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
Яхина Шамиля Илдусовича

Преподаватель
аспирант _____ Р. А. Фарахутдинов
подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы - реализация протокола передачи секретного ключа по открытому ключу Encrypted Key Exchange (EKE) на базе алгоритма Диффи-Хеллмана.

1 Теоретические сведения

Данный протокол использует симметричное и асимметричное шифрование. В каждой сессии протокола генерируется случайным образом открытый ключ и шифруется на имеющимся у сторон секретном ключе. Использование двух видов шифрования - главная особенность данного протокола. В качестве общего секрета в схеме может выступать пароль, что делает данный протокол удобным при использовании в системах, где общий секрет является паролем.

При реализации EKE с использованием протокола Диффи — Хеллмана ключ K генерируется автоматически участниками работы протокола во время его исполнения.

При использовании протокола DH-EKE решается уязвимость базового протокола Диффи-Хеллмана к атаке человек посередине (MITM). Если криптоаналитик не знает общий секрет пользователей, то он не сможет подобрать ключ, потому что шифруются случайные значения.

1.1 Описание протокола EKE на базе алгоритма Диффи-Хеллмана

Алгоритм протокола EKE на базе алгоритма Диффи-Хеллмана.

Вход: Простое число n, по модулю которого производятся вычисления, и g - порождающий элемент группы.

Выход: Сеансовый ключ K.

1. Алиса случайно выбирает r_A и посылает Бобу $\{A, g^{r_A} \mod n\}$. При этом нет необходимости, чтобы Алиса шифровала первое сообщение на общем секрете P;
2. Боб случайно выбирает r_B, вычисляет K, генерирует случайную строку R_B, шифрует её на ключе K. Далее - вычисляет $g^{r_B} \mod n$, шифрует его на общем секрете P. Передает Алисе $K = g^{r_A * r_B} \mod n$ и $\{E_P(g^{r_B} \mod n), E_K(R_B)\}$
3. Трент передает Алисе $\{E_{AT}(B, R_A, K_{AB}, T_B), E_{BT}(A, K_{AB}, T_B), R_B\}$;
4. Алиса получает $\{g^{r_B} \mod n\}$, расшифровывая первую часть сообщения с использованием общего секрета P. Далее - вычисляет K, расшифровывает полученным ключом вторую часть сообщения Боба. Генерирует случайную строку R_A, шифрует полученные строки на ключе K и посылает Бобу $\{E_K(R_A, R_B)\}$;
5. Боб получает R_A и R_B, расшифровывая сообщение. Значение R_B, полу-

ченное от Алисы, сравнивается с тем, что было выбрано в пункте 3. Если значения совпадают, то Боб шифрует R_A на ключе K и посылает Алисе $\{E_K(R_A)\}$;

6. Алиса получает R_A, расшифровывая сообщение. Значение R_A, полученное от Боба, сравнивается с тем, что было выбрано в пункте 4. Если значения совпадают, то работа протокола завершена - участники могут обмениваться сообщениями используя при их шифровании ключ K.

2 Практическая реализация

2.1 Описание программы

Все шаги алгоритма происходят в функции EKE_DH.

Для генерации секретного ключа P (общего секрета) используется функция $generateRandomHexKey$, т.е. общий секрет создается в таком формате, что его можно будет использовать как ключ для AES-шифрования.

Функция $generateAESKeyFromK$ преобразует сеансовый ключ K в ключ для AES-шифрования.

Для генерации случайной строки используется функция $generate\_random\_string$.

AES-шифрование реализовано в функции $encryptAES$, а дешифрование в функции $decryptAES$.

Функция $generateRandomPrime$ генерирует простое случайное число в заданном диапазоне. Для проверки числа на простоту с помощью теста Миллера-Рабина используется функция $isPrimeMillerRabin$.

Описание дополнительных функций, используемых в программе: $powMod$ - функция возведения в степень по модулю, $isCorrectNG$ - проверка чисел, поступающих на вход алгоритма, на корректность, $rand\_large\_by\_bit\_length$ - генерация случайного числа в выбранном промежутке, $byteArrayToHexString$ - функция преобразования массива байтов в строку в шестнадцатеричном формате.

