X = g^x \bmod p\}$ , где  $1 < x < p$ ,  $x$  - секретное большое число Алисы;
2.
  - а) Боб выбирает случайное секретное большое число  $y (1 < y < p)$ ;
  - б) Боб вычисляет  $Y = g^y \bmod p$  и  $K = X^y \bmod p$ ;
  - в) Боб подписывает  $X$  и  $Y$ , вычисляя подпись  $S_B(X, Y)$ ;
  - г) Боб шифрует подпись  $E_K(S_B(X, Y))$  ключом  $K$ ;
  - д) Боб передает Алисе  $\{Y, E_K(S_B(X, Y))\}$ .
3.
  - а) Алиса вычисляет  $K = Y^X \bmod p$ ;
  - б) Алиса расшифровывает  $D_K(E_K(S_B(X, Y))) = S_B(X, Y)$  подпись Боба и проверяет ее. Если подпись верна, то протокол продолжается;
  - в) Алиса вычисляет свою подпись  $S_A(X, Y)$  и шифрует ее  $E_K(S_A(X, Y))$ ;
  - г) Алиса передает Бобу  $E_K(S_A(X, Y))$ .

Боб расшифровывает и проверяет подпись Алисы, если она верна, то протокол заканчивается положительно.

## 2 Практическая реализация

### 2.1 Описание программы

Все шаги алгоритма происходят в функции *stsFunc*.

Функция *generateKeys* генерирует публичный и приватный ключ для пользователя. Приватный ключ *pc* выбирается случайным числом от 1 до *p*. Публичным ключом будет являться тройка *y, g, p*, где  $y = g^{pc} \bmod p$ .

Для подписи сообщения используется функция *ElGamalSignatureSigning*, которая реализует подпись сообщения по алгоритму Эль-Гамала.

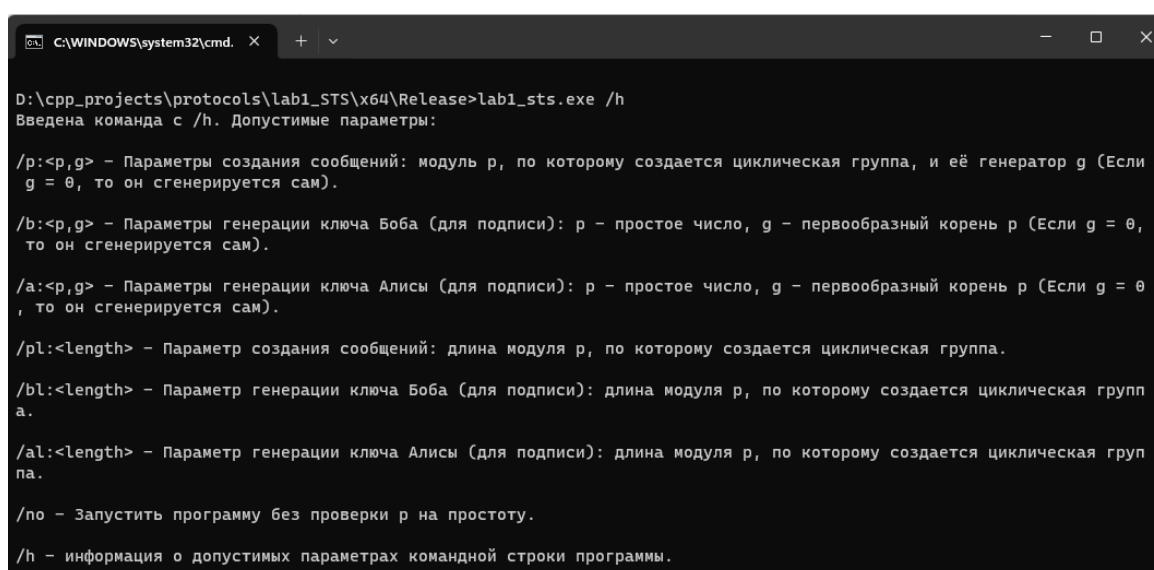
В функции *generateAESKeyFromK* происходит преобразование ключа *K* в подходящий для AES-шифрования ключ. Само AES-шифрование реализовано в функции *encryptAES*, а дешифрование в функции *decryptAES*.

