Министерство образования Республики Беларусь Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ

И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №4

По теме «Асимметричная криптография. Алгоритм Мак-Элиса»

Выполнила: студентка гр. 053501 Шурко Т.А.

Проверил: ассистент кафедры информатики Лещенко Е. А.

Минск 2023

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc147869480)

[1 Теоретические сведения 4](#_Toc147869481)

[1.1 Алгоритм Мак-Элиса 4](#_Toc147869482)

[1.2 Блок-схема алгоритма 4](#_Toc147869483)

[2 Результат выполнения задачи 6](#_Toc147869484)

[Заключение 7](#_Toc147869485)

[Приложение А 8](#_Toc147869486)

# ВВЕДЕНИЕ

2. Шифрование — обратимое преобразование информации в целях сокрытия от неавторизованных лиц, с предоставлением, в это же время, авторизованным пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения конфиденциальности передаваемой информации. Важной особенностью любого алгоритма шифрования является использование ключа, который утверждает выбор конкретного преобразования из совокупности возможных для данного алгоритма.
3. Пользователи являются авторизованными, если они обладают определенным аутентичным ключом. Вся сложность и, собственно, задача шифрования состоит в том, как именно реализован этот процесс.
4. В целом, шифрование состоит из двух составляющих — зашифрование и расшифрование.
5. С помощью шифрования обеспечиваются три состояния безопасности информации. Первой является конфиденциальность: шифрование используется для скрытия информации от неавторизованных пользователей при передаче или при хранении. Второе — целостность. Шифрование используется для предотвращения изменения информации при передаче или хранении. Последнее это идентифицируемость. Шифрование используется для аутентификации источника информации и предотвращения отказа отправителя информации от того факта, что данные были отправлены именно им.
6. Для выполнения лабораторной работы 4 необходимо реализовать программное средство зашифрования и расшифрования текстовых файлов используя алгоритм Мак-Элиса для криптостойких размеров порождающей матрицы.

## 1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 1.1 Алгоритм Мак-Элиса

McEliece — [криптосистема](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) с [открытыми ключами](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87) на основе теории алгебраического кодирования, разработанная в [1978 году](https://ru.wikipedia.org/wiki/1978_%D0%B3%D0%BE%D0%B4) [Робертом Мак-Элисом](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B0%D0%BA-%D0%AD%D0%BB%D0%B8%D1%81,_%D0%A0%D0%BE%D0%B1%D0%B5%D1%80%D1%82&action=edit&redlink=1). Это была первая схема, использующая [рандомизацию](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%BD%D0%B4%D0%BE%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) в процессе шифрования. Алгоритм не получил широко признания в [криптографии](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F), но в то же время является кандидатом для [постквантовой криптографии](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D0%BA%D0%B2%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F), так как устойчив к атаке с использованием [Алгоритма Шора](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D0%BE%D1%80%D0%B0).

## 1.2 Блок-схема алгоритма

В данном подразделе на рисунке 1.1 указана блок-схема генерации ключей криптосистемы Мак-Элиса, основными шагами которой являются генерация порождающей матрицы G, случайной невырожденной матрицы S, случайной матрицы перестановки P.



Рисунок 1.1 — Блок-схема алгоритма генерации ключей криптосистемы Мак-Элиса

На рисунке 1.2 блок-схема алгоритма шифрования сообщения криптосистемы Мак-Элиса, для которого необходимо сгенерировать ещё один вектор — вектор ошибок.



Рисунок 1.2 — Блок-схема алгоритма шифрования сообщения криптосистемы Мак-Элиса

На рисунке 1.3 блок-схема алгоритма расшифрования сообщения криптосистемы Мак-Элиса.



Рисунок 1.3 — Блок-схема алгоритма расшифрования сообщения криптосистемы Мак-Элиса

# 2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ЗАДАЧИ

В результате выполнения задачи было реализовано зашифрование и расшифрование текстовых файлов с помощью криптосистемы Мак-Элиса на языке программирования С#. Текст для шифрования необходимо поместить в файл и передать его наименование в виде первого параметра, текст после применения алгоритма будет помещен в файл с наименованием, указанным во втором параметре. Текст исполняемой программы в приложении А.

Для запуска программного продукта необходимо подключить классы Crypt и Matrix, а также указать файл, из которого берется текст и файл, в который будет помещаться результат работы (Рисунок 2.1). Для шифрования необходимо вызвать статический метод класса Crypt EncryptMessageFromFile. Аналогично, для дешифрования используется метод DecryptMessageFromFile.

