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
Гельфанова Даниила Руслановича

Преподаватель
аспирант _____ Р. А. Фарахутдинов
подпись, дата

Саратов 2023

1 Постановка задачи

Необходимо реализовать схему разделения секрета Карнина-Грина-Хеллмана.

2 Теоретические сведения

Разделение секрета — термин в криптографии, под которым понимают любой из способов распределения секрета среди группы участников, каждому из которых достаётся своя некая доля. Секрет может воссоздать только коалиция участников из первоначальной группы, причём входить в коалицию должно не менее некоторого изначально известного их числа.

Схемы разделения секрета применяются в случаях, когда существует значимая вероятность компрометации одного или нескольких хранителей секрета, но вероятность недобросовестного сговора значительной части участников считается пренебрежимо малой.

Существующие схемы имеют две составляющие: разделение и восстановление секрета. К разделению относится формирование частей секрета и распределение их между членами группы, что позволяет разделить ответственность за секрет между её участниками. Обратная схема должна обеспечить его восстановление при условии доступности его хранителей в некотором необходимом количестве.

В схеме Карнина-Грини-Хеллмана используется матричное умножение в конечных полях. Выбирается $n + 1$ таких t-мерных векторов $V_0, V_1, \dots, V_n$, что ранг любой матрицы размером $t \times t$, образованной из этих векторов, равен t. Вектор U – это вектор размерности t.

M – это матричное произведение UV_0^T. Долями секрета являются произведения $\alpha_i = UV_i^T$, где i меняется от 1 до n.

Любые m долей можно использовать для решения системы линейных уравнений размерности $m \times m$, неизвестными являются коэффициенты U. V_0 известно всем, остальные V_i известны i-м участникам.

Восстановление секрета происходит с помощью решения системы линейных алгебраических уравнений из t уравнений и t неизвестных относительно компонент вектора U:

$$\begin{cases} U^1V_1^1 + U^2V_1^2 + \cdots + U^tV_1^t = \alpha_1 \\ \vdots \\ U^1V_t^1 + U^2V_t^2 + \cdots + U^tV_t^t = \alpha_t \end{cases} \quad (*)$$

Система имеет единственное решение, так как на этапе распределения секрета выбирались линейно независимые векторы, значит определитель матрицы не равен нулю. Найдя вектор U, легко восстановить секрет путем вычисления произведения UV_0^T.

Алгоритм работы схемы Карнина-Грина-Хеллмана:

Вход: простое число p, количество участников n, минимальное число участников для раскрытия секрета t.

Выход: «Найденное решение не является правильным. Секрет не может быть раскрыт.» или «Найденное решение является правильным. Секрет успешно раскрыт.»

Шаг 1. Сгенерировать попарно линейно независимые векторы $U, V_0, \dots, V_n$.

Шаг 2. Вычислить доли секрета UV_i^T для i от 1 до n.

Шаг 3. Вычислить секрет $M = UV_0^T$.

Шаг 4. Случайным образом выбрать t участников и из их долей составить СЛУ (*).

Шаг 5. Решить СЛУ относительно компонент вектора U и получить $U_{\text{СЛУ}} = \{u_1, \dots, u_t\}$.

Шаг 6. Вычислить значение секрета $U_{\text{СЛУ}}V_0^T$ и сравнить его с M. Если значение совпадает, то выдать «Найденное решение является правильным. Секрет успешно раскрыт.», иначе выдать «Найденное решение не является правильным. Секрет не может быть раскрыт.».

Если злоумышленник получит меньше, чем t значений, система будет иметь бесконечное количество решений и любое из них равновероятно может

быть искомым секретом. Это означает, что схема Карнина-Грина-Хеллмана является совершенной. Однако, размер каждой доли секрета в t раз больше самого секрета, что делает схему не идеальной.

3 Результаты работы

3.1 Сведения о программе

Программа была реализована на языке программирования Java. В ней есть 5 классов: KGH, $KGHService$, $Matrix$, $MatrixService$ и SLE.

В классе KGH происходит считывание входных параметров: чисел p, n и t.