Функция *ElGamalSignatureVerifying* проверяет подпись и если подпись некорректна, то протокол завершается.

Описание дополнительных функций, используемых в программе: *powMod* - функция возведения числа в степень по модулю, *isPrimeSoloveyShtrassen* - проверка числа на простоту тестом Соловея-Штрассена, *jacobiSymbol* - подсчет символа Якоби, *findGen* - нахождение первообразного корня.

### 2.2 Тестирование программы

На рисунке 1 показан вызов параметра *help*, который выводит информацию о допустимых параметрах командной строки программы.



```
C:\WINDOWS\system32\cmd. X + v
D:\cpp_projects\protocols\lab1_STS\x64\Release>lab1_sts.exe /h
Введена команда с /h. Допустимые параметры:

/p:<p,g> - Параметры создания сообщений: модуль p, по которому создается циклическая группа, и её генератор g (Если g = 0, то он сгенерируется сам).

/b:<p,g> - Параметры генерации ключа Боба (для подписи): p - простое число, g - первообразный корень p (Если g = 0, то он сгенерируется сам).

/a:<p,g> - Параметры генерации ключа Алисы (для подписи): p - простое число, g - первообразный корень p (Если g = 0, то он сгенерируется сам).

/pl:<length> - Параметр создания сообщений: длина модуля p, по которому создается циклическая группа.

/bl:<length> - Параметр генерации ключа Боба (для подписи): длина модуля p, по которому создается циклическая группа.

/al:<length> - Параметр генерации ключа Алисы (для подписи): длина модуля p, по которому создается циклическая группа.

/no - Запустить программу без проверки p на простоту.

/h - информация о допустимых параметрах командной строки программы.
```

Рисунок 1 – Вызов параметра *help*

В примере, показанном на рисунках 2 и 3, задаются параметры  $p = 20$ ,  $bl = 20$ ,  $al = 20$ .

```
D:\cpp_projects\protocols\lab1_STS\к64\Release>lab1_sts.exe /pl:20 /bl:20 /al:20

Демонстрация работы протокола "станция-станция"

Параметры создания сообщений:
p = 658449
g = 17

Сгенерированные открытый и закрытый ключи для схемы Эль-Гамала:
Боб:
Открытый ключ (y, g, p) = (810437, 3, 936737)
Закрытый ключ x = 489647
Алиса:
Открытый ключ (y, g, p) = (72439, 5, 225983)
Закрытый ключ x = 2830

АЛИСА:

> Сгенерировала секретное большое число x = 49419
> Вычислила Y = g^x mod p = 104487
> Отправила Бобу X
```

Рисунок 2 – Первый пример корректной работы протокола «станция-станция»

```
C:\WINDOWS\system32\cmd. X + v

БОБ:

> Сгенерировал секретное большое число y = 309503
> Вычислил Y = g^y mod p = 112380
> Вычислил сеансовый ключ K = X^y mod p = 28124
> Вычислил подпись сообщения "X,Y" = (r, s) = (236507, 868342)
> Получил свой ключ для AES-шифрования из сеансового ключа: AESKey = C4843E80233B1AF3D98C28B0F21E36AA540725B0ABDD37CFB81F928C09788B64
> Зашифровал подпись сообщения 236507,868342 и получил &@:ï!T"?и",ÿ@
> Отправил Алисе Y и зашифрованную подпись

АЛИСА:

> Вычислила сеансовый ключ K = Y^x mod p = 28124
> Получила свой ключ для AES-шифрования из сеансового ключа: AESKey = C4843E80233B1AF3D98C28B0F21E36AA540725B0ABDD37CFB81F928C09788B64
> Расшифровала сообщение &@:ï!T"?и",ÿ@ и получила 236507,868342
> Проверила подпись Боба. Она верна. Боб подтвердил свою личность
> Вычислила подпись сообщения "X,Y" = (r, s) = (133064, 73967)
> Зашифровала подпись сообщения 236507,868342 и получила &@:ï!T"?и",ÿ@
> Отправила Бобу зашифрованную подпись

БОБ:

> Расшифровал сообщение &@:ï!T"?и",ÿ@ и получил 236507,868342
> Проверил подпись Алисы. Она верна. Алиса подтвердила свою личность
```

