

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

**Скрытый канал связи
ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Яхина Шамяля Илдусовича

Преподаватель

аспирант

подпись, дата

Р. А. Фарахутдинов

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Теоретические сведения	4
1.1 Описание алгоритма скрытого канала на основе ESIGN	4
2 Практическая реализация	5
2.1 Описание программы	5
2.2 Тестирование программы.....	5

ВВЕДЕНИЕ

Цель работы - реализация скрытого канала связи на основе ESIGN.

1 Теоретические сведения

Основные параметры схемы в случае скрытого канала. Алиса подписывает безобидное сообщение m , Боб проверяет подпись. Вместе в этом передаётся секретное сообщение m^* . Генерируются: p, q, r — большие простые числа одинаковой длины; вычисляется $n = p^2qr$; выбирается $k \geq 4$ (параметр безопасности). Элементы n, k объявляются открытым ключом, элементы p, q, r — закрытым ключом Алисы. Часть этого ключа, а именно r , должна быть известна Бобу для извлечения секретного сообщения и необходимо выполнение неравенства $m^* < r$.

1.1 Описание алгоритма скрытого канала на основе ESIGN

Алгоритм скрытого канала на основе ESIGN.

Вход: Битовая длина чисел p, q, r , параметр безопасности k , секретное число Алисы m^* , сообщение m .

Выход: Найденное Бобом секретное сообщение m .

Генерация подписи.

1. Алиса вычисляет h , где $h = H(m)$ — хэш-функция со значением от 0 до $n-1$;
2. Алиса вычисляет x , где $x = m^* + ur$ и u — случайное число из интервала $0 < u < pqr$;
3. Алиса вычисляет a и b , где $a = \lceil \frac{h - (x^k \bmod n)}{pqr} \rceil$, $b = a(k * x^{k-1})^{-1} \bmod p$;
4. Алиса вычисляет s , где $s = x + bpqr$;
5. Алиса передает Уолтеру $\{m, s\}$.

Проверка подписи.

6. Уолтер вычисляет h , где $h = H(m)$;
7. Уолтер вычисляет c , где $c = \lceil \frac{3}{4} \log_2 n \rceil$, ($\frac{3}{4} \log_2 n$ - это $\frac{3}{4}$ от числа бит в n);
8. Уолтер проверяет неравенство, что $h \leq s^k \bmod n \leq h + 2^c$;
9. Уолтер передает Бобу $\{m, s\}$.

Проверка подписи.

10. Боб также проверяет подпись и извлекает секретное сообщение, $m^* = s \bmod r$;

2 Практическая реализация

2.1 Описание программы

Все действия алгоритма разделения секрета выполняются в функции *esign_main*.

Параметры p, q, r вычисляются в функции *generatePQR*, где используется функция для генерации больших простых чисел *generateRandomPrime*.

Функция *generate_random* генерирует случайное число в заданном промежутке.

Для возведения числа в степень по модулю реализована функция *powMod*.

Для вычисления обратного элемента используется расширенный алгоритм Евклида, который реализован в функции *algEuclidExtended*.

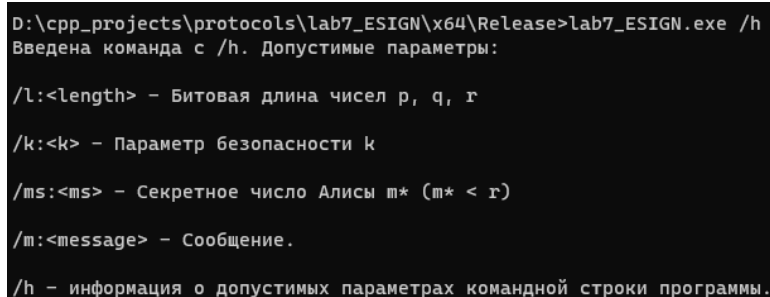
Подпись генерируется в функции *generateSign*.

Проверки подписи Уолтером и Бобом реализованы в функциях *wolterCheckSign* и *bobCheckSign*, соответственно. После проверки подписи Боб находит секретное сообщение, вычислив $s \bmod r$.

При вычислении и проверке подписи используется хэш-функция SHA256, которая реализована в функции *HashFunc*.

