

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

Разделение секрета
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

Р. А. Фарахутдинов

подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Теоретические сведения	4
1.1 Описание алгоритма схемы разделения секрета Миньотта	4
2 Практическая реализация	6
2.1 Описание программы	6
2.2 Тестирование программы.....	6

ВВЕДЕНИЕ

Цель работы - реализация схемы разделения секрета Миньотта.

1 Теоретические сведения

Схема Миньотта — пороговая схема разделения секрета, построенная с использованием простых чисел. Позволяет разделить секрет (число) между n сторонами таким образом, что его смогут восстановить любые k участников, но $k - 1$ участников восстановить секрет уже не смогут. Схема основана на китайской теореме об остатках.

В основе схемы лежит использование китайской теоремы об остатках, которая позволяет пользователям, имеющим некоторую долю секрета, восстановить сам секрет, причём единственным образом.

Пусть $k \geq 2, m_1, \dots, m_k \geq 2, b_1, \dots, b_k \in \mathbb{Z}$. Тогда система линейных сравнений

$$(s) \begin{cases} x \equiv b_1 \pmod{m_1} \\ x \equiv b_2 \pmod{m_2} \\ \dots\dots\dots \\ x \equiv b_k \pmod{m_k} \end{cases}$$

имеет решения в \mathbb{Z} тогда и только тогда, когда $b_i \equiv b_j \pmod{(m_i, m_j)} \forall 1 \leq i, j \leq k$. Более того, если приведенная система имеет решения в \mathbb{Z} , она имеет единственное решение в $\mathbb{Z}_{[m_1, \dots, m_k]}$ определяет наименьшее общее кратное m_1, \dots, m_k . В случае, если $(m_i, m_j) = 1 \forall 1 \leq i < j \leq k$, китайскую теорему об остатках называют стандартной.

Пороговая схема разделения секрета Миньотта использует специальные последовательности чисел, названные последовательностями Миньотта. Пусть n — целое, $n \geq 2$, и $2 \leq k \leq n$. (k, n) — последовательность Миньотта — последовательность взаимно простых положительных $p_1 < p_2 < \dots < p_n$ таких, что $\prod_{i=0}^{k-2} p_{n-i} < \prod_{i=1}^k p_i$. Последнее утверждение также равносильно следующему: $\max_{1 \leq i_1 < \dots < i_{k-1} \leq n} (p_{i_1} \dots p_{i_{k-1}}) < \min_{1 \leq i_1 < \dots < i_k \leq n} (p_{i_1} \dots p_{i_k})$

1.1 Описание алгоритма схемы разделения секрета Миньотта

Алгоритм схемы разделения секрета Миньотта.

Вход: Для разделения секрета: число сторон n , между которыми нужно разделить секрет и число сторон k , необходимых для восстановления секрета. Для восстановления секрета: k уравнений.

Выход: Для разделения секрета: Система уравнений, где каждой стороне соответствует одно из уравнений. Для восстановления секрета: Секрет - решение системы уравнений.

Имея открытый ключ-последовательность Миньотта, схема работает так:

1. Секрет S выбирается, как случайное число такое, что $\beta < S < \alpha$, где $\alpha = \prod_{i=1}^k p_i$, $\beta = \prod_{i=0}^{k-2} p_{n-i}$. Другими словами, секрет должен находиться в промежутке между $p_1 * p_2 * \dots * p_k$ и $p_{n-k+2} * \dots * p_n$;
2. Доли вычисляются, как $I_i = S \bmod p_i$, для всех $1 \leq i \leq n$;
3. Имея k различных теней I_{i_1}, \dots, I_{i_k} , можно получить секрет S , используя стандартный вариант китайской теоремы об остатках – им будет единственное решение по модулю $p_{i_1} \dots p_{i_k}$ системы:

$$(s) \begin{cases} x \equiv I_{i_1} \pmod{p_{i_1}} \\ x \equiv I_{i_2} \pmod{p_{i_2}} \\ \dots\dots\dots \\ x \equiv I_{i_k} \pmod{p_{i_k}} \end{cases}$$