Рисунок 3 – Первый пример корректной работы протокола «станция-станция»

В примере, показанном на рисунках 4 и 5, задаются параметры  $p = 101$ ,  $0$ ,  $bl = 20$ ,  $al = 20$ .

```
C:\WINDOWS\system32\cmd. X + v

D:\cpp_projects\protocols\lab1_STS\x64\Release>lab1_sts.exe /p:101,0 /bl:20 /al:20

Демонстрация работы протокола "станция-станция"

Параметры создания сообщений:
p = 101
g = 2

Сгенерированные открытый и закрытый ключи для схемы Эль-Гамала:
Боб:
Открытый ключ (y, g, p) = (71277, 7, 82231)
Закрытый ключ x = 70717
Алиса:
Открытый ключ (y, g, p) = (532890, 3, 757327)
Закрытый ключ x = 467067

АЛИСА:
> Сгенерировала секретное большое число x = 59
> Вычислила X = g^x mod p = 94
> Отправила Бобу X
```

Рисунок 4 – Второй пример работы протокола «станция-станция»

```
C:\WINDOWS\system32\cmd. X + v

БОБ:
> Сгенерировал секретное большое число y = 60
> Вычислил Y = g^y mod p = 87
> Вычислил сеансовый ключ K = X^y mod p = 36
> Вычислил подпись сообщения "X,Y" = (r, s) = (72369, 68308)
> Получил свой ключ для AES-шифрования из сеансового ключа: AESkey = C2356069E9D1E79CA924378153CFBFB4D4416B1F99D4
1A2940BFDB66C5319DB
> Зашифровал подпись сообщения 72369,68308 и получил ?Нкст~э:ЛЈГ2н6
> Отправил Алисе Y и зашифрованную подпись

АЛИСА:
> Вычислила сеансовый ключ K = Y^x mod p = 36
> Получила свой ключ для AES-шифрования из сеансового ключа: AESkey = C2356069E9D1E79CA924378153CFBFB4D4416B1F99D
41A2940BFDB66C5319DB
> Расшифровала сообщение ?Нкст~э:ЛЈГ2н6 и получила 72369,68308
> Проверила подпись Боба. Она верна. Боб подтвердил свою личность
> Вычислила подпись сообщения "X,Y" = (r, s) = (104461, 455221)
> Зашифровала подпись сообщения 72369,68308 и получила ?Нкст~э:ЛЈГ2н6
> Отправила Бобу зашифрованную подпись

БОБ:
> Расшифровал сообщение ?Нкст~э:ЛЈГ2н6 и получил 72369,68308
> Проверил подпись Алисы. Она верна. Алиса подтвердила свою личность
```

Рисунок 5 – Второй пример работы протокола «станция-станция»

На рисунке 6 показан вызов программы с некорректными параметрами, в качестве простого числа  $p$  передано составное число.

```
D:\cpp_projects\protocols\lab1_STS\x64\Release>lab1_sts.exe /p:100,0 /bl:20 /al:20

ERROR! (один из p не является простым числом)
```