2.2 Тестирование программы

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

```
C:\Users\Shamil_\source\repos\lab3_EKE_DiffieHellman\x64\Release>lab3_EKE_DiffieHellman.exe /h
Введена команда с /h. Допустимые параметры:

/i:<n,g> - Параметры: модуль n, по которому производятся вычисления, и g - порождающий элемент группы (Если g = 0, то он сгенерируется сам).

/pl:<length> - Длина модуля n, по которому производятся вычисления.

/bl:<length> - Битовая длина случайного числа и случайной строки Боба (r_b и R_b).

/al:<length> - Битовая длина случайного числа и случайной строки Алисы (r_a и R_a).

/h - информация о допустимых параметрах командной строки программы.

C:\Users\Shamil_\source\repos\lab3_EKE_DiffieHellman\x64\Release>
```

Рисунок 1 – Вызов параметра $help$

В примере, показанном на рисунках 2 и 3, задаются параметры $n_l = 18, b_l = 32, a_l = 32$.

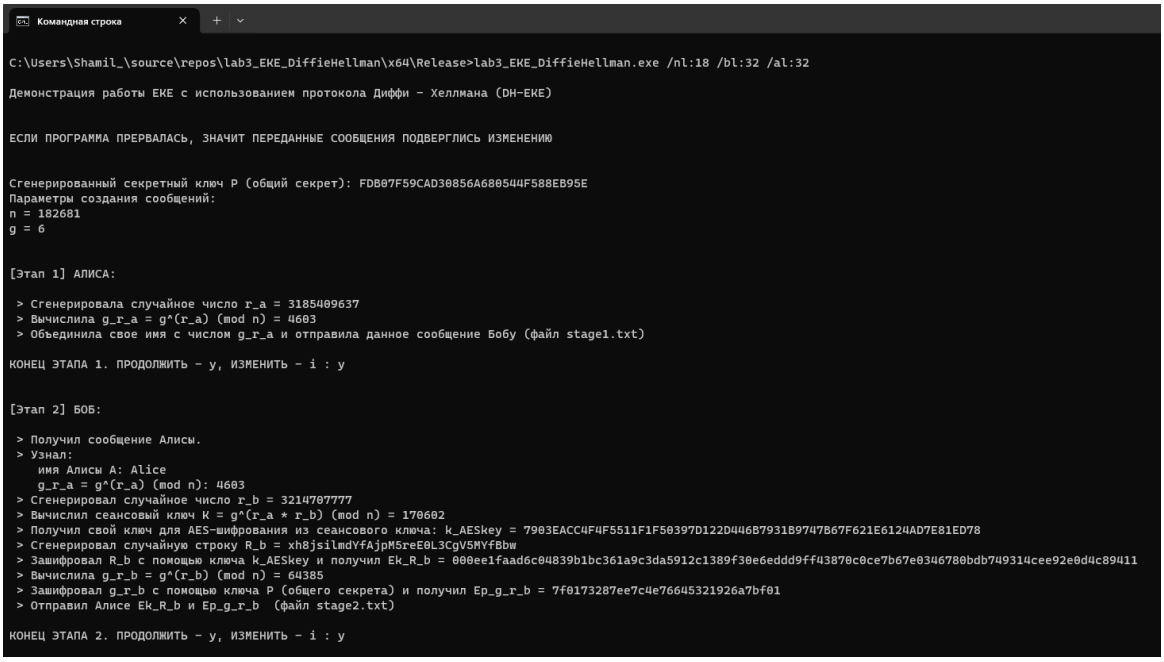

Рисунок 2 – Пример корректной работы протокола EKE на базе алгоритма Диффи-Хеллмана