Секретом является решение приведенной выше системы, более того, S лежит в пределах $Z_{p_{i_1}, \dots, p_{i_k}}$, т.к. $S < \alpha$. С другой стороны, имея всего $k - 1$ различных теней $I_{i_1}, \dots, I_{i_{k-1}}$, можно сказать, что $S \equiv x_0 \bmod p_{i_1} \dots p_{i_{k-1}}$, где x_0 – единственное решение по модулю $p_{i_1} \dots p_{i_{k-1}}$ исходной системы (в данном случае: $S > \beta \geq p_{i_1} \dots p_{i_{k-1}} > x_0$). Для того, чтобы получить приемлемый уровень безопасности, должны быть использованы (k, n) последовательности Миньотта с большим значением $\frac{\alpha - \beta}{\beta}$. Очевидно, что схема Миньотта не обладает значительной криптографической стойкостью, однако может оказаться удобной в приложениях, где компактность теней является решающим фактором.

2 Практическая реализация

2.1 Описание программы

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

Последовательность Миньотта строится в функции *findMinyottSequence*, в конце данной функции вызывается *isGoodSequence* для проверки последовательности Миньотта на корректность.

Функция *generate_random* генерирует случайное число в заданном промежутке.

Все действия алгоритма восстановления секрета выполняются в функции *minyott_rec*. Функция *getCongruences* считывает систему уравнений из файла. В функции *ChinaXi* реализована китайская теорема об остатках.

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

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

2.2 Тестирование программы

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

```
D:\cpp_projects\protocols\lab6_Minyott\Debug>lab6_Minyott.exe /h
Введена команда с /h. Допустимые параметры:

/n:<n> - Число сторон n, между которыми нужно разделить секрет
/k:<k> - Число сторон k, необходимых для восстановления секрета
/m:<m> - Режим работы - gen (режим разделения секрета) или rec (режим восстановления секрета)
/l:<l> - Модуль для последовательности Миньотта
/input:<filename> - Файл с входными данными для восстановления секрета.
/output:<filename> - Файл, куда будет записываться результат работы программы.
/h - информация о допустимых параметрах командной строки программы.
```

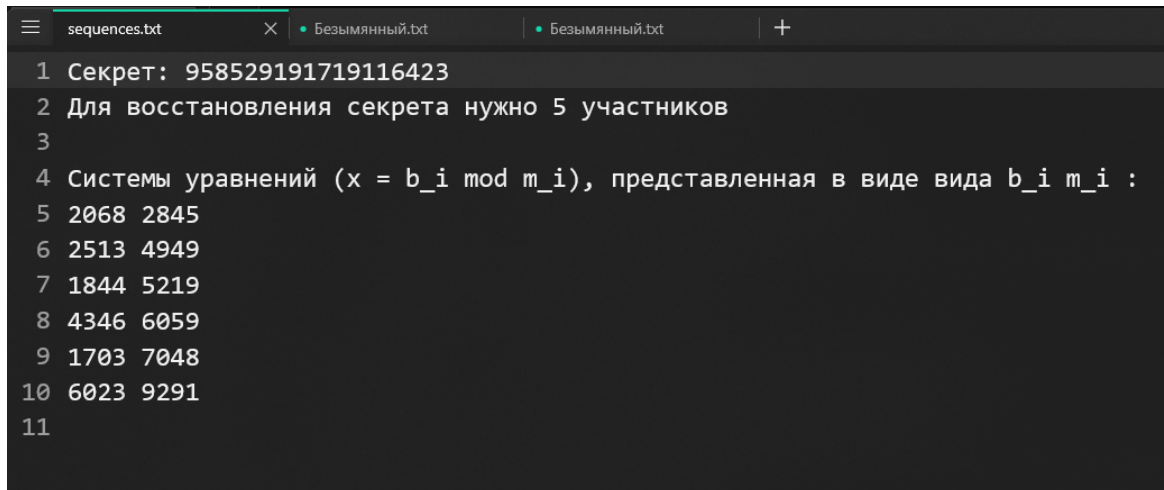
Рисунок 1 – Вызов параметра *help*

На рисунке 2 показан запуск программы в режиме разделения секрета *gen* с параметрами $n = 6$, $k = 5$, $l = 10000$, $output = sequences.txt$.

```
D:\cpp_projects\protocols\lab6_Minyott\Debug>lab6_Minyott.exe /m:gen /n:6 /k:5 /l:10000 /output:sequences.txt
D:\cpp_projects\protocols\lab6_Minyott\Debug>
```