Класс $KGHService$– класс реализации самого протокола. Для инициализации передаются p, n и t. В нем описаны следующие методы:

- `private void scheme()` – логика самой схемы;
- `private void checkSolution(ArrayList<BigInteger> solution)` – проверка полученного решения СЛУ;
- `private void generateVectors()` – генерация векторов;
- `private void setShares()` – вычисление долей секрета.
- `private ArrayList<Matrix> combination()` – получение всех возможных сочетаний матриц размерности $t \times t$ из векторов;
- `private boolean checkLinearDependency(ArrayList<Matrix> combination)` – проверка попарной линейной независимости векторов.

Класс $Matrix$ – класс объекта матрицы над полем $GF(p)$.

Класс $MatrixService$ – класс с описанием методов с матричными операциями, такими как умножение матриц, умножение числа на строку матрицы, смена строк местами, вычисление определителя матрицы.

Класс SLE – класс, для инициализации которого подается матрица СЛУ. В нем описаны методы для нахождения решения СЛУ методом Гаусса.

3.2 Тестирование программы

На рисунке 1 представлено негативное тестирование программы.

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 6. Karnin-Greene-Hellman\out\artifacts\KGH_jar> java -jar KGH.jar
Входные параметры отсутствуют
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 6. Karnin-Greene-Hellman\out\artifacts\KGH_jar> java -jar KGH.jar 10
Передано некорректное число параметров.
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 6. Karnin-Greene-Hellman\out\artifacts\KGH_jar> java -jar KGH.jar 10 10 7
Передано составное число.
Ошибка в чтении входных параметров.
```

Рисунок 1 – Негативное тестирование

На рисунках 2-3 представлено положительное тестирование программы.

```
PS D:\Учеба\5.1 курс\Криптографические протоколы\Task 6. Karnin-Greene-Hellman\out\artifacts\KGH_jar> java -jar KGH.jar 1031 10 7
Схема Карнина-Грини-Хеллмана. Параметры:
   Количество участников n = 10
   Минимальное число участников для раскрытия секрета t = 7
   Вычисления проводятся в поле F1031

U = 494 381 255 261 962 946 901
V0 = 959 221 35 943 961 220 398
V1 = 794 901 615 374 1010 903 869
V2 = 188 69 946 685 764 618 914
V3 = 878 291 585 371 835 765 763
V4 = 694 756 98 343 24 222 735
V5 = 192 79 893 766 972 465 777
V6 = 72 348 936 802 1013 992 596
V7 = 255 898 748 316 634 477 648
V8 = 563 756 993 451 586 574 546
V9 = 228 759 911 569 495 59 326
V10 = 112 331 1005 362 814 256 1010

Секрет M = UV = 941

Доли секрета:
UV1 = 593
UV2 = 655
UV3 = 698
UV4 = 399
UV5 = 631
UV6 = 929
UV7 = 588
UV8 = 537
UV9 = 1026
UV10 = 267
```

Рисунок 2 – Положительное тестирование, генерация векторов и вычисление долей секрета (входные параметры: 1031, 10, 7)

```
Для проверки выбраны следующие t участников: [1, 2, 3, 5, 6, 7, 10]
Они составили систему линейных уравнений. Матрица системы:
794 901 615 374 1010 903 869 593
188 69 946 685 764 618 914 655
878 291 585 371 835 765 763 698
192 79 893 766 972 465 777 631
72 348 936 802 1013 992 596 929
255 898 748 316 634 477 648 588
112 331 1005 362 814 256 1010 267
Решение системы: [494, 381, 255, 261, 962, 946, 901]
Участники вычислили значение секрета: 941
Найденное решение является правильным. Секрет успешно раскрыт.
```

Рисунок 3 – Положительное тестирование, выбор участников, решение СЛУ и проверка

ПРИЛОЖЕНИЕ А

Листинг программы