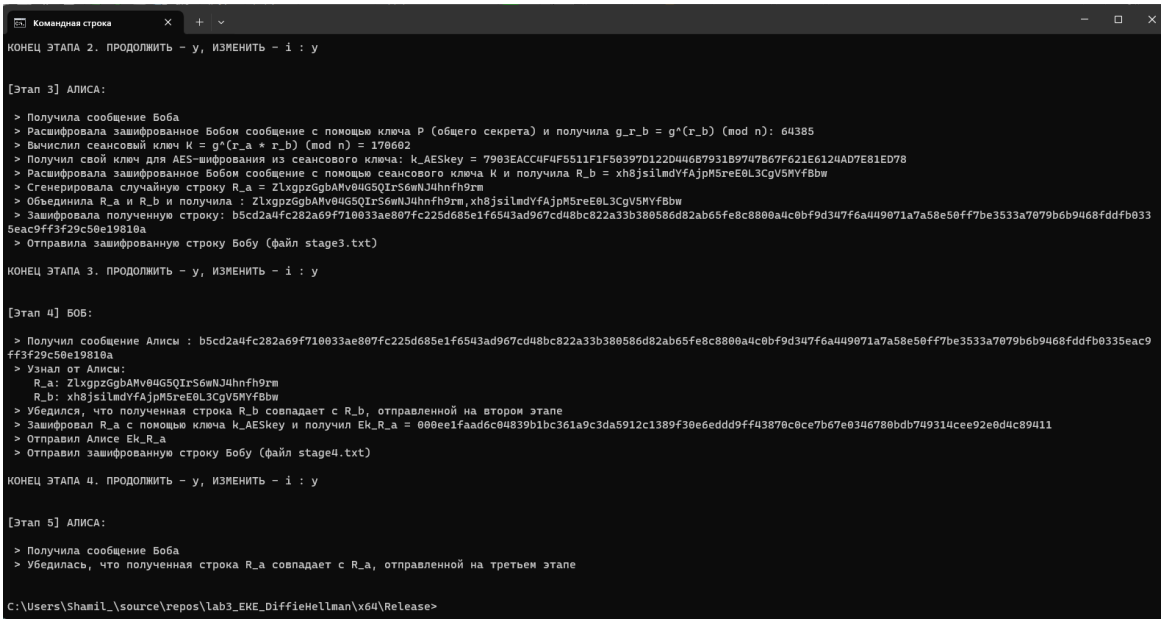

Рисунок 3 – Пример корректной работы протокола EKE на базе алгоритма Диффи-Хеллмана

В примере, показанном на рисунках 4 и 5, изменим после первого шага $g^{r_A} \mod n$ на другое число.

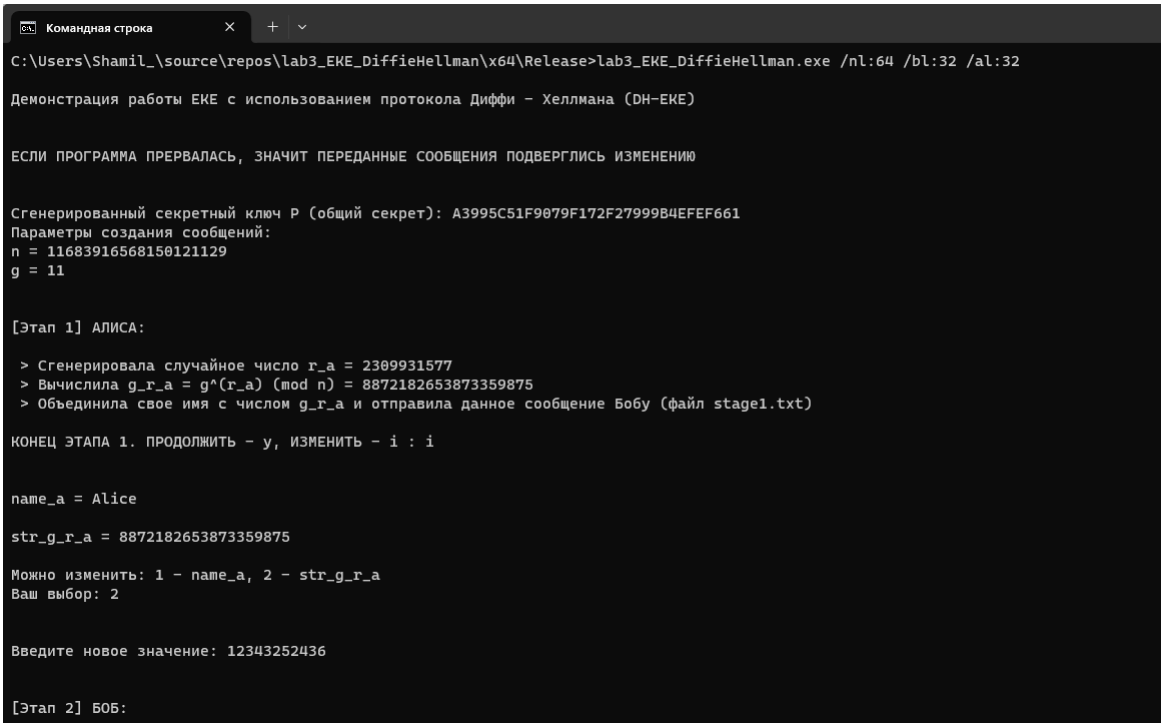

Рисунок 4 – Пример работы протокола EKE на базе алгоритма Диффи-Хеллмана с изменением параметра

