

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 ЭЦП
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 схемы подписи Гиллу-Кискате	4
2 Практическая реализация	5
2.1 Описание программы	5
2.2 Тестирование программы.....	5

ВВЕДЕНИЕ

Цель работы - реализация схемы подписи Гиллу-Кискате.

1 Теоретические сведения

Генерация общих параметров. Доверенный центр T (Трент) публикует большое число $n = p * q$, где p, q — большие простые числа, которые держатся в секрете.

Генерация индивидуальных параметров.

1. T выбирает целое число $e(1 < e < \phi(n))$, взаимно простое с $\phi(n)$, где $\phi(n) = (p - 1)(q - 1)$ - функция Эйлера;
2. T вычисляет $s = e^{-1} \pmod{\phi(n)}$ и $x = J^{-s} \pmod{n}$, где J - битовая строка личной информации о пользователе A , с условием $(J, n) = 1$;
3. T вычисляет $y = x^e \pmod{n}$;
4. Тройка $\{n, e, y\}$ публикуется в качестве открытого ключа A , а x является закрытым ключом пользователя A .

1.1 Описание алгоритма схемы подписи Гиллу-Кискате

Алгоритм схемы подписи Гиллу-Кискате.

Вход: Простое число n , по модулю которого производятся вычисления, сообщение m , битовая строка личной информации о пользователе A .

Выход: Результат проверки подписи.

Алиса подписывает свое сообщение m , используя свою пару открытого и закрытого ключа. Боб проверяет подпись Алисы, используя ее открытый ключ.

Генерация подписи

1. Алиса вычисляет $a = r^e \pmod{n}$, r - случайное число Алисы, $1 \leq r \leq n-1$;
2. Алиса вычисляет $d = (m || a) \pmod{e}$;
3. Алиса вычисляет $z = r * x^d \pmod{n}$;
4. Алиса передает Бобу $\{m, d, z, J\}$.

Проверка подписи

1. Боб вычисляет $a* = z^e * J^d \pmod{n}$;
2. Боб вычисляет $d* = h(m || a*) \pmod{e}$;
3. Боб проверяет, что $d* = d$. Если выполняется, то подпись корректна.

2 Практическая реализация

2.1 Описание программы

Все действия алгоритма в функции *guillouQuisquater_main*.

Публичный и приватный ключи вычисляются в функции *generateKeys*.

Внутри этой же функции происходит генерация больших простых чисел p, q и вычисление по ним n . За это отвечает функция *generatePQ*, где происходит вызов функций *generateRandomPrime* для генерации простого случайного числа в заданном диапазоне, а для проверки числа на простоту с помощью теста Миллера-Рабина используется функция *isPrimeMillerRabin*.

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

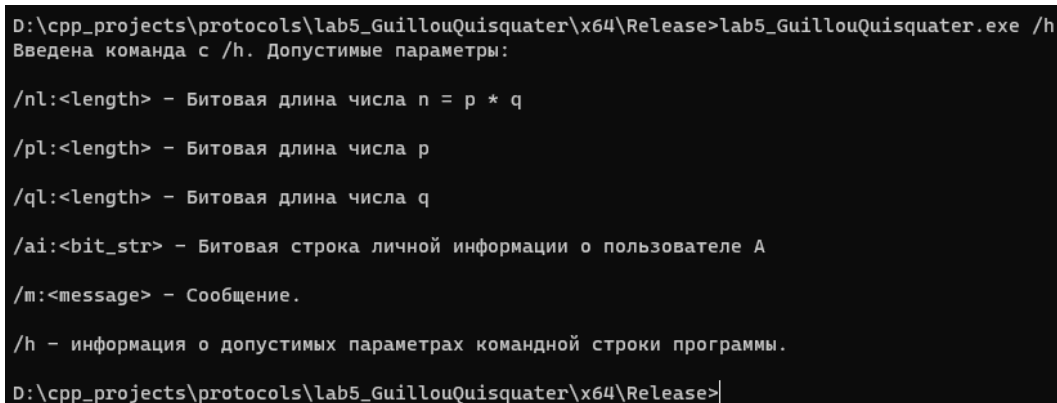
Для вычисления обратного элемента используется расширенный алгоритм Евклида, который реализован в функции *algEuclidExtended*.

Хэш-функция SHA256, вычисляемая с помощью средств из библиотеки *crypto++*, реализована в функции *hashFunc*.

Функция *generateSign* отвечает за генерацию подписи, а функция *checkSign* отвечает за проверку подписи.