```
import java.math.BigInteger;
import java.security.SignatureException;

public class KGH {
    public static void main(String[] args) throws SignatureException {
        if (args.length == 0) {
            System.out.println("Входные параметры отсутствуют");
            return;
        }
        if (args[0].equals("/help") || args[0].equals("h")) {
            System.out.println("""
                    Программе должны передаваться следующие параметры:
                    \t- простое число p (вычисления проводятся в
конечном поле GF(p))
                    \t- количество участников n
                    \t- минимальное число участников для раскрытия
секрета t""");
            return;
        }
        if (args.length < 3) {
            System.out.println("Передано некорректное число
параметров.");
            return;
        }
        BigInteger p;
        int n, t;
        try {
            p = new BigInteger(args[0]);
            if (!p.isProbablePrime(50)){
                System.out.println("Передано составное число.");
                throw new IllegalArgumentException();
            }
            n = Integer.parseInt(args[1]);
            if (n < 2){
                System.out.println("Число n должно быть больше или равно
2.");
                throw new IllegalArgumentException();
            }
            t = Integer.parseInt(args[2]);
            if (t < 2 || t > n) {
                System.out.println("Число t должно быть больше или равно
2 и не более, чем n.");
                throw new IllegalArgumentException();
            }
        } catch (IllegalArgumentException e) {
            System.out.println("Ошибка в чтении входных параметров.");
            return;
        }
        KGHService service = new KGHService(p, n, t);
//        SecureRandom rnd = new SecureRandom();
//       KGHService service = new KGHService(BigInteger.probablePrime(4,
rnd), 3, 10, 7);
    }
```

```
}


import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;

public class KGHService {
    private final BigInteger p;
    private final int n, t; // n - участники, t - минимум для раскрытия
секрета
    private ArrayList<Matrix> V;
    private Matrix U, trU;
    private BigInteger secret;
    private ArrayList<BigInteger> shares = new ArrayList<>(); // доли
секрета

    public KGHService(BigInteger p, int n, int t) {
        this.p = p;
        this.n = n;
        this.t = t;
        System.out.println(this);
        this.scheme();
    }

    @Override
    public String toString() {
        return String.format("""
                        Схема Карнина-Грини-Хеллмана. Параметры:
                           Количество участников n = %d
                           Минимальное число участников для раскрытия
секрета t = %d
                           Вычисления проводятся в поле GF(%d)
                           """,
                n, t, p);
    }

    private void scheme() {
        generateVectors();
        setShares();
        System.out.println("U = " + U);
        for (int i = 0; i < V.size(); ++i) {
            System.out.println("V" + i + " = " + V.get(i));
        }
        this.secret                =                MatrixService.mult(U,
MatrixService.transpose(V.get(0))).get(0).get(0);
        System.out.println("\nСекрет M = UV0 = " + secret);
        System.out.println("\nДоли секрета: ");
        for (int i = 0; i < shares.size(); ++i) {
            System.out.println("UV" + (i + 1) + " = " + shares.get(i));
        }
        ArrayList<Integer> equastions = new ArrayList<>();
        for (int i = 1; i <= n; ++i) {
            equastions.add(i);
        }
```

```
        while (equastions.size() != t) {
            equastions.remove(new Random().nextInt(equastions.size()));
        }
        System.out.println("\nДля проверки выбраны следующие t
участников: " + equastions);
        System.out.println("Они составили систему линейных уравнений.
Матрица системы:");
        SLE system = new SLE(V, shares, equastions, p);
        System.out.println("Решение системы: " + system.getSolution());
        checkSolution(system.getSolution());
    }

    private void checkSolution(ArrayList<BigInteger> solution) {
        for (int i = 0; i < solution.size(); ++i) {
            if (!solution.get(i).equals(U.get(0).get(i))) {
                System.out.println("Найденное решение не является
правильным. Секрет не может быть раскрыт.");
                return;
            }
        }
        BigInteger calculatedSecret =
MatrixService.mult(getMatrix(solution),
MatrixService.transpose(V.get(0))).get(0).get(0);
        System.out.println("Участники вычислили значение секрета: " +
calculatedSecret);
        if (calculatedSecret.equals(secret)) {
            System.out.println("Найденное решение является правильным.
Секрет успешно раскрыт.");
        } else {
            System.out.println("Ошибка в подсчете значения секрета.");
        }
    }

    private void generateVectors() {
        this.U = generateVector(t);
        this.trU = MatrixService.transpose(this.U);
        do {
            V = new ArrayList<>();
            for (int i = 0; i <= n; ++i) {
                V.add(generateVector(t));
            }
        } while (!checkLinearDependency(combination()));
    }

    private void setShares() {
        for (int i = 1; i < V.size(); ++i) {
            shares.add(MatrixService.mult(U,
MatrixService.transpose(V.get(i))).get(0).get(0));
        }
    }

    private ArrayList<Matrix> combination() {
        ArrayList<Matrix> subsets = new ArrayList<>();
        int[] s = new int[t];
        if (t <= V.size()) {
            for (int i = 0; (s[i] = i) < t - 1; ++i) ;
            subsets.add(getSubset(s));
            for (; ; ) {
```