Рисунок 6 – Некорректный вызов функции

## ПРИЛОЖЕНИЕ А

### Листинг программы

```
#include <iostream>
#include <vector>
#include <chrono>
#include <time.h>
#include <boost/random/random_device.hpp>
#include <boost/multiprecision/cpp_int.hpp>
#include <boost/random.hpp>
#include <sstream>
#include <fstream>
#include <unordered_map>
#include <string>
#include <windows.h>
#include <cryptlib.h>
#include "rijndael.h"
#include "modes.h"
#include "files.h"
#include "osrng.h"
#include "hex.h"
#include <unordered_set>

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

const int AES_KEY_SIZE = AES::DEFAULT_KEYLENGTH;
const int AES_BLOCK_SIZE = AES::BLOCKSIZE;

// Функция для преобразования числа K в строку (предполагается, что K - целое число)
std::string intToHexString(cpp_int K) {
    std::ostringstream stream;
    stream << std::hex << K;
    return stream.str();
}

// Функция для генерации ключа на основе числа K
string generateAESKeyFromK(cpp_int K) {
    string KString = intToHexString(K);
    SHA256 hash;
    byte digest[SHA256::DIGESTSIZE];
    hash.CalculateDigest(digest, (const byte*)KString.c_str(), KString.length());
    string hexKey;
    HexEncoder encoder(new StringSink(hexKey));
    encoder.Put(digest, sizeof(digest));
    encoder.MessageEnd();
    return hexKey;
}

string encryptAES(const string& plainText, const string& hexKey) {
    SecByteBlock key((const byte*)hexKey.data(), AES_BLOCK_SIZE);
    ECB_Mode<AES>::Encryption encryptor;
    encryptor.SetKey(key, key.size());
    string cipherText;
    StringSource(plainText, true, new StreamTransformationFilter(encryptor, new
StringSink(cipherText)));
    return cipherText;
}
```



```

string decryptAES(const string& cipherText, const string& hexKey) {
    SecByteBlock key((const byte*)hexKey.data(), AES_BLOCK_SIZE);
    ECB_Mode<AES>::Decryption decryptor;
    decryptor.SetKey(key, key.size());
    string decryptedText;
    StringSource(cipherText, true, new StreamTransformationFilter(decryptor, new
StringSink(decryptedText)));
    return decryptedText;
}

cpp_int HashFunc(const std::string& strXY, cpp_int p) {
    SHA256 hash;
    byte digest[SHA256::DIGESTSIZE];
    hash.CalculateDigest(digest, reinterpret_cast<const byte*>(strXY.c_str()),
strXY.length());
    cpp_int hashValue = 0;
    for (int i = 0; i < SHA256::DIGESTSIZE; ++i) {
        hashValue = (hashValue << 8) | digest[i];
    }
    return hashValue % p;
}

cpp_int rand_large(cpp_int w1, cpp_int w2) {
    random_device gen;
    uniform_int_distribution<cpp_int> ui(w1, w2);
    cpp_int y = ui(gen);
    return y;
}

cpp_int generate_random(cpp_int a, cpp_int b) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<cpp_int> dist(a, b);
    return dist(gen);
}

cpp_int rand_large_by_bit_length(int bit_length) {
    random_device gen;
    uniform_int_distribution<int> ui(0, 1);
    cpp_int result = 0;
    for (int i = 0; i < bit_length; ++i) {
        result <<= 1;
        result |= ui(gen);
    }
    return result;
}

cpp_int powModOld(cpp_int a, cpp_int b, cpp_int mod) {
    cpp_int result = 1;
    a = a % mod;
    while (b > 0) {
        if (b % 2 == 1)
            result = (result * a) % mod;
        b = b >> 1; // equivalent to exponent /= 2
        a = (a * a) % mod;
    }
    return result;
}

cpp_int powMod(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 nod(cpp_int a, cpp_int m) {
    if (m == 0)
        return a;
    else
        return nod(m, a % m);
}

cpp_int algEuclidExtended(cpp_int a, cpp_int b, cpp_int& x, cpp_int& y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    cpp_int xi, yi;
    cpp_int nod = algEuclidExtended(b % a, a, xi, yi);
    x = yi - (b / a) * xi;
    y = xi;
    return nod;
}

cpp_int jacobi_symbol(cpp_int a, cpp_int n) {
    cpp_int dop_n = n;
    if (nod(a, n) != 1)
        return 0;
    cpp_int r = 1;
    if (a < 0) {
        a = a * (-1);
        if (n % 4 == 3) {
            r = r * (-1);
        }
    }
    while (a != 0) {
        cpp_int t = 0;
        while (a % 2 == 0) {
            t = t + 1;
            a = a / 2;
        }
        if (t % 2 != 0) {
            if (n % 8 == 3 || n % 8 == 5)
                r = r * (-1);
        }
        if (a % 4 == 3 && n % 4 == 3) {
            r = r * (-1);
        }
        cpp_int c = a;
        a = n % c;
        n = c;
    }
    if (r < 0)
        r += dop_n;
}