Рисунок 2 – Запуск программы в режиме разделения секрета *gen*

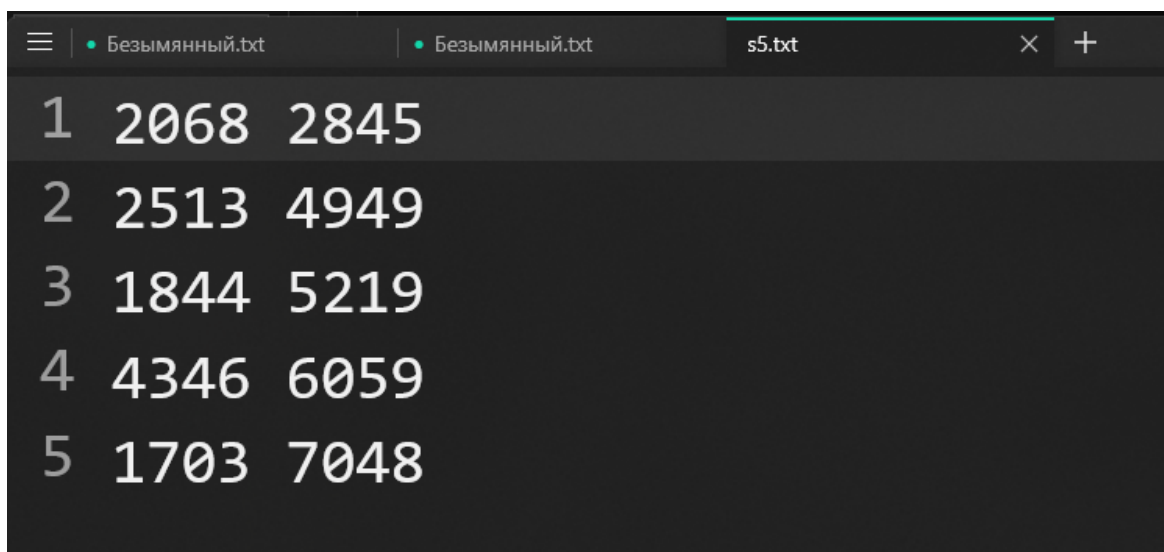
На рисунке 3 показано содержимое файла *sequences.txt*.



```
sequences.txt x Безымянный.txt Безымянный.txt +
1 Секрет: 958529191719116423
2 Для восстановления секрета нужно 5 участников
3
4 Системы уравнений ( $x = b_i \bmod m_i$ ), представленная в виде вида  $b_i \ m_i$  :
5 2068 2845
6 2513 4949
7 1844 5219
8 4346 6059
9 1703 7048
10 6023 9291
11
```

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

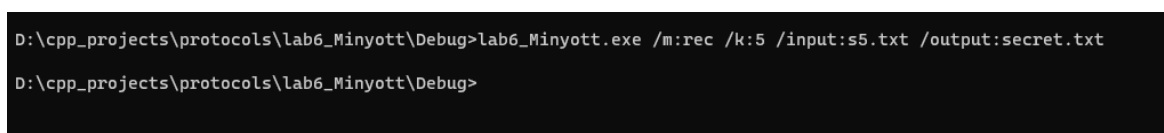
В файл *s5.txt* внесем систему уравнений пяти человек (рисунок 4) и далее данный файл будем передавать в параметрах программы в режиме запуска *rec*.



```
Безымянный.txt Безымянный.txt s5.txt x +
1 2068 2845
2 2513 4949
3 1844 5219
4 4346 6059
5 1703 7048
```