2.2 Тестирование программы

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



```
D:\cpp_projects\protocols\lab5_GuillouQuisquater\x64\Release>lab5_GuillouQuisquater.exe /h
Введена команда с /h. Допустимые параметры:

/nl:<length> - Битовая длина числа n = p * q
/pl:<length> - Битовая длина числа p
/ql:<length> - Битовая длина числа q
/ai:<bit_str> - Битовая строка личной информации о пользователе A
/m:<message> - Сообщение.
/h - информация о допустимых параметрах командной строки программы.
D:\cpp_projects\protocols\lab5_GuillouQuisquater\x64\Release>
```

Рисунок 1 – Вызов параметра *help*

В примере, показанном на рисунках 2 и 3, задаются параметры $n_l = 16$, $ai = 1010010101001$, $m = message$.

```
Командная строка
D:\cpp_projects\protocols\lab5_GuillouQuisquater\x64\Release>lab5_GuillouQuisquater.exe /nl:16 /ai:1010010101001 /m:message

ГЕНЕРАЦИЯ ОБЩИХ ПАРАМЕТРОВ

p = 131
Количество битов в числе p: 8
q = 269
Количество битов в числе q: 9
n = p * q = 35239
Количество битов в числе n: 16

ГЕНЕРАЦИЯ ИНДИВИДУАЛЬНЫХ ПАРАМЕТРОВ

phi(n) = 34840
e = 20107
s = 31203
x = 12
y = 12306
Открытый ключ { n, e, y }: { 35239, 20107, 12306 }
Закрытый ключ { x }: { 12 }
```

Рисунок 2 – Первый пример корректной работы схемы подписи Гиллу-Кискате

```
ГЕНЕРАЦИЯ ПОДПИСИ АЛИСой

r = 4228
a = 29238
d = 889
z = 208

ПРОВЕРКА ПОДПИСИ БОБОМ

a* = 29238
d* = 889
d* = d. Проверка подписи пройдена
D:\cpp_projects\protocols\lab5_GuillouQuisquater\x64\Release>
```

Рисунок 3 – Первый пример корректной работы схемы подписи Гиллу-Кискате

В примере, показанном на рисунках 4 и 5, показана работа программы на параметрах $n_l = 1024$, $a_i = 78234532$, $m = message$.

```
Командная строка
D:\cpp_projects\protocols\lab5_GuillouQuisquater\x64\Release>lab5_GuillouQuisquater.exe /nl:1024 /ai:78234532 /m:message

ГЕНЕРАЦИЯ ОБЩИХ ПАРАМЕТРОВ

p = 92849133423476415810573511051827384994146387266760848507321722411284381710705251417709475040099867017350154391353999315245444678166
25427584571349413666691

Количество битов в числе p: 512

q = 16335990873062126323736107621848956562117888031290271476570841641398803283351852342579003409184732770959987430671688821234971224213
788809753558216718045293

Количество битов в числе q: 513

n = p * q = 151678259617763834793396571982483847909743012214301343441478469375429843098836513658128903224563367157908092412426177196776336
336196877707538846226153223010321370573989579254774784864885693401233961299779969974329134830656691709432674548117348480204839704228635211
560914418995523939277153916219595367211723400

Количество битов в числе n: 1024

ГЕНЕРАЦИЯ ИНДИВИДУАЛЬНЫХ ПАРАМЕТРОВ

phi(n) = 151678259617763834793396571982483847909743012214301343441478469375429843098836513658128903224563367157908092412426177196776336
196877707538846226153223010321370573989579254774784864885693401233961299779969974329134830656691709432674548117348480204839704228635211
560914418995523939277153916219595367211723400

e = 1264904262229719627251863039278581954044131253388800808732927181696973916133297786998990243665995968180861109371675439196403090986
502554313246200656689642361207983851723924184573657585093629063283397119693834145979370175629234644892627849211976712181241871129031119
358583987479620573242047195659743991818579

s = 10234075495939246863700945683288094669778804826955476079163982803250552413884584767496719131984367738196293658751570493719268538534
4374251826944787253938021825204297100238381463455297642131084517259792093450731986018347788816483044122949196537421134017259148293385383
80025635363377416139618714133808950981139

x = 141930545757112294704536900489904812571409261102135229918068728594352750321554907609398366644937886452127623981449555353581227741266
7501550268630585084336811016798008355976452929235616874642714763652203838620385608200145053298031346274420955899346730503783160429461884
276701958985372685254634245030619198821626

y = 89278975597797700226492918951715176301304431240063448987705681341234021570834150444341696201799956618262358192124268856330375103225
486082991766026897088368798781475088475374688506936604012376461556126759943890507178583432065525224962249106727900766972170059666728833
1234621817862586443901443587606836037657

Открытый ключ { n, e, y } : { 1516782596177638347933965719824838479097430122143013434414784693754298430988365136581289032245633671579080
9241242617719677633619687770753884622615322301034699147820498902267957832361272509629549382653793624876021649694819641239768983229608479
8933678689895558524024089538652714755084801950153557724933343435463, 1264904262229719627251863039278581954044131253388800808732927181696
973916133297786998990243665995968180861109371675439196403090986502554313246200656689642361207983851723924184573657585093629063283397119
6938341459793701756292346464892627849211976712181241871129031119358583987479620573242047195659743991818579, 8927897559779770022649291895
1715176301304431240063448987705681341234021570834150444341696201799956618262358192124268856330375103225486082991766026897088368798781475
088475374688506936604012376461556126759943890507178583432065525224962249106727900766972170059666728833123462181786258644390144358760683
6037657 }
```