```

```

    return r;
}

bool isPrimeSolovey_shtrassen(cpp_int n, cpp_int k) {
    //cout << "Проверка на простоту с помощью теста Соловея-Штрассена: " << endl;
    for (int i = 0; i < k; i++)
    {
        cpp_int a = rand() % n;
        if (a == 0 || a == 1)
            a += 2;
        if (nod(a, n) > 1)
            return false;
        cpp_int d = (n - 1) / 2;
        cpp_int new_a = a;
        for (int j = 0; j < d - 1; j++)
        {
            new_a *= a;
            new_a = new_a % n;
        }
        cpp_int dop_a = new_a;
        cpp_int dop_jac = jacobi_symbol(a, n);
        if (dop_a != dop_jac)
            return false;
    }
    return true;
}

pair <cpp_int, cpp_int> ElGamalSignatureSigning(cpp_int x_big, cpp_int y_big,
vector<cpp_int> publicKey, cpp_int privateKey) { // publicKey - (y, g, p)
    cpp_int p = publicKey[2];
    string strX = boost::lexical_cast<string>(x_big);
    string strY = boost::lexical_cast<string>(y_big);
    string strXY = strX + strY;
    cpp_int hashedStrXY = HashFunc(strXY, p);
    cpp_int k;
    bool flagS0 = true;
    cpp_int r, s;
    while (flagS0) {
        while (true) {
            //k = rand_large(2, p - 2);
            if (p < 10)
                k = rand() % (p - 1) + 2;
            else
                k = generate_random(2, p - 2);

            if (nod(k, p - 1) == 1)
                break;
            //cout << "\nNOD(" << k << ", " << p - 1 << ") = " << nod(k, p - 1) << "\n";
//УБРАТЬ
        }
        r = powMod(publicKey[1], k, p);
        cpp_int x1, y1;
        algEuclidExtended(k, p - 1, x1, y1);
        if (x1 < 0)
            x1 = x1 + p - 1;
        cpp_int k_obr = x1;
        s = ((hashedStrXY - privateKey * r) * k_obr) % (p - 1);
        if (s < 0)
            s = s + p - 1;
        if (s != 0)
            flagS0 = false;
    }
    pair <cpp_int, cpp_int> resPair = make_pair(r, s);
    return(resPair);
}

```

```

}

bool ElGamalSignatureVerifying(cpp_int x_big, cpp_int y_big, pair <cpp_int, cpp_int> rs,
vector<cpp_int> publicKey) { // publicKey - (y, g, p)
    cpp_int p = publicKey[2];
    cpp_int r = rs.first;
    cpp_int s = rs.second;
    if (r <= 0 || r >= p || s <= 0 || s >= p - 1)
        return false;
    string strX = boost::lexical_cast<string>(x_big);
    string strY = boost::lexical_cast<string>(y_big);
    string strXY = strX + strY;
    cpp_int hashedStrXY = HashFunc(strXY, p);
    cpp_int yRrS = ((powMod(publicKey[0], r, p)) * (powMod(r, s, p))) % p;
    cpp_int gM = powMod(publicKey[1], hashedStrXY, p);
    if (yRrS != gM)
        return false;
    return true;
}

vector<cpp_int> generateKeys(cpp_int p, cpp_int g, cpp_int &privateKey) {
    privateKey = rand_large(2, p - 2);
    cpp_int y = powMod(g, privateKey, p);
    vector<cpp_int> publicKey = { y, g, p };
    return(publicKey);
}

string findInStr(string const& str, int n) {
    if (str.length() < n)
        return str;
    return str.substr(0, n);
}

void helpFunc() {
    cout << "Введена команда с /h. Допустимые параметры:";
    cout << "\n\n/p:<p,g> - Параметры создания сообщений: модуль p, по которому создается
циклическая группа, и её генератор g (Если g = 0, то он сгенерируется сам).";
    cout << "\n\n/b:<p,g> - Параметры генерации ключа Боба (для подписи): p - простое число,
g - первообразный корень p (Если g = 0, то он сгенерируется сам).";
    cout << "\n\n/a:<p,g> - Параметры генерации ключа Алисы (для подписи): p - простое
число, g - первообразный корень p (Если g = 0, то он сгенерируется сам).";
    cout << "\n\n/pl:<length> - Параметр создания сообщений: длина модуля p, по которому
создается циклическая группа.";
    cout << "\n\n/bl:<length> - Параметр генерации ключа Боба (для подписи): длина модуля p,
по которому создается циклическая группа.";
    cout << "\n\n/al:<length> - Параметр генерации ключа Алисы (для подписи): длина модуля
p, по которому создается циклическая группа.";
    cout << "\n\n/no - Запустить программу без проверки p на простоту.";
    cout << "\n\n/h - информация о допустимых параметрах командной строки программы.\n";
}

void splitPG(string str, char symbol, cpp_int &a, cpp_int &b) {
    cpp_int elem;
    bool firstEl = true;
    bool secondEl = true;
    stringstream ss(str);
    while (ss >> elem) {
        if (firstEl) {
            a = elem;
            firstEl = false;
        }
        else if (secondEl) {
            b = elem;
        }
    }
}