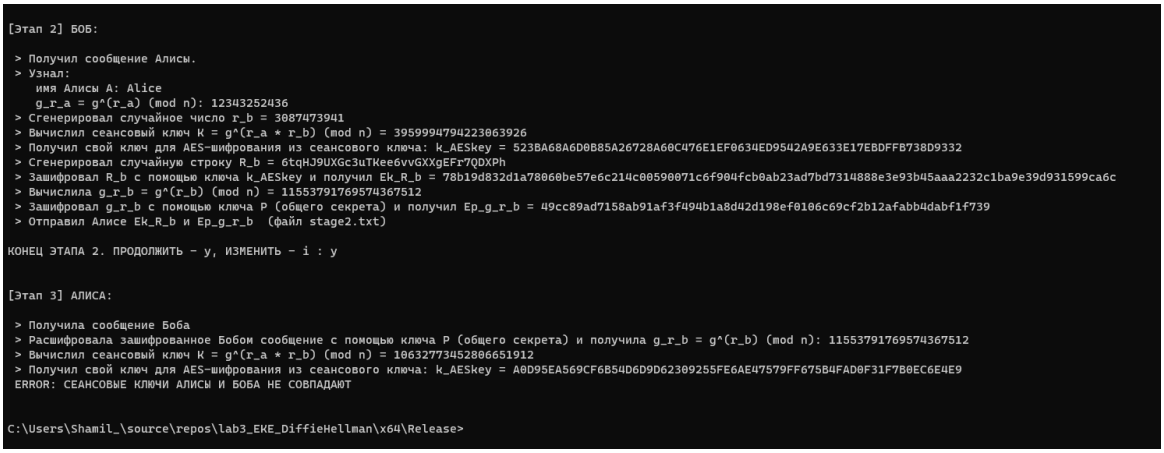

Рисунок 5 – Пример работы протокола EKE на базе алгоритма Диффи-Хеллмана с изменением параметра

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;
cpp_int pSize;

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 l) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> distribution(0, 1);
    cpp_int result = 0;
    for (int i = 1; i < l - 1; ++i) {
        result <<= 1;
        result += distribution(gen);
    }
    result |= (cpp_int(1) << (l - 1));
    result |= 1;
    return result;
}
string generate_random_string(int length) {
    const std::string valid_chars =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    random_device gen;
    uniform_int_distribution<int> ui(0, valid_chars.size() - 1);
    string result;

    for (int i = 0; i < length; ++i) {
        int random_index = ui(gen);
        result.push_back(valid_chars[random_index]);
    }

    return result;
}
```

```
cpp_int powMod(cpp_int a, cpp_int n, cpp_int m) {
    if (n == 0)
        return 1;
    if (n % 2 == 1)
        return (powMod(a, n - 1, m) * a) % m;
    else {
        cpp_int b = powMod(a, n / 2, m);
        return (b * b) % m;
    }
}

bool isPrimeMillerRabin(cpp_int n, cpp_int k) {
    if (n <= 1 || n == 4) return false;
    if (n <= 3) return true;

    cpp_int d = n - 1;
    while (d % 2 == 0)
        d /= 2;

    for (int i = 0; i < k; i++) {
        cpp_int a = 2 + rand() % (n - 3);
        cpp_int x = powMod(a, d, n);

        if (x == 1 || x == n - 1)
            continue;

        while (d != n - 1) {
            x = (x * x) % n;
            d *= 2;

            if (x == 1) return false;
            if (x == n - 1) break;
        }

        if (x != n - 1) return false;
    }

    return true;
}

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

void splitNG(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();
        }
    }
}