Рисунок 4 – Второй пример корректной работы схемы подписи Гиллу-Кискате

```
Командная строка

Открытый ключ { n, e, y } : { 1516782596177638347933965719824838479097430122143013434414784693754298430988365136581289032245633671579080
9241242617719677633619687770753884622615322301034699147820498902267957832361272509629549382653793624876021649694819641239768983229608479
8933678689895558524024089538652714755084801950153557724933343435463, 1264904262229719627251863039278581954044131253388800808732927181696
973916133297786998990243665995968180861109371675439196403090986502554313246200656689642361207983851723924184573657585093629063283397119
6938341459793701756292346464892627849211976712181241871129031119358583987479620573242047195659743991818579, 8927897559779770022649291895
1715176301304431240063448987705681341234021570834150444341696201799956618262358192124268856330375103225486082991766026897088368798781475
088475374688506936604012376461556126759943890507178583432065525224962249106727900766972170059666728833123462181786258644390144358760683
6037657 }

Закрытый ключ { x } : { 1419305457571122947045369004899048125714092611021352299180687285943527503215549076093983666449378864521276239814
4955353581227741266750155026863058508433681101679800835597645292923561687464271476365220383862038560820014505329803134627442095589934673
0503783160429461884276701958985372685254634245030619198821626 }

ГЕНЕРАЦИЯ ПОДПИСИ АЛИСОЙ

r = 9798241456406178936630948770975325375326547214800846463580811291231385683826802065986076063934032937349248581432534466385851938809
8662616720014763984102315091722884983686494339886917540573092908179715753028533248677879085813944997703983983909845581217347882272741975
77131496383459761931332895730221228636046

a = 74179513570045130019124128178498088673322492268968697846617738194893593189034430855317132381616553608590537034565226694884856025451
4840352698824074901928112632160045065084666403822359942509152695198966372345127679239367004746721197536345440194529968250383594934382437
44962116121835285243511642940404199715056

d = 77492214967865460963471699751820070564084647936440628989923671088450960649501

z = 593114771823183744159711147968852685429034188410546074816946350040380081776614066473340382496424692836180900721362967131827026125982
27355169155543752976238312865522094380298077705343863028689190583053129432759370217454513029115524087836470381214198367534832729427419517
30702985395144004179001104892925030257840

ПРОВЕРКА ПОДПИСИ БОБОМ

a* = 7417951357004513001912412817849808867332249226896869784661773819489359318903443085531713238161655360859053703456522669488485602545
1484035269882407490192811263216004506508466640382235994250915269519896637234512767923936700474672119753634544019452996825038359493438243
744962116121835285243511642940404199715056

d* = 77492214967865460963471699751820070564084647936440628989923671088450960649501

d* = d. Проверка подписи пройдена

D:\cpp_projects\protocols\lab5_GuillouQuisquater\x64\Release>
```

Рисунок 5 – Второй пример корректной работы схемы подписи Гиллу-Кискате

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;
}

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) {
    istream 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/nl:<length> - Битовая длина числа n = p * q";
    cout << "\n\n/pl:<length> - Битовая длина числа p";
    cout << "\n\n/ql:<length> - Битовая длина числа q";
    cout << "\n\n/ai:<bit_str> - Битовая строка личной информации о пользователе A";
    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
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "rus");
    int pBits, qBits, nBits;
    string aliceInformation, aliceMessage;
    bool isN = 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);
            char symbol = ',';
            if (ifStr == "/nl") {
                nBits = stoi(checkStr.substr(4, checkStr.length()));
                if (nBits % 2 == 0) {
                    pBits = nBits / 2;
                    qBits = pBits + 1;
                }
                else {
                    pBits = (nBits + 1) / 2;
                    qBits = pBits;
                }
                isN = true;
            }
            if (ifStr == "/pl") {
                if (!isN)
                    pBits = stoi(checkStr.substr(4, checkStr.length()));
            }
            if (ifStr == "/ql") {
                if (!isN)
                    qBits = stoi(checkStr.substr(4, checkStr.length()));
            }
            if (ifStr == "/ai") { // J - битовая строка (информация Алисы)
                aliceInformation = checkStr.substr(4, checkStr.length());
            }
            if (ifStr == "/m:") { // m - сообщение
                aliceMessage = checkStr.substr(3, checkStr.length());
            }
        }
    }
}

```

```
cpp_int J;
for (char bit : aliceInformation) {
    J *= 2;
    J += (bit - '0');
}

string m = aliceMessage;

guillouQuisquater_main(pBits, qBits, J, m);
return 0;
}
```