```

```

        secondEl = false;
    }
    else
        return;
    if (ss.peek() == symbol) {
        ss.ignore();
    }
}
}

void printVecKeys(vector <cpp_int> a) {
    cout << "(";
    for (int i = 0; i < a.size() - 1; i++){
        cout << a[i] << ", ";
    }
    cout << a[a.size() - 1] << ")";
    return;
}

cpp_int findGen(cpp_int p) {
    vector<cpp_int> fact;
    cpp_int phi = p - 1, n = phi;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back(n);

    for (int res = 2; res <= p; ++res) {
        bool ok = true;
        for (size_t i = 0; i < fact.size() && ok; ++i)
            ok &= powMod(res, phi / fact[i], p) != 1;
        if (ok) return res;
    }
    return -1;
}

bool checkG(cpp_int p, cpp_int &g) {
    if (g == 0)
        g = findGen(p);
    unordered_set<cpp_int> powers;
    for (int i = 0; i < p - 1; ++i) {
        cpp_int powered_g = powMod(g, i, p);
        powers.insert(powered_g);
    }
    if (powers.size() == p - 1)
        return true;
    else
        return false;
}

void setPG(cpp_int& p, cpp_int& g, int lenP) {
    bool numIsPrime = false;
    while (!numIsPrime) {
        p = rand_large_by_bit_length(lenP);
        if (p == 0)
            continue;
        //cout << "\np =" << p << "\n";
        if (isPrimeSolovey_shtrassen(p, 15))
            numIsPrime = true;
    }
}

```

```

    g = findGen(p);
    return;
}

bool isCorrectPG(cpp_int &p, cpp_int &g, int lenP, bool dontCheckPrime) {
    if (lenP != 0)
        setPG(p, g, lenP);
    if (!dontCheckPrime)
        if (!isPrimeSolovey_shtrassen(p, 15)) {
            cout << "\n\nERROR! (один из p не является простым числом)\n";
            return false;
        }
    if (!checkG(p, g)) {
        cout << "\n\nERROR! (один из g не является примитивным элементом)\n";
        return false;
    }
    return true;
}

bool stsFunc(int argc, char* argv[]) {
    cpp_int p, g; // p = 7; g = 5
    cpp_int pAlice, gAlice, pBob, gBob; // alice: 11,2 ; bob: 23,5
    bool defaultAlice = true, defaultBob = true, defaultPG = true;
    int lenP = 0, lenBobP = 0, lenAliceP = 0;
    bool dontCheckPrime = false;
    for (int i = 0; argv[i]; i++) {
        string checkStr = string(argv[i]);
        if (findInStr(checkStr, 2) == "/h") {
            helpFunc();
            return 0;
        }
        if (checkStr.length() > 2) {
            string ifStr = findInStr(checkStr, 3);
            string subStr = checkStr.substr(3, checkStr.length());
            char symbol = ',';
            if (ifStr == "/p:") {
                splitPG(subStr, symbol, p, g);
            }
            if (ifStr == "/b:") {
                splitPG(subStr, symbol, pBob, gBob);
            }
            if (ifStr == "/a:") {
                splitPG(subStr, symbol, pAlice, gAlice);
            }
            if (ifStr == "/pl") {
                lenP = stoi(checkStr.substr(4, checkStr.length()));
            }
            if (ifStr == "/bl") {
                lenBobP = stoi(checkStr.substr(4, checkStr.length()));
            }
            if (ifStr == "/al") {
                lenAliceP = stoi(checkStr.substr(4, checkStr.length()));
            }
            if (ifStr == "/no") {
                dontCheckPrime = true;
            }
        }
    }

    if (!isCorrectPG(p, g, lenP, dontCheckPrime))
        return 0;
    if (!isCorrectPG(pBob, gBob, lenBobP, dontCheckPrime))
        return 0;
    if (!isCorrectPG(pAlice, gAlice, lenAliceP, dontCheckPrime))