2.2 Тестирование программы

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



```
D:\cpp_projects\protocols\lab7_ESIGN\x64\Release>lab7_ESIGN.exe /h
Введена команда с /h. Допустимые параметры:

/l:<length> - Битовая длина чисел p, q, r
/k:<k> - Параметр безопасности k
/ms:<ms> - Секретное число Алисы m* (m* < r)
/m:<message> - Сообщение.
/h - информация о допустимых параметрах командной строки программы.
```

Рисунок 1 – Вызов параметра *help*

На рисунках 2 - 3 показан запуск программы с параметрами $l = 64$, $ms = 12329843443$, $m = message$, $k = 4$.

```
Командная строка
D:\cpp_projects\protocols\lab7_ESIGN\Release>lab7_ESIGN.exe /l:64 /ms:12329843443 /m:message /k:4

ГЕНЕРАЦИЯ ПАРАМЕТРОВ

p = 12391501583100519541
Битовая длина p: 64
q = 12500126525092535887
Битовая длина q: 64
r = 9382014351319198367
Битовая длина r: 64
n = p^2 * q * r = 18007705322005458156204092009040859365633889956690416448930533016323728586249
Битовая длина n: 254
Открытый ключ: {n, k} = {18007705322005458156204092009040859365633889956690416448930533016323728586249, 4}
Закртыый ключ Алисы: {p, q, r} = {12391501583100519541, 12500126525092535887, 9382014351319198367}

ГЕНЕРАЦИЯ ПОДПИСИ

h = 490423192
u = 41856534419008723380009581295038198193
x = 392698606615625827380265996489203319333638205821157794274
a = 7626130865300630152
b = 92565288911908876352602502363723171664
s = 134518680774350230063832525062016995510525346780587574847558817658425801183042670429119234310770
```

Рисунок 2 – Корректный запуск программы

```
ПРОВЕРКА ПОДПИСИ УОЛТЕРОМ

h = 490423192
c = 191
Проверка  $h \leq s^k \bmod n \leq h + 2^c$ 
Проверка  $490423192 \leq 1010053014393849644878333905766177985736334706892779104831 \leq 3138550867693340381917894711603833208051177722232507679640$ 
 $2^c = 3138550867693340381917894711603833208051177722232017256448$ 
Проверка Уолтера: Подпись корректна

ПРОВЕРКА ПОДПИСИ БОБОМ

h = 490423192
c = 191
Проверка  $h \leq s^k \bmod n \leq h + 2^c$ 
Проверка  $490423192 \leq 1010053014393849644878333905766177985736334706892779104831 \leq 3138550867693340381917894711603833208051177722232507679640$ 
Проверка Боба: Подпись корректна
Найденное Бобом секретное сообщение m* = 12329843443
D:\cpp_projects\protocols\lab7_ESIGN\Release>
```

Рисунок 3 – Корректный запуск программы

На рисунке 4 показан некорректный запуск программы, т.к. m^* имеет размер, больший чем требуемая битовая длина числа r .

```
D:\cpp_projects\protocols\lab7_ESIGN\x64\Release>lab7_ESIGN.exe /l:16 /ms:12329843443 /m:message /k:4  
Размер m* должен быть меньше размера r  
D:\cpp_projects\protocols\lab7_ESIGN\x64\Release>
```

Рисунок 4 – Некорректный запуск программы

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;

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;
}

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;
}

// Функция для проверки числа n на простоту с точностью k
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 (cpp_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);
    }

    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;
    }

    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;
    }

    void helpFunc() {
        cout << "Введена команда с /h. Допустимые параметры:";
        cout << "\n\n/l:<length> - Битовая длина чисел p, q, r";
        cout << "\n\n/k:<k> - Параметр безопасности k";
        cout << "\n\n/ms:<ms> - Секретное число Алисы m* (m* < r)";
        cout << "\n\n/m:<message> - Сообщение.";
        cout << "\n\n/h - информация о допустимых параметрах командной строки программы.\n";
    }

```