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

На рисунке 5 показан запуск программы в режиме восстановления секрета *rec* с параметрами $k = 5$, $input = s5.txt$, $output = secret.txt$.



```
D:\cpp_projects\protocols\lab6_Minnyott\Debug>lab6_Minnyott.exe /m:rec /k:5 /input:s5.txt /output:secret.txt
D:\cpp_projects\protocols\lab6_Minnyott\Debug>
```

Рисунок 5 – Запуск программы в режиме восстановления секрета *rec*

Результат программы, т.е. восстановленный секрет, показан на рисунке 6.

```
D:\cpp_projects\protocols\lab6_Minnyott\Debug>lab6_Minnyott.exe /m:rec /k:5 /input:s5.txt /output:secret.txt  
D:\cpp_projects\protocols\lab6_Minnyott\Debug>
```

Рисунок 6 – Запуск программы в режиме восстановления секрета *rec*

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 <map>
#include <unordered_set>

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

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;
}

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

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

void helpFunc() {
    cout << "Введена команда с /h. Допустимые параметры:";
    cout << "\n\n/n:<n> - Число сторон n, между которыми нужно разделить секрет";
    cout << "\n\n/k:<k> - Число сторон k, необходимых для восстановления секрета";
    cout << "\n\n/m:<m> - Режим работы - gen (режим разделения секрета) или res (режим
восстановления секрета)";
    cout << "\n\n/l:<l> - Модуль для последовательности Миньотта";
    cout << "\n\n/input:<filename> - Файл с входными данными для восстановления секрета.";
    cout << "\n\n/output:<filename> - Файл, куда будет записываться результат работы
программы.";
    cout << "\n\n/h - информация о допустимых параметрах командной строки программы.\n";
}

//Решение системы линейных сравнений с помощью Греко-Китайской теоремы
cpp_int ChinaXi(vector <pair <cpp_int, cpp_int>> congruences, int k) { // r_i a_i
    vector <cpp_int> m_i(k);
    vector <cpp_int> d_i(k);
    cpp_int M = 1;
    for (int i = 0; i < k; i++) {
        cpp_int a_i = congruences[i].second;
        M *= a_i;
    }
    for (int i = 0; i < k; i++) {
        cpp_int a_i = congruences[i].second;

```

```

        m_i[i] = M / a_i;
        cpp_int x, y;
        algEuclidExtended(m_i[i], a_i, x, y);
        if (x < 0)
            x += a_i;
        d_i[i] = x;
    }
    cpp_int u = 0;
    for (int i = 0; i < k; i++) {
        u += m_i[i] * d_i[i] * congruences[i].first;
        u %= M;
        if (u < 0)
            u += M;
    }
    return u;
}

bool isGoodSequence(vector <cpp_int> minyottSequence, cpp_int k, cpp_int n, cpp_int
&firstMult, cpp_int &secondMult) {
    firstMult = 1;
    secondMult = 1;
    for (int i = 0; i < k - 1; i++)
        firstMult *= minyottSequence[(int)n - 1 - i];

    for (int i = 0; i < k; i++)
        secondMult *= minyottSequence[i];

    if (firstMult < secondMult)
        return true;
    return false;
}

void findMinyottSequence(vector <cpp_int>& minyottSequence, cpp_int k, cpp_int n, cpp_int
modSequence, cpp_int& firstMult, cpp_int& secondMult) {
    while (true) {
        minyottSequence.clear();
        for (int i = 0; i < n; i++) {
            while (true) {
                cpp_int p_i = generate_random(2, modSequence);
                bool nodNot1 = false;
                for (int j = 0; j < minyottSequence.size(); j++) {
                    if (nod(p_i, minyottSequence[j]) != 1) {
                        nodNot1;
                        nodNot1 = true;
                    }
                }
                if (!nodNot1) {
                    minyottSequence.push_back(p_i);
                    break;
                }
            }
        }
        sort(minyottSequence.begin(), minyottSequence.end());
        if (isGoodSequence(minyottSequence, k, n, firstMult, secondMult))
            break;
    }
    return;
}

void minyott_gen(cpp_int k, cpp_int n, const string& output, cpp_int modSequence) {
    vector <cpp_int> minyottSequence;
    cpp_int beta, alpha;
    findMinyottSequence(minyottSequence, k, n, modSequence, beta, alpha);
}

```