cpp_int findGen(cpp_int p) {
    cpp_int countOfOp = pSize * 1000;
    if (pSize > 95)
        countOfOp *= 3;
    vector<cpp_int> fact;
    cpp_int phi = p - 1, n = phi;
    for (cpp_int i = 2; i * i <= n; ++i) {
        cpp_int nmodi = n % i;
        if (n % i == 0) {
            fact.push_back(i);
            while (n % i == 0)
                n /= i;
        }
        if (i > countOfOp)
            return -1;
    }
    if (n > 1)
        fact.push_back(n);
    for (cpp_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;
}

cpp_int min_num_by_l_bits(int l) {
    cpp_int result = 1;
    result |= (cpp_int(1) << (l - 1));
    return result;
}

cpp_int max_num_by_l_bits(int l) {
    cpp_int result = (cpp_int(1) << l) - 1;
    return result;
}

cpp_int generateRandomPrime(int l) {
    cpp_int minNum = min_num_by_l_bits(l);
    cpp_int maxNum = max_num_by_l_bits(l);
    cpp_int randNum = rand_large_by_bit_length(l);
    cpp_int newNum = randNum;
    bool plus1 = true;
    if (newNum % 2 == 0) {
        if (newNum + 1 < maxNum)
            newNum += 1;
        else {
            newNum -= 1;
            plus1 = false;
        }
    }
    randNum = newNum;
    while (!isPrimeMillerRabin(newNum, 20)) {
        if (plus1)
            newNum += 2;
        else
```

```
            newNum -= 2;
        if (newNum > maxNum) {
            newNum = randNum;
            plus1 = false;
        }
        if (newNum < minNum) {
            newNum = randNum;
            plus1 = true;
        }
    }
    return newNum;
}

void setNG(cpp_int& p, cpp_int& g, int lenP) {
    p = generateRandomPrime(lenP);
    while (!isPrimeMillerRabin(p, 20)) {
        p = generateRandomPrime(lenP);
    }
    cpp_int x0 = 1;
    g = findGen(p);
    while (g == -1) {
        p = generateRandomPrime(lenP);
        while (!isPrimeMillerRabin(p, 20)) {
            p = generateRandomPrime(lenP);
        }
        g = findGen(p);
    }

    return;
}

bool isCorrectNG(cpp_int& p, cpp_int& g, int lenP) {
    if (lenP != 0)
        setNG(p, g, lenP);
    return true;
}

string generateRandomHexKey(int keySize) {
    AutoSeededRandomPool prng;
    SecByteBlock key(keySize);
    prng.GenerateBlock(key, keySize);
    string hexKey;
    HexEncoder encoder(new StringSink(hexKey));
    encoder.Put(key, keySize);
    encoder.MessageEnd();
    return hexKey;
}

vector<string> splitString(const string& input, char zn) {
    istringstream stream(input);
    string str1;
    vector<string> strs;
    while (getline(stream, str1, zn)) {
        strs.push_back(str1);
    }
    return strs;
}

string byteArrayToHexString(const byte* input, size_t length) {
    ostringstream ss;
    ss << hex << setfill('0');
    for (size_t i = 0; i < length; ++i)
        ss << setw(2) << static_cast<int>(input[i]);
    return ss.str();
```

```
}

void helpFunc() {
    cout << "Введена команда с /h. Допустимые параметры:";
    cout << "\n\n/i:<n,g> - Параметры: модуль n, по которому производятся вычисления, и g -
порождающий элемент группы (Если g = 0, то он сгенерируется сам).";
    cout << "\n\n/pl:<length> - Длина модуля n, по которому производятся вычисления.";
    cout << "\n\n/bl:<length> - Битовая длина случайного числа и случайной строки Боба (r_b
и R_b).";
    cout << "\n\n/al:<length> - Битовая длина случайного числа и случайной строки Алисы (r_a
и R_a).";
    cout << "\n\n/h – информация о допустимых параметрах командной строки программы.\n";
}

bool EKE_DH(cpp_int n, cpp_int g, int lenRa, int lenRb) {
    cout << "\nДемонстрация работы EKE с использованием протокола Диффи — Хеллмана (DH-
EKE)\n\n";

    cout << "\nЕСЛИ ПРОГРАММА ПРЕРВАЛАСЬ, ЗНАЧИТ ПЕРЕДАННЫЕ СООБЩЕНИЯ ПОДВЕРГЛИСЬ
ИЗМЕНЕНИЮ\n\n";

    const string secretPkey = generateRandomHexKey(AES_KEY_SIZE);
    cout << "\nСгенерированный секретный ключ P (общий секрет): " << secretPkey;
    char zn = ',';

    cout << "\nПараметры создания сообщений:\nn = " << n << "\ng = " << g << "\n";