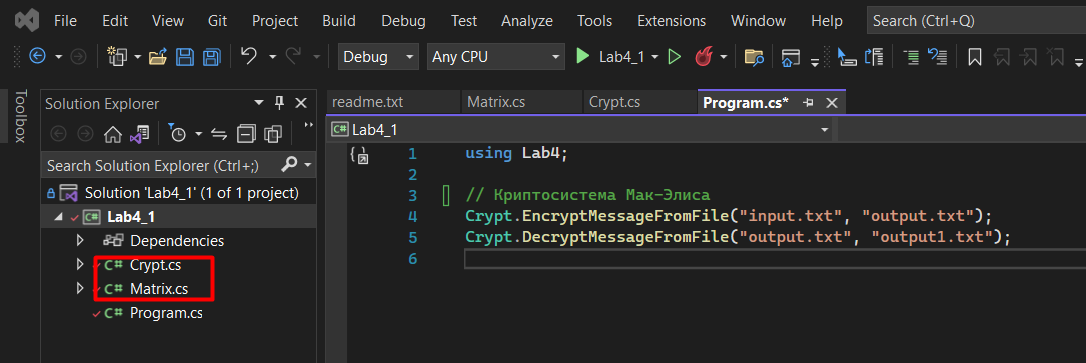


Рисунок 2.1 — Запускаемая программа

# ЗАКЛЮЧЕНИЕ

При выполнении лабораторной работы был изучен ассиметричный криптографический алгоритм Мак-Элиса, рассмотрены и составлены блок-схемы алгоритма, произведена его реализация на языке программирования C#, а также проверена работоспособность программного продукта.

Алгоритм не получил широкого признания, но в то же время является кандидатом для постквантовой криптографии, так как устойчив к атаке при помощи алгоритма Шора. Алгоритм основан на сложности декодирования полных линейных кодов. Открытый ключ получается при помощи маскировки выбранного кода как полного линейного.

Криптосистема имеет несколько преимуществ, например над RSA. Шифрование и дешифрование проходит быстрее и с ростом длины ключа степень защиты данных возрастает. Долгое время считалось, что криптосистема Мак-Элиса не используется для ЭЦП должным образом. Однако оказалось возможным построить схему для ЭЦП на основе криптосистемы Нидеррайтера (модификация криптосистемы Мак-Элиса).

Рассмотрим основные недостатки криптосистемы Мак-Элиса. Основной недостаток — это размер открытого ключа, он слишком большой. При использовании кодов Гоппы с параметрами, предложенными Мак-Элисом, открытый ключ составляет 219 бит, что вызывает сложности в реализации. Зашифрованное сообщение гораздо длиннее исходного. А также криптосистема не может быть использована для аутентификации, потому что схема шифрования не является взаимно-однозначной, а сам алгоритм является асимметричным.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Код программы**

**Crypt.cs**

using System.Collections;

using System.Numerics;

using static Lab4.Matrix;

using static System.Reflection.Metadata.BlobBuilder;

namespace Lab4 {

internal class Crypt {

private static readonly int m = 4;

private static readonly int n = 8;

private static readonly int t = 2;

private static int[][] matrixG = new int[][] {

new int[] { 1,0,0,0, 1,1,0 },

new int[] { 0,1,0,0, 1,0,1 },

new int[] { 0,0,1,0, 0,1,1 },

new int[] { 0,0,0,1, 1,1,1 },

};

private static int[][] matrixS = new int[][] {

new int[] { 1,0,0,0 },

new int[] { 0,1,0,0 },

new int[] { 0,0,1,0 },

new int[] { 0,0,0,1 }

};

private static int[][] matrixP = new int[][] {

new int[] { 1,0,0,0,0,0,0 },

new int[] { 0,1,0,0,0,0,0 },

new int[] { 0,0,1,0,0,0,0 },

new int[] { 0,0,0,1,0,0,0 },

new int[] { 0,0,0,0,1,0,0 },

new int[] { 0,0,0,0,0,1,0 },

new int[] { 0,0,0,0,0,0,1 },

};

private static int[][] matrixGHead;

private static int[][] matrixH = new int[][] {

new int[] { 1,1,0 },

new int[] { 1,0,1 },

new int[] { 0,1,1 },

new int[] { 1,1,1 },

new int[] { 1,0,0 },

new int[] { 0,1,0 },

new int[] { 0,0,1 },

};

private static int[][] matrixInverseP;

private static int[][] matrixInverseS;

private static int[] vectorZ;

private static void PrepareAllInfo() {

matrixGHead = MatrixProduct(matrixS, matrixG);

matrixGHead = MatrixProduct(matrixGHead, matrixP);

for (int i = 0; i < matrixGHead.Length; i++) {

for (int j = 0; j < matrixGHead[i].Length; j++) {

if (matrixGHead[i][j] % 2 == 0) {

matrixGHead[i][j] = 0;