```
                int i;
                for (i = t - 1; i >= 0 && s[i] == V.size() - t + i; --
i) ;
                if (i < 0) {
                    break;
                }
                s[i]++;
                for (++i; i < t; ++i) {
                    s[i] = s[i - 1] + 1;
                }
                subsets.add(getSubset(s));
            }
        }
        return subsets;
    }

    private Matrix getSubset(int[] subset) {
        ArrayList<ArrayList<BigInteger>> vi = new ArrayList<>();
        for (int i = 0; i < subset.length; ++i) {
            vi.add(V.get(subset[i]).get(0));
        }
        return new Matrix(vi, p);
    }

    private       boolean       checkLinearDependency(ArrayList<Matrix>
combination) {
        for (Matrix each : combination) {
            if
(MatrixService.determinant(each).equals(BigInteger.ZERO)) {
                return false;
            }
        }
        return true;
    }

    private Matrix generateVector(int t) {
        SecureRandom rnd = new SecureRandom();
        ArrayList<BigInteger> res = new ArrayList<>();
        for (int i = 0; i < t; ++i) {
            res.add(new BigInteger(p.bitLength(), rnd).mod(p));
        }
        return getMatrix(res);
    }

    private Matrix getMatrix(ArrayList<BigInteger> vector) {
        ArrayList<ArrayList<BigInteger>> matrix = new ArrayList<>();
        matrix.add(vector);
        return new Matrix(matrix, p);
    }
}


import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Collections;

public class SLE {
    private final int t;
```

```
    private final BigInteger p;
    private final ArrayList<BigInteger> solution;
    private Matrix system;

    public SLE(ArrayList<Matrix> V, ArrayList<BigInteger> shares,
ArrayList<Integer> indexes, BigInteger p) {
        this.t = indexes.size();
        this.p = p;
        ArrayList<ArrayList<BigInteger>> matrix = new ArrayList<>();
        for (int i = 0; i < t; ++i) {
            int curIndex = indexes.get(i);
            ArrayList<BigInteger> row = new ArrayList<>();
            for (int j = 0; j < t; ++j) {
                row.add(V.get(curIndex).get(0).get(j));
            }
            row.add(shares.get(curIndex - 1));
            matrix.add(row);
        }
        this.system = new Matrix(matrix, p);
        System.out.println(this);
        this.gauss();
        this.solution = this.setSolution();
    }

    private void gauss() {
        int rows = system.getRow();
        int cols = system.getCol();
        int numPivots = 0;
        for (int j = 0; j < cols && numPivots < rows; ++j) {
            int pivotRow = numPivots;
            while (pivotRow < rows &&
system.get(pivotRow).get(j).equals(BigInteger.ZERO)) {
                ++pivotRow;
            }
            if (pivotRow == rows) {
                continue;
            }
            system = MatrixService.swapRows(system, numPivots,
pivotRow);
            pivotRow = numPivots;
            ++numPivots;
            system = MatrixService.multiplyRow(system, pivotRow,
system.get(pivotRow).get(j).modInverse(p));
            for(int i = pivotRow + 1; i < rows; ++i){
                system = MatrixService.addRows(system, pivotRow, i,
system.get(i).get(j).negate().mod(p));
            }
        }
        for (int i = numPivots - 1; i >= 0; --i) {
            int pivotCol = 0;
            while (pivotCol < cols &&
system.get(i).get(pivotCol).equals(BigInteger.ZERO)) {
                ++pivotCol;
            }
            if (pivotCol == cols) {
                continue;
            }
            for (int j = i - 1; j >= 0; --j) {
```