```

```

        return 0;

cout << "\nДемонстрация работы протокола \"станция-станция\"\n\n";

cout << "Параметры создания сообщений:\nr = " << p << "\ng = " << g << "\n";
if (dontCheckPrime)
    cout << "Обратите внимание, введенные p не проверялись на простоту!\n";
//Открытые ключи для Алисы и Боба (нужны для цифровой подписи)
cpp_int privateAliceKey, privateBobKey;
vector<cpp_int> publicAliceKey = generateKeys(pAlice, gAlice, privateAliceKey);
vector<cpp_int> publicBobKey = generateKeys(pBob, gBob, privateBobKey);

cout << "\nСгенерированные открытый и закрытый ключи для схемы Эль-Гамала:\n";
cout << "Боб:\nОткрытый ключ (y, g, p) = ";
printVecKeys(publicBobKey);
cout << "\nЗакрытый ключ x = " << privateBobKey;
cout << "\nАлиса:\nОткрытый ключ (y, g, p) = ";
printVecKeys(publicAliceKey);
cout << "\nЗакрытый ключ x = " << privateAliceKey;

cout << "\n\nАЛИСА:\n";
//Alice

cpp_int x = rand_large(2, p - 1);
cout << "\n > Сгенерировала секретное большое число x = " << x;

cpp_int x_big = powMod(g, x, p); // X АЛИСЫ
cout << "\n > Вычислила  $X = g^x \bmod p =$  " << x_big;
cout << "\n > Отправила Бобу X";

cout << "\n\nБОБ:\n";
//Bob

cpp_int y = rand_large(2, p - 1);
cout << "\n > Сгенерировал секретное большое число y = " << y;
cpp_int y_big = powMod(g, y, p); // Y БОБА
cout << "\n > Вычислил  $Y = g^y \bmod p =$  " << y_big;
cpp_int k_big_bob = powMod(x_big, y, p);
cout << "\n > Вычислил сеансовый ключ  $K = X^y \bmod p =$  " << k_big_bob;
pair<cpp_int, cpp_int> SbXY = ElGamalSignatureSigning(x_big, y_big, publicBobKey,
privateBobKey); //подписываем Sb(X,Y)
cout << "\n > Вычислил подпись сообщения \"X,Y\" = (r, s) = (" << SbXY.first << ", " <<
SbXY.second << ")";
string AESKeyBob = generateAESKeyFromK(k_big_bob);
cout << "\n > Получил свой ключ для AES-шифрования из сеансового ключа: AESkey = " <<
AESKeyBob;
//шифруем Ek(Sb(X,Y)) ключом K

string strX = boost::lexical_cast<string>(SbXY.first);
string strY = boost::lexical_cast<string>(SbXY.second);
string strSbXY = strX + "," + strY;
//cout << "strSbXY: " << strSbXY << std::endl;
string EkSbXY = encryptAES(strSbXY, AESKeyBob);
//cout << "EkSbXY: " << EkSbXY << std::endl;
cout << "\n > Зашифровал подпись сообщения " << strSbXY << " и получил " << EkSbXY;
cout << "\n > Отправил Алисе Y и зашифрованную подпись";
//БОБ ОТПРАВИЛА АЛИСЕ Y и ЗАШИФРОВАННОЕ XY

//Alice
cout << "\n\nАЛИСА:\n";

cpp_int k_big_alice = powMod(y_big, x, p);
cout << "\n > Вычислила сеансовый ключ  $K = Y^x \bmod p =$  " << k_big_alice;