```

    cpp_int S = generate_random(beta + 1, alpha - 1);

    ofstream outFile(output, ios::out | ios::trunc);
    if (!outFile.is_open()) {
        cerr << "Ошибка открытия файла\n";
        return;
    }
    outFile << "Секрет: " << S << "\n";
    outFile << "Для восстановления секрета нужно " << k << " участников\n\n";

    outFile << "Системы уравнений ( $x = b_i \bmod m_i$ ), представленная в виде вида  $b_i \bmod m_i$  :";
    "<< "\n";
    for (int i = 0; i < n; i++) {
        outFile << S % minyottSequence[i] << " " << minyottSequence[i] << "\n";
    }

    outFile.close();

    return;
}

void getCongruences(vector<pair<cpp_int, cpp_int>> &congruences, const string& input) {
    ifstream inputFile(input);
    if (!inputFile.is_open()) {
        cerr << "Ошибка открытия файла\n";
        return;
    }

    cpp_int a, b;
    while (inputFile >> a >> b) {
        congruences.push_back(make_pair(a, b));
    }

    inputFile.close();
}

void minyott_rec(const string& input, const string& output, cpp_int k) {

    vector<pair<cpp_int, cpp_int>> congruences;
    getCongruences(congruences, input);
    if (congruences.size() < k) {
        cout << "\n Количество уравнений должно быть больше либо равно " << k << "\n";
        return;
    }
    cpp_int S = ChinaXi(congruences, (int)k);

    ofstream outFile(output, ios::out | ios::trunc);
    if (!outFile.is_open()) {
        cerr << "Ошибка открытия файла\n";
        return;
    }

    outFile << "Секрет: " << S << "\n";

    outFile.close();
    return;
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "rus");
    string mode = "gen";
    string input = "null", output = "null";
    cpp_int k = 0;
    cpp_int n = 0;

```

```

cpp_int modSequence = 10000;
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 ifStr2 = findInStr(checkStr, 7);
        string ifStr3 = findInStr(checkStr, 8);
        char symbol = ',';
        if (ifStr == "/n:") { //разделить между n сторонами
            n = stoi(checkStr.substr(3, checkStr.length()));
        }
        if (ifStr == "/k:") { //k - кол-во для восстановления
            k = stoi(checkStr.substr(3, checkStr.length()));
        }
        if (ifStr == "/m:") { //режим gen или rec
            mode = checkStr.substr(3, checkStr.length());
        }
        if (ifStr == "/l:") { //модуль для последовательности Миньотта
            modSequence = stoi(checkStr.substr(3, checkStr.length()));
        }
        if (ifStr2 == "/input:") {
            input = checkStr.substr(7, checkStr.length());
        }
        if (ifStr3 == "/output:") {
            output = checkStr.substr(8, checkStr.length());
        }
    }
}
if (mode == "gen") {
    if (k == 0 || n == 0 || output == "null") {
        cout << "\n Не хватает параметров \n";
        return 0;
    }
    if (n < 2) {
        cout << "\n n < 2 \n";
        return 0;
    }
    if (k < 2) {
        cout << "\n k < 2 \n";
        return 0;
    }
    if (k > n) {
        cout << "\n k > n \n";
        return 0;
    }
    minyott_gen(k, n, output, modSequence);
    return 0;
}
if (mode == "rec") {
    if (input == "null" || output == "null" || k == 0) {
        cout << "\n Не хватает параметров \n";
        return 0;
    }
    minyott_rec(input, output, k);
    return 0;
}
return 0;
}

```