}

}

}

matrixInverseP = MatrixInverse(matrixP);

matrixInverseS = MatrixInverse(matrixS);

}

private static byte[] CryptByte(byte byteForEncryption) {

BitArray bitArrayFull = new BitArray(new byte[] { byteForEncryption });

byte[] byteArr = new byte[(n-1)\*2];

int index = 0;

for (int k = 0; k < n; k += n/2) {

BitArray bitArrayHalf = new BitArray(4);

for (int j = 0; j < n/2; j++) {

bitArrayHalf[j] = bitArrayFull[j+k];

}

vectorZ = GetErrorVector(n-1, n/2);

int[] arr = new int[bitArrayHalf.Length];

for (int i = 0; i < bitArrayHalf.Length; i++) {

arr[i] = bitArrayHalf[i] ? 1 : 0;

}

int[] result = MultiplyMatrixOnVector(matrixGHead, arr);

for (int i = 0; i < result.Length; i++) {

result[i] += vectorZ[i];

result[i] %= 2;

}

for (int i = index; i < index + result.Length; i++) {

byteArr[i] = (byte)result[i-index];

}

index += (n - 1);

}

return byteArr;

}

private static byte DectyptBytes(byte[] inputBytes) {

byte[] bytes = new byte[2];

bool[] boolsResult = new bool[n];

int index = 0;

for (int k = 0; k < (n - 1) \* 2; k += (n - 1)) {

int[] arr = new int[inputBytes.Length / 2];

for (int i = k; i < k+inputBytes.Length/2; i++) {

arr[i-k] = (int)inputBytes[i];

}

int[] vectorCHead = MultiplyMatrixOnVector(matrixInverseP, arr);

for (int i = 0; i < vectorCHead.Length; i++) {

vectorCHead[i] %= 2;

}

int[] syndrom = MultiplyMatrixOnVector(matrixH, vectorCHead);

for (int i = 0; i < syndrom.Length; i++) {

syndrom[i] %= 2;

}

int resultIndex = -1;

for (int i = 0; i < matrixH.Length; i++) {

int sumIndex = 0;

for (int j = 0; j < matrixH[0].Length; j++) {

if (syndrom[j] == matrixH[i][j]) {

sumIndex++;

}

}

if (sumIndex == matrixH[0].Length) {

resultIndex = i;

break;

}

}

if (resultIndex != -1) {

vectorCHead[resultIndex] = vectorCHead[resultIndex] == 0 ? 1 : 0;

}

vectorCHead = MultiplyMatrixOnVector(matrixInverseS, vectorCHead);

for (int i = index; i < index+boolsResult.Length/2; i++) {

boolsResult[i] = vectorCHead[i-index] == 1 ? true : false;

}

index += (n / 2);

}

BitArray a = new BitArray(boolsResult);

a.CopyTo(bytes, 0);

return bytes[0];

}

public static void EncryptMessageFromFile(string inputFileName, string ouputFileName) {

PrepareAllInfo();

using (FileStream outputFile = File.OpenWrite(ouputFileName)) {

using (FileStream inputFile = File.OpenRead(inputFileName)) {

using (BinaryReader binaryReader = new BinaryReader(inputFile)) {

using (BinaryWriter binaryWriter = new BinaryWriter(outputFile)) {

while (binaryReader.BaseStream.Position != binaryReader.BaseStream.Length) {

byte[] encryptedBytes = CryptByte(binaryReader.ReadByte());

binaryWriter.Write(encryptedBytes);

}

}

}

}

}

}

public static void DecryptMessageFromFile(string inputFileName, string ouputFileName) {

using (FileStream outputFile = File.OpenWrite(ouputFileName)) {

using (FileStream inputFile = File.OpenRead(inputFileName)) {

using (BinaryReader binaryReader = new BinaryReader(inputFile)) {

using (BinaryWriter binaryWriter = new BinaryWriter(outputFile)) {

byte[] inputVal;

while ((inputVal = binaryReader.ReadBytes((n-1)\*2)).Length == (n-1)\*2) {

binaryWriter.Write(DectyptBytes(inputVal));

}

}

}

}

}

}

}

}

**Program.cs**

using Lab4;

// Криптосистема Мак-Элиса

Crypt.EncryptMessageFromFile("input.txt", "output.txt");

Crypt.DecryptMessageFromFile("output.txt", "output1.txt");