```
                system = MatrixService.addRows(system, i, j,
system.get(j).get(pivotCol).negate().mod(p));
            }
        }
    }

    @Override
    public String toString(){
        return system.toString();
    }

    private ArrayList<BigInteger> setSolution() {
        ArrayList<BigInteger> solution = new ArrayList<>();
        for(int i = 0; i < system.getRow(); ++i) {
            try {
                solution.add(system.get(i).get(t));
            } catch (IndexOutOfBoundsException e) {
                System.out.println("Ошибка в вычислении решении СЛУ.
Решение не может быть найдено.");
            }
        }
        return solution;
    }

    public ArrayList<BigInteger> getSolution() {
        return solution;
    }
}


import java.math.BigInteger;
import java.rmi.MarshalException;
import java.util.ArrayList;

class IncorrectMatrixException extends Exception {
    public IncorrectMatrixException(String s) {
        super(s);
    }
}

public class Matrix {
    private int row, col;
    private BigInteger p;
    private ArrayList<ArrayList<BigInteger>> m;

    public int getRow() {
        return row;
    }

    public int getCol() {
        return col;
    }

    public BigInteger getP() {
        return p;
    }

    public ArrayList<BigInteger> get(int i) {
```

```
        return m.get(i);
    }

    public void set(int i, ArrayList<BigInteger> row){
        m.set(i, row);
    }

    public void set(int i, int j, BigInteger n){
        m.get(i).set(j, n);
    }

    @Override
    public String toString() {
        String matrix = "";
        for (int i = 0; i < row; ++i) {
            for (int j = 0; j < col; ++j) {
                matrix += m.get(i).get(j) + " ";
            }
            if (i != row-1) {
                matrix += "\n";
            }
        }
        return matrix;
    }

    public Matrix(ArrayList<ArrayList<BigInteger>> m, BigInteger p) {
        try {
            if (m == null || m.size() == 0 || m.get(0).size() == 0) {
                throw      new      IncorrectMatrixException("Передана
некорректная матрица");
            }
            this.m = m;
            this.row = m.size();
            this.col = m.get(0).size();
            this.p = p;
        } catch (IncorrectMatrixException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
    }
}


import java.math.BigInteger;
import java.util.ArrayList;

public class MatrixService {
    public static Matrix mult(Matrix m1, Matrix m2) {
        int row1 = m1.getRow();
        int col1 = m1.getCol();
        int row2 = m2.getRow();
        int col2 = m2.getCol();
        if (col1 != row2) {
            System.out.println("Неправильная  размерность  матриц  для
произведения");
            return null;
        }
        ArrayList<ArrayList<BigInteger>> res = new ArrayList<>();
```

```
        for (int i = 0; i < row1; ++i) {
            ArrayList<BigInteger> row = new ArrayList<>();
            for (int j = 0; j < col2; ++j) {
                row.add(multIJ(m1, m2, i, j));
            }
            res.add(row);
        }
        return new Matrix(res, m1.getP());
    }

    private static BigInteger multIJ(Matrix m1, Matrix m2, int row, int
col) {
        BigInteger res = BigInteger.ZERO;
        for (int k = 0; k < m1.get(0).size(); ++k) {
            res                                                          =
res.add(m1.get(row).get(k).multiply(m2.get(k).get(col))).mod(m1.getP()
);
        }
        res = res.mod(m1.getP());
        return res;
    }

    public static Matrix transpose(Matrix m) {
        // System.out.println(m);
        ArrayList<ArrayList<BigInteger>> res = new ArrayList<>();
        for (int i = 0; i < m.getCol(); ++i) {
            ArrayList<BigInteger> row = new ArrayList<>();
            for (int j = 0; j < m.getRow(); ++j) {
                // System.out.println("j, i = " + j + ", " + i);
                row.add(m.get(j).get(i));
            }
            res.add(row);
        }
        return new Matrix(res, m.getP());
    }

    public static BigInteger determinant(Matrix m) {
        BigInteger p = m.getP();
        int col = m.getCol();

        //System.out.printf("Вызов det: %d, %d", p, col);