```

}

cpp_int pow2(cpp_int s, cpp_int k) {
    if (k == 0)
        return 1;
    else if (k == 1)
        return s;
    cpp_int s_start = s;
    for (int i = 0; i < k - 1; i++)
        s *= s_start;
    return s;
}

void generatePQ(int pBits, int qBits, cpp_int& p, cpp_int& q) {
    p = generateRandomPrime(pBits);
    q = generateRandomPrime(qBits);
    while (q == p)
        q = generateRandomPrime(qBits);
    return;
}

void generateKeys(int pBits, int qBits, cpp_int J, map <string, cpp_int>& publicKey,
cpp_int& privateKey) {
    //шаг 1
    cout << "\n\n ГЕНЕРАЦИЯ ОБЩИХ ПАРАМЕТРОВ " << "\n\n";
    cpp_int p, q, n;
    generatePQ(pBits, qBits, p, q);
    cout << "\n p = " << p << "\n";
    int bit_count = msb(p) + 1;
    cout << "\n Количество битов в числе p: " << bit_count << "\n";
    cout << "\n q = " << q << "\n";
    bit_count = msb(q) + 1;
    cout << "\n Количество битов в числе q: " << bit_count << "\n";
    n = p * q;
    cout << "\n n = p * q = " << n << "\n";
    bit_count = msb(n) + 1;
    cout << "\n Количество битов в числе n: " << bit_count << "\n";

    cout << "\n\n ГЕНЕРАЦИЯ ИНДИВИДУАЛЬНЫХ ПАРАМЕТРОВ " << "\n\n";
    //шаг 2
    cpp_int phi = (p - 1) * (q - 1);
    cout << "\n phi(n) = " << phi << "\n";
    cpp_int e;
    while (true) {
        e = generate_random(2, phi - 1);
        if (nod(phi, e) == 1)
            break;
    }
    cout << "\n e = " << e << "\n";

    //шаг 3
    cpp_int x1, y1;
    algEuclidExtended(e, phi, x1, y1);
    if (x1 < 0)
        x1 = x1 + phi;
    cpp_int s = x1;
    cout << "\n s = " << s << "\n";

    //шаг 4
    algEuclidExtended(J, n, x1, y1);
    if (x1 < 0)
        x1 = x1 + n;
    cpp_int j_obr = x1;
}

```

```

    cpp_int x = powMod(j_obr, s, n);

    cout << "\n x = " << x << "\n";

    cpp_int y = powMod(x, e, n);

    cout << "\n y = " << y << "\n";

    publicKey.insert({ "n", n });
    publicKey.insert({ "e", e });
    publicKey.insert({ "y", y });
    privateKey = x;

    cout << "\n Открытый ключ { n, e, y }: { " << n << ", " << e << ", " << y << " }\n";
    cout << "\n Закрытый ключ { x }: { " << x << " }\n";

    return;
}

cpp_int HashFunc(const 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;
}

void generateSign(string m, map <string, cpp_int> publicKey, cpp_int privateKey, cpp_int& d,
cpp_int& z) {

    cout << "\n\n ГЕНЕРАЦИЯ ПОДПИСИ АЛИСОЙ " << "\n\n";

    cpp_int r = generate_random(1, publicKey["n"] - 1);

    cout << "\n r = " << r << "\n";

    cpp_int a = powMod(r, publicKey["e"], publicKey["n"]);

    cout << "\n a = " << a << "\n";

    d = HashFunc(m, a);

    cout << "\n d = " << d << "\n";

    z = (r * powMod(privateKey, d, publicKey["n"])) % publicKey["n"];

    cout << "\n z = " << z << "\n";

    return;
}

void checkSign(string m, cpp_int d, cpp_int z, cpp_int J, map <string, cpp_int> publicKey) {

    cout << "\n\n ПРОВЕРКА ПОДПИСИ БОБОМ " << "\n\n";

    cpp_int a_bob = ((powMod(z, publicKey["e"], publicKey["n"])) * powMod(J, d,
publicKey["n"])) % publicKey["n"];

    cout << "\n a* = " << a_bob << "\n";

```

```

    cpp_int d_bob = HashFunc(m, a_bob);

    cout << "\n d* = " << d_bob << "\n";

    if (d == d_bob)
        cout << "\n d* = d. Проверка подписи пройдена \n";
    else
        cout << "\n d* != d. Проверка подписи не пройдена \n";
    return;
}

void guillouQuisquater_main(int pBits, int qBits, cpp_int J, string m) {
    map <string, cpp_int> publicKey; // n, e, y
    cpp_int privateKey; // x
    generateKeys(pBits, qBits, J, publicKey, privateKey);
    cpp_int d, z;
    generateSign(m, publicKey, privateKey, d, z); //Алиса генерирует