    //АЛИСА

    cout << "\n\n[Этап 1] АЛИСА:\n";

    cpp_int r_a = rand_large_by_bit_length(lenRa);
    string name_a = "Alice";
    string str_r_a = boost::lexical_cast<string>(r_a);
    cout << "\n > Сгенерировала случайное число r_a = " << str_r_a;
    cpp_int g_r_a = powMod(g, r_a, n);
    string str_g_r_a = boost::lexical_cast<string>(g_r_a);
    cout << "\n > Вычислила g_r_a = g^(r_a) (mod n) = " << str_g_r_a;
    cout << "\n > Объединила свое имя с числом g_r_a и отправила данное сообщение Бобу (файл
stage1.txt)";
    vector <string> stage1_A = { name_a , str_g_r_a };

    string userCout;
    cout << "\n\nКОНЕЦ ЭТАПА 1. ПРОДОЛЖИТЬ - y, ИЗМЕНИТЬ - i : ";
    cin >> userCout;
    if (userCout == "i") {
        cout << "\n\nname_a = " << name_a;
        cout << "\n\nstr_g_r_a = " << str_g_r_a;
        cout << "\n\nМожно изменить: 1 - name_a, 2 - str_g_r_a";
        cout << "\nВаш выбор: ";
        string userChoiceIzm;
        cin >> userChoiceIzm;
        if (userChoiceIzm == "1") {
            cout << "\n\nВведите новое значение: ";
            cin >> stage1_A[0];
        }
        else if (userChoiceIzm == "2") {
            cout << "\n\nВведите новое значение: ";
            cin >> stage1_A[1];
        }
    }
    else if (userCout != "y")
        return false;
```

```
    //БОБ

    cout << "\n\n[Этап 2] БОБ:\n";

    vector <string> stage1_B = stage1_A;
    cout << "\n > Получил сообщение Алисы.";

    string BZ_name_a = stage1_B[0];
    string BZ_g_r_a_str = stage1_B[1];
    cpp_int BZ_g_r_a = cpp_int(BZ_g_r_a_str);
    cout << "\n > Узнал:\n    имя Алисы A: " << BZ_name_a << "\n    g_r_a = g^(r_a) (mod n):
" << BZ_g_r_a;

    cpp_int r_b = rand_large_by_bit_length(lenRb);
    string str_r_b = boost::lexical_cast<string>(r_b);
    cout << "\n > Сгенерировал случайное число r_b = " << str_r_b;

    cpp_int BZ_k = powMod(BZ_g_r_a, r_b, n);
    cout << "\n > Вычислил сеансовый ключ K = g^(r_a * r_b) (mod n) = " << BZ_k;

    string BZ_AESKeyK = generateAESKeyFromK(BZ_k);
    cout << "\n > Получил свой ключ для AES-шифрования из сеансового ключа: k_AESkey = " <<
BZ_AESKeyK;

    string BZ_R_b = generate_random_string(lenRb);
    cout << "\n > Сгенерировал случайную строку R_b = " << BZ_R_b;

    string Ek_R_b = encryptAES(BZ_R_b, BZ_AESKeyK);
    cout << "\n > Зашифровал R_b с помощью ключа k_AESkey и получил Ek_R_b = " <<
byteArrayToHexString(reinterpret_cast<const byte*>(Ek_R_b.data()), Ek_R_b.length());

    cpp_int g_r_b = powMod(g, r_b, n);
    string str_g_r_b = boost::lexical_cast<string>(g_r_b);
    cout << "\n > Вычислила g_r_b = g^(r_b) (mod n) = " << str_g_r_b;

    string Ep_g_r_b = encryptAES(str_g_r_b, secretPkey);
    cout << "\n > Зашифровал g_r_b с помощью ключа P (общего секрета) и получил Ep_g_r_b = "
<< byteArrayToHexString(reinterpret_cast<const byte*>(Ep_g_r_b.data()), Ep_g_r_b.length());

