Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Методы защиты информации»

**ОТЧЕТ**

к лабораторной работе

на тему:

**«Асимметричная криптография. Алгоритм Мак-Элиса»**

|  |  |
| --- | --- |
|  | Выполнил:  Слуцкий Никита Сергеевич,  студент группы 053505 |
|  | Проверил: Лещенко Евгений Александрович, ассистент каф. Информатики |

Минск 2023

СОДЕРЖАНИЕ

[Введение 3](#_Toc148876153)

[1 Краткие теоретические сведения 4](#_Toc148876154)

[2 Ход выполнения работы 6](#_Toc148876155)

[Заключение 7](#_Toc148876156)

[Приложение А 8](#_Toc148876157)

# Введение

Целью данной лабораторной работы является реализовать программное средство шифрования и дешифрования текстовых файлов при помощи алгоритма Мак-Элиса на выбранном языке программирования.

Этот алгоритм основан на использовании кодовых матриц, и его эффективность зависит от размеров порождающей матрицы.

# 1 Краткие теоретические сведения

McEliece – криптосистема с открытыми ключами на основе теории алгебраического кодирования, разработанная в 1978 году Робертом Мак-Элисом. Это была первая схема, использующая рандомизацию в процессе шифрования. Алгоритм не получил широко признания в криптографии, но в то же время является кандидатом для постквантовой криптографии.

Алгоритм основан на сложности декодирования полных линейных кодов, потому что общая задача декодирования является NP-сложной.

На рисунке 1 представлена схема генерации ключа.



Рисунок 1 – Генерация ключа

Для построения ключа необходимо:

– выбрать порождающую матрицу G, зависящую от параметров n, k, t;

– выбрать случайную невырожденную матрицу M;

– выбрать случайную матрицу перестановок P;

– посчитать произведение G’ = M \* G \* P;

– пара (G’, t) будет являться открытым ключом;

– а тройка (M, G, P) хранится в секрете.

На рисунке 2 представлена схема шифрования.



Рисунок 2 – Схема шифрования

При шифровании необходимо:

– разбить сообщение на блоки по k бит;

– зашифровать при помощи формулы c = m \* G’ + e, где e ­– это вектор ошибок.

Для дешифрования нужно:

– вычислить значение x = c \* P­-1;

– произвести процесс декодирования, то есть поиска ближайшего кодового слова в пространстве векторов к получившемуся вектору;

– из полученного значения и его равенства a = m \* M \* G выразить m и получить текст.

# 2 Ход выполнения работы

Ввиду удобной работы с матрицами с использованием сторонних библиотек программное средство написано на языке программирования Python. В файлах encoder.py и decoder.py находятся классы для шифрования и дешифрования соответственно

В приложении с листингом кода предоставлена вырезка кода с функцией шифрования и дешифрования.

Реализованное программное средство имеет возможность считать файл, зашифровать или расшифровать его содержимое и сохранить преобразованный результат.

# Заключение

В ходе выполнения данной работы было произведено ознакомление с теорией об алгоритме шифрования Мак-Элиса, реализовано программное средство для шифрования и расшифрования текстовых файлов.

Из недостатков системы можно выделить:

– большой открытый ключ;

– зашифрованное сообщение сильно длиннее исходного.

Алгоритм Мак-Элиса с кодами Хэмминга предоставляет высокую стойкость к взлому, обеспечивая надежную защиту информации.

Цель лабораторной работы, поставленная во введении настоящего отчёта, можно считать достигнутой.

# Приложение А

(Листинг кода)

class Encoder:

"""Encodes a given message, m, using public key g\_prime, by permuting and then adding errors to the message."""

def \_\_init\_\_(self, m, g\_prime, t=1):

self.g\_prime = g\_prime

self.message = m

(k, n) = g\_prime.shape

self.k = k

self.n = n

self.t = t

self.z = self.generate\_errors()

self.encoded = self.encode()

def generate\_errors(self):

"""Generates 1 random errors in bitstring of length n"""

self.z = np.zeros(self.n)

idx\_list = np.random.choice(self.n, self.t, replace=False)

for idx in idx\_list:

self.z[idx] = 1

return self.z

def encode(self):

"""Encode message by multiplying by G^, and add random error."""

self.c\_prime = np.matmul(self.message, self.g\_prime) % 2

c = (self.c\_prime + self.z) % 2

return c

def get\_message(self):

return self.message

def get\_encrypted(self):

return self.encoded