    checkSign(m, d, z, J, publicKey); //Боб проверяет
}

void generatePQR(cpp_int &p, cpp_int &q, cpp_int &r, int bitsLength) {
    p = generateRandomPrime(bitsLength);
    q = generateRandomPrime(bitsLength);
    while (q == p)
        q = generateRandomPrime(bitsLength);
    r = generateRandomPrime(bitsLength);
    while (r == p || r == q)
        r = generateRandomPrime(bitsLength);
    return;
}

cpp_int normalMod(cpp_int n, cpp_int p) {
    while (n < 0)
        n += p;
    n %= p;
    return n;
}

cpp_int generateSign(string aliceMessage, map <string, cpp_int> alicePrivateKey, map
<string, cpp_int> publicKey, cpp_int hashFuncKey, cpp_int aliceSecretMessage) {

    cout << "\n\n ГЕНЕРАЦИЯ ПОДПИСИ " << "\n\n";

    //шаг 1
    cpp_int h = HashFunc(aliceMessage, hashFuncKey) % publicKey["n"];

    cout << "\n h = " << h << "\n";

    //шаг 2
    cpp_int u = generate_random(1, alicePrivateKey["p"] * alicePrivateKey["q"] - 2);

    cout << "\n u = " << u << "\n";

    cpp_int x = aliceSecretMessage + u * alicePrivateKey["r"];

    cout << "\n x = " << x << "\n";

    //шаг 3
    cpp_int a_chisl = h - (powMod(x, publicKey["k"], publicKey["n"]));
    if (a_chisl < 0)
        a_chisl += publicKey["n"];
    cpp_int a_znam = alicePrivateKey["p"] * alicePrivateKey["q"] * alicePrivateKey["r"];

```

```

cpp_int a = a_chisl / a_znam;
if (a_chisl % a_znam != 0)
    a += 1;
cout << "\n a = " << a << "\n";

cpp_int b_dop = publicKey["k"] * pow2(x, publicKey["k"] - 1);
cpp_int x1, y1;
algEuclidExtended(b_dop, alicePrivateKey["p"], x1, y1);
if (x1 < 0)
    x1 = x1 + alicePrivateKey["p"];
b_dop = x1;
cpp_int b = a * b_dop;

cout << "\n b = " << b << "\n";

//war 4
cpp_int s = x + b * alicePrivateKey["p"] * alicePrivateKey["q"] * alicePrivateKey["r"];

cout << "\n s = " << s << "\n";
//war 5
return s;
}

bool wolterCheckSign(cpp_int s, string aliceMessage, map <string, cpp_int> publicKey,
cpp_int hashFuncKey) {

    cout << "\n\n ПРОВЕРКА ПОДПИСИ УОЛТЕРОМ " << "\n\n";
    //war 6
    cpp_int h = HashFunc(aliceMessage, hashFuncKey) % publicKey["n"];

    cout << "\n h = " << h << "\n";

    //war 7
    int c_bit_count = msb(publicKey["n"]) + 1;
    cpp_int c = 3 * c_bit_count;
    if (c % 4 == 0)
        c /= 4;
    else
        c = (c / 4) + 1;

    cout << "\n c = " << c << "\n";

    //war 8
    cpp_int ifS = powMod(s, publicKey["k"], publicKey["n"]);
    cout << " \nПроверка h <= s^k mod n <= h + 2^c\n";
    cout << " \nПроверка " << h << " <= " << ifS << " <= " << h + pow2(2, c) << "\n";
    cout << "\n2^c = " << pow2(2, c) << "\n";
    if (h > ifS || ifS > h + pow2(2, c)) {
        cout << "\nПроверка Уолтера: Подпись некорректна\n";
        return false;
    }
    else {
        cout << "\nПроверка Уолтера: Подпись корректна\n";
        return true;
    }
}

bool bobCheckSign(cpp_int s, string aliceMessage, map <string, cpp_int> publicKey, cpp_int
hashFuncKey, cpp_int r, cpp_int &BOBBEDaliceSecretMessage) {

    cout << "\n\n ПРОВЕРКА ПОДПИСИ БОБОМ " << "\n\n";
    //war 6
    cpp_int h = HashFunc(aliceMessage, hashFuncKey) % publicKey["n"];

```

```

cout << "\n h = " << h << "\n";

//шаг 7
int c_bit_count = msb(publicKey["n"]) + 1;
cpp_int c = 3 * c_bit_count;
if (c % 4 == 0)
    c /= 4;
else
    c = (c / 4) + 1;

cout << "\n c = " << c << "\n";