```

```

if (k_big_bob != k_big_alice) {
    cout << "\n\nERROR! (Вычисленные сеансовые ключи не совпадают)";
    return 0;
}
//расшифровываем Dk(Ek(Sb(X,Y))) ключом K

string AESKeyAlice = generateAESKeyFromK(k_big_alice);
cout << "\n > Получила свой ключ для AES-шифрования из сеансового ключа: AESkey = " <<
AESKeyAlice;
string DkEkSbXY = decryptAES(EkSbXY, AESKeyAlice);
cout << "\n > Расшифровала сообщение " << EkSbXY << " и получила " << DkEkSbXY;

istringstream iss(DkEkSbXY);
string strXnew, strYnew;
getline(iss, strXnew, ','); // Разделяем строку по запятой и получаем первое число
getline(iss, strYnew, ','); // Получаем второе число
cpp_int xNew = stoi(strXnew);
cpp_int yNew = stoi(strYnew);
//cout << "\nxNew = " << xNew << "; yNew = " << yNew << "\n";
pair <cpp_int, cpp_int> SbXYnew = make_pair(xNew, yNew);
//проверяем подпись Sb(X,Y). Если верно, то дальше
if (ElGamalSignatureVerifying(x_big, y_big, SbXYnew, publicBobKey))
    cout << "\n > Проверила подпись Боба. Она верна. Боб подтвердил свою личность";
else {
    cout << "\n > Проверила подпись Боба. Она не верна. Боб не подтвердил свою
личность";
    return 0;
}

//подписываем Sa(X,Y)
pair <cpp_int, cpp_int> SaXY = ElGamalSignatureSigning(x_big, y_big, publicAliceKey,
privateAliceKey); //подписываем Sb(X,Y)
cout << "\n > Вычислила подпись сообщения \"X,Y\" = (r, s) = (" << SaXY.first << ", " <<
SaXY.second << ")";
//шифруем Ek(Sa(X,Y)) ключом K

string strXa = boost::lexical_cast<string>(SaXY.first);
string strYa = boost::lexical_cast<string>(SaXY.second);
string strSaXY = strX + "," + strY;
//cout << "strSaXY: " << strSaXY << std::endl;
string EkSaXY = encryptAES(strSaXY, AESKeyAlice);
//cout << "EkSaXY: " << EkSaXY << std::endl;
cout << "\n > Зашифровала подпись сообщения " << strSaXY << " и получила " << EkSaXY;
cout << "\n > Отправила Бобу зашифрованную подпись";

cout << "\n\nБОБ:\n";
//Bob
//расшифровываем Dk(Ek(Sa(X,Y))) ключом K
string DkEkSaXY = decryptAES(EkSaXY, AESKeyBob);
cout << "\n > Расшифровал сообщение " << EkSaXY << " и получил " << DkEkSaXY;

istringstream iss2(DkEkSaXY);
string strXnew2, strYnew2;
getline(iss2, strXnew2, ','); // Разделяем строку по запятой и получаем первое число
getline(iss2, strYnew2, ','); // Получаем второе число
cpp_int xNew2 = std::stoi(strXnew2);
cpp_int yNew2 = std::stoi(strYnew2);
pair <cpp_int, cpp_int> SbXYnew2 = make_pair(xNew2, yNew2);

//проверяем подпись Sa(X,Y). Если верно, то дальше
if (ElGamalSignatureVerifying(x_big, y_big, SaXY, publicAliceKey))
    cout << "\n > Проверил подпись Алисы. Она верна. Алиса подтвердила свою
личность\n\n\n";
else {

```



```

        cout << "\n > Проверил подпись Алисы. Она не верна. Алиса не подтвердила свою
личность\n\n\n";
        return 0;
    }
    return 0;
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "rus");

    stsFunc(argc, argv);
}

```