        BigInteger det = BigInteger.ZERO;
        if (col == 1) {
            det = m.get(0).get(0);
        } else if (col == 2) {
            //System.out.println("test " + p);
            det = m.get(0).get(0).multiply(m.get(1).get(1))

.subtract(m.get(1).get(0).multiply(m.get(0).get(1))).mod(p);
        } else {
            BigInteger k = BigInteger.ONE;
            for (int i = 0; i < col; ++i) {
                int col_1 = col - 1;
                Matrix tmp = getMatr(m, col, 0, i);
//                System.out.println("\nтемп матр:" + tmp);
                det                                                      =
det.add(k.multiply(m.get(0).get(i)).multiply(determinant(tmp)));
```

```
//                System.out.println("i = " + i + ": " + det);
//                det = det.mod(p);
                k = k.negate();
                //det                                                    =
det.add(m.get(0).get(i).multiply(BigInteger.ONE.pow(i)).multiply(deter
minant(tmp))).mod(p);
            }
        }
        return det.mod(p);
    }

    public static Matrix swapRows(Matrix m, int row0, int row1) {
        if (row0 < 0 || row0 >= m.get(0).size() || row1 < 0 || row1 >=
m.get(0).size())
            throw new IndexOutOfBoundsException("Выход за гарницы
массива");
        ArrayList<BigInteger> tmp = m.get(row0);
        m.set(row0, m.get(row1));
        m.set(row1, tmp);
        return m;
    }

    public static Matrix multiplyRow(Matrix m, int row, BigInteger
factor) {
        if (row < 0 || row >= m.get(0).size()) {
            throw new IndexOutOfBoundsException("Выход за гарницы
массива");
        }
        for (int j = 0, cols = m.getCol(); j < cols; j++) {
            m.set(row,                                                  j,
m.get(row).get(j).multiply(factor).mod(m.getP()));
        }
        return m;
    }

    // destRow += srcRow * factor
    public static Matrix addRows(Matrix m, int srcRow, int destRow,
BigInteger factor) {
        if (srcRow < 0 || srcRow >= m.get(0).size() || destRow < 0 ||
destRow >= m.get(0).size()) {
            throw new IndexOutOfBoundsException("Выход за гарницы
массива");
        }
        for (int j = 0, cols = m.getCol(); j < cols; j++) {
            m.set(destRow,                                              j,
m.get(destRow).get(j).add(m.get(srcRow).get(j).multiply(factor)).mod(m
.getP()));
        }
        return m;
    }

    private static Matrix getMatr(Matrix m, int col, int indRow, int
indCol) {
        ArrayList<ArrayList<BigInteger>> res = new ArrayList<>();
        int col_1 = col - 1;
        for(int i = 0; i < col_1; ++i){
            ArrayList<BigInteger> row = new ArrayList<>();
            for(int j = 0; j < col_1; ++j){
```

```
                row.add(BigInteger.ZERO);
            }
            res.add(row);
        }
        for(int i = 0, ki = 0; i < col; ++i){
            if (i != indRow){
                for(int j = 0, kj = 0; j < col; ++j){
                    if (j != indCol){
                        res.get(ki).set(kj, m.get(i).get(j));
                        ++kj;
                    }
                }
                ++ki;
            }
        }
        return new Matrix(res, m.getP());
    }

    public static Matrix inverse(Matrix m){
        int col = m.getCol();
        BigInteger p = m.getP();
        BigInteger det = determinant(m);
        if (det.equals(BigInteger.ZERO)){
            System.out.println("Матрица вырожденная и обратной не
имеет.");
            return null;
        }
        ArrayList<ArrayList<BigInteger>> res = new ArrayList<>();
        for(int i = 0; i < col; ++i){
            ArrayList<BigInteger> row = new ArrayList<>();
            for(int j = 0; j < col; ++j){
                Matrix tmp = getMatr(m, col, i, j);
                BigInteger a = determinant(tmp);
                if ((i + j + 2) % 2 != 0){
                    a = a.negate();
                }
                a = a.mod(p);
                row.add(a.multiply(det.modInverse(p)).mod(p));
            }
            res.add(row);
        }
        return transpose(new Matrix(res, p));
    }
}
```