//шаг 8
cpp_int ifS = powMod(s, publicKey["k"], publicKey["n"]);
cout << " \nПроверка h <= s^k mod n <= h + 2^c\n";
cout << " \nПроверка " << h << " <= " << ifS << " <= " << h + pow2(2, c) << "\n";

if (h > ifS || ifS > h + pow2(2, c)) {
    cout << "\nПроверка Боба: Подпись некорректна\n";
    return false;
}
else {
    cout << "\nПроверка Боба: Подпись корректна\n";
    BOBBEDaliceSecretMessage = s % r;
    return true;
}
}

void esign_main(cpp_int aliceSecretMessage, string aliceMessage, cpp_int k, int bitsLength)
{
    cpp_int hashFuncKey = generate_random(1, aliceSecretMessage);

    cout << "\n\n ГЕНЕРАЦИЯ ПАРАМЕТРОВ " << "\n\n";
    cpp_int p, q, r;
    generatePQR(p, q, r, bitsLength);

    if (aliceSecretMessage >= r) {
        cout << "\nr >= m*\n";
        exit(EXIT_FAILURE);
    }

    cout << "\n p = " << p << "\n";
    int bit_count_p = msb(p) + 1;
    cout << "\nБитовая длина p: " << bit_count_p << "\n";
    cout << "\n q = " << q << "\n";
    int bit_count_q = msb(q) + 1;
    cout << "\nБитовая длина q: " << bit_count_q << "\n";
    cout << "\n r = " << r << "\n";
    int bit_count_r = msb(r) + 1;
    cout << "\nБитовая длина p: " << bit_count_r << "\n";

    cpp_int n = p * p * q * r;
    cout << "\n n = p^2 * q * r = " << n << "\n";
    int bit_count_n = msb(n) + 1;
    cout << "\nБитовая длина n: " << bit_count_n << "\n";
    map <string, cpp_int> publicKey;
    publicKey.insert({ "n", n });
    publicKey.insert({ "k", k });

    cout << "\nОткрытый ключ: {n, k} = {" << publicKey["n"] << ", " << publicKey["k"] <<
    "}\n";

    map <string, cpp_int> alicePrivateKey;

```

```

    alicePrivateKey.insert({ "p", p });
    alicePrivateKey.insert({ "q", q });
    alicePrivateKey.insert({ "r", r });

    cout << "\nЗакрытый ключ Алисы: {p, q, r} = {" << alicePrivateKey["p"] << ", " <<
    alicePrivateKey["q"] << ", " << alicePrivateKey["r"] << "}\n";

    cpp_int s = generateSign(aliceMessage, alicePrivateKey, publicKey, hashFuncKey,
    aliceSecretMessage);

    if (!wolverCheckSign(s, aliceMessage, publicKey, hashFuncKey)) {
        exit(EXIT_FAILURE);
    }

    cpp_int BOBBEDaliceSecretMessage;
    if (!bobCheckSign(s, aliceMessage, publicKey, hashFuncKey, alicePrivateKey["r"],
    BOBBEDaliceSecretMessage)) {
        exit(EXIT_FAILURE);
    }
    cout << "\nНайденное Бобом секретное сообщение m* = " << BOBBEDaliceSecretMessage <<
    "\n";

    return;
}

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

    int bitsLength;
    cpp_int k;
    string aliceMessage;
    cpp_int aliceSecretMessage;
    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);
            char symbol = ',';
            if (ifStr == "/l:") {
                bitsLength = stoi(checkStr.substr(3, checkStr.length()));
            }
            if (ifStr == "/k:") {
                k = stoi(checkStr.substr(3, checkStr.length()));
            }
            if (ifStr == "/ms") { // m* - секретное сообщение
                aliceSecretMessage = (cpp_int)(checkStr.substr(4, checkStr.length()));
            }
            if (ifStr == "/m:") { // m - сообщение
                aliceMessage = checkStr.substr(3, checkStr.length());
            }
        }
    }

    if (k < 4) {
        cout << "\nПараметр безопасности k должен быть >= 4\n";
        return 0;
    }

    int bit_count = msb(aliceSecretMessage) + 1;
    if (bit_count > bitsLength) {
        cout << "\nРазмер m* должен быть меньше размера r\n";
    }
}

```



```
        return 0;
    }

    esign_main(aliceSecretMessage, aliceMessage, k, bitsLength);
    return 0;
}
```