    vector <string> stage2_B = { Ek_R_b , Ep_g_r_b };
    cout << "\n > Отправил Алисе Ek_R_b и Ep_g_r_b  (файл stage2.txt)";

    cout << "\n\nКОНЕЦ ЭТАПА 2. ПРОДОЛЖИТЬ - y, ИЗМЕНИТЬ - i : ";
    cin >> userCout;
    if (userCout == "i") {
        cout << "\n\nEk_R_b = " << byteArrayToHexString(reinterpret_cast<const
byte*>(Ek_R_b.data()), Ek_R_b.length());
        cout << "\n\nEp_g_r_b = " << byteArrayToHexString(reinterpret_cast<const
byte*>(Ep_g_r_b.data()), Ep_g_r_b.length());
        cout << "\n\nМожно изменить: 1 - Ek_R_b, 2 - Ep_g_r_b";
        cout << "\nВаш выбор: ";
        string userChoiceIzm;
        cin >> userChoiceIzm;
        if (userChoiceIzm == "1") {
            cout << "\n\nВведите новое значение: ";
            cin >> stage2_B[0];
        }
        else if (userChoiceIzm == "2") {
            cout << "\n\nВведите новое значение: ";
            cin >> stage2_B[1];
        }
    }
    else if (userCout != "y")
        return false;
```

```
    //АЛИСА

    cout << "\n\n[Этап 3] АЛИСА:\n";

    vector <string> stage2_A = stage2_B;
    string AZ_Ek_R_b = stage2_A[0];
    string AZ_Ep_g_r_b = stage2_A[1];
    cout << "\n > Получила сообщение Боба";
    string AZ_g_r_b_str = decryptAES(AZ_Ep_g_r_b, secretPkey);
    cpp_int AZ_g_r_b = cpp_int(AZ_g_r_b_str);
    cout << "\n > Расшифровала зашифрованное Бобом сообщение с помощью ключа P (общего
секрета) и получила g_r_b = g^(r_b) (mod n): " << AZ_g_r_b;
    cpp_int AZ_k = powMod(AZ_g_r_b, r_a, n);
    cout << "\n > Вычислил сеансовый ключ K = g^(r_a * r_b) (mod n) = " << AZ_k;
    string AZ_AESKeyK = generateAESKeyFromK(AZ_k);
    cout << "\n > Получил свой ключ для AES-шифрования из сеансового ключа: k_AESkey = " <<
AZ_AESKeyK;

    if (AZ_AESKeyK != BZ_AESKeyK) {
        cout << "\n ERROR: СЕАНСОВЫЕ КЛЮЧИ АЛИСЫ И БОБА НЕ СОВПАДАЮТ\n\n";
        return false;
    }
    string AZ_R_b = decryptAES(AZ_Ek_R_b, AZ_AESKeyK);
    cout << "\n > Расшифровала зашифрованное Бобом сообщение с помощью сеансового ключа K и
получила R_b = " << AZ_R_b;
    string AZ_R_a = generate_random_string(lenRa);
    cout << "\n > Сгенерировала случайную строку R_a = " << AZ_R_a;
    string alice_second_str = AZ_R_a + "," + AZ_R_b;
    cout << "\n > Объединила R_a и R_b и получила : " << alice_second_str;
    string Ek_R_a_R_b = encryptAES(alice_second_str, AZ_AESKeyK);
    cout << "\n > Зашифровала полученную строку: " <<
byteArrayToHexString(reinterpret_cast<const byte*>(Ek_R_a_R_b.data()), Ek_R_a_R_b.length());
    vector <string> stage3_A = { Ek_R_a_R_b };
    cout << "\n > Отправила зашифрованную строку Бобу (файл stage3.txt)";

    cout << "\n\nКОНЕЦ ЭТАПА 3. ПРОДОЛЖИТЬ - y, ИЗМЕНИТЬ - i : ";
    cin >> userCout;
    if (userCout == "i") {
        cout << "\n\nEk_R_a_R_b = " << byteArrayToHexString(reinterpret_cast<const
byte*>(Ek_R_a_R_b.data()), Ek_R_a_R_b.length());
        cout << "\n\nМожно изменить: 1 - Ek_R_a_R_b";
        cout << "\nВаш выбор: ";
        string userChoiceIzm;
        cin >> userChoiceIzm;
        if (userChoiceIzm == "1") {
            cout << "\n\nВведите новое значение: ";
            cin >> stage3_A[0];
        }
    }
    else if (userCout != "y")
        return false;

    //БОБ

    cout << "\n\n[Этап 4] БОБ:\n";

    vector <string> stage3_B = stage3_A;
    string BZ_Ek_R_a_R_b = Ek_R_a_R_b;
    cout << "\n > Получил сообщение Алисы : " << byteArrayToHexString(reinterpret_cast<const
byte*>(BZ_Ek_R_a_R_b.data()), BZ_Ek_R_a_R_b.length());
    string BZ_alice_second_str = decryptAES(BZ_Ek_R_a_R_b, BZ_AESKeyK);
    vector<string> alice_second_str_vec = splitString(BZ_alice_second_str, zn);
    string BZ_new_R_a = alice_second_str_vec[0];
```

```
    string BZ_new_R_b = alice_second_str_vec[1];
    cout << "\n > Узнал от Алисы:\n    R_a: " << BZ_new_R_a << "\n    R_b: " << BZ_new_R_b;

    if (BZ_R_b == BZ_new_R_b) {
        cout << "\n > Убедился, что полученная строка R_b совпадает с R_b, отправленной на
втором этапе";
    }
    else {
        cout << "\n ERROR: Полученная строка R_b не совпадает с R_b, отправленной на втором
этапе\n\n";
        return false;
    }

    string Ek_R_a = encryptAES(BZ_new_R_a, BZ_AESKeyK);
    cout << "\n > Зашифровал R_a с помощью ключа k_AESkey и получил Ek_R_a = " <<
byteArrayToHexString(reinterpret_cast<const byte*>(Ek_R_b.data()), Ek_R_b.length());
    cout << "\n > Отправил Алисе Ek_R_a";
    vector <string> stage4_B = { Ek_R_a };
    cout << "\n > Отправил зашифрованную строку Бобу (файл stage4.txt)";

    cout << "\n\nКОНЕЦ ЭТАПА 4. ПРОДОЛЖИТЬ - y, ИЗМЕНИТЬ - i : ";
    cin >> userCout;
    if (userCout == "i") {
        cout << "\n\nEk_R_a = " << byteArrayToHexString(reinterpret_cast<const
byte*>(Ek_R_b.data()), Ek_R_b.length());
        cout << "\n\nМожно изменить: 1 - Ek_R_a";
        cout << "\nВаш выбор: ";
        string userChoiceIzm;
        cin >> userChoiceIzm;
        if (userChoiceIzm == "1") {
            cout << "\n\nВведите новое значение: ";
            cin >> stage4_B[0];
        }
    }
    else if (userCout != "y")
        return false;

    //АЛИСА

    cout << "\n\n[Этап 5] АЛИСА:\n";

    vector <string> stage4_A = stage4_B;
    string AZ_Ek_R_a = stage4_A[0];
    cout << "\n > Получила сообщение Боба";
    string AZ_new_R_a_str = decryptAES(AZ_Ek_R_a, AZ_AESKeyK);

    if (AZ_new_R_a_str == AZ_R_a) {
        cout << "\n > Убедилась, что полученная строка R_a совпадает с R_a, отправленной на
третьем этапе\n\n";
    }
    else {
        cout << "\n ERROR: Полученная строка R_a не совпадает с R_a, отправленной на третьем
этапе\n\n";
        return false;
    }
    return true;
}

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

    cpp_int n, g; // n = 7; g = 5
    int lenP = 0, lenRb = 0, lenRa = 0, lenN = 0;
    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 == "/i:") {
                splitNG(subStr, symbol, n, g);
            }
            if (ifStr == "/nl") {
                lenN = stoi(checkStr.substr(4, checkStr.length()));
            }
            if (ifStr == "/bl") {
                lenRb = stoi(checkStr.substr(4, checkStr.length()));
            }
            if (ifStr == "/al") {
                lenRa = stoi(checkStr.substr(4, checkStr.length()));
            }
        }
    }

    pSize = lenN;
    if (!isCorrectNG(n, g, lenN))
        return 0;

    EKE_DH(n, g, lenRa, lenRb);

    return 0;
}
```