Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчет по лабораторной работе №4

“Асимметричная криптография. Алгоритм Мак-Элиса”

|  |  |
| --- | --- |
|  | Выполнил:  студент группы 053504  Горожанкин В.О. |
|  | Проверил  ассистент кафедры информатики  Лещенко Евгений Александрович |

Минск 2023

**СОДЕРЖАНИЕ**

[Введение 3](#_heading=h.3ph5x23z8dty)

[1 Демонстрация работы программы 4](#_heading=h.8tumusvepw5e)

[2 Описание работы алгоритма 5](#_heading=h.30j0zll)

Заключение [7](#_heading=h.lqoqr01lz4qy)

Приложение [А](#_heading=h.k4rb95thohzo) [(обязательное)](#_heading=h.cxtr2jim0x3y) [Листинг программного кода](#_heading=h.qazk0foupweb) [8](#_heading=h.k4rb95thohzo)

# ВВЕДЕНИЕ

Криптография является неотъемлемой частью информационной безопасности в современном цифровом мире. Одним из ключевых аспектов криптографии является защита информации с помощью шифрования, которое позволяет передавать данные так, чтобы они были недоступны несанкционированным лицам. Асимметричная криптография представляет собой одну из наиболее важных и широко используемых техник шифрования, которая обеспечивает высокий уровень безопасности в обмене информацией.

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.

# 1 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

Входные данные записываются в файл input.txt. Содержимое файла представлено на рисунке 1.

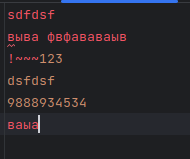


Рисунок 1 – Запись исходного текста в файл input.txt

В результате выполнения зашифрованный текст сохраняется в файл encrypted.txt. Содержимое файла:

11101100010100101111000000111001100001110010101000100010001011001101011011101101100000001000111000010111100100001000011111101100011011001010100001010101110001011100000010100011111011000101100001001000011000100011101110000001001111111010111000010010011110011110100000101011111110110001101111101010001000101011001111100001001010010000010000011100100000101001011110110101111100000010101110001000000101100100001110011111111000111110000100101011110000000101111010101010001110010011000110100000011010111111101100001001111001011111111010110110111101101001001101011010011111000110110000101010101001101011011111001011100101110000010011100011111000101011011001000001011011000101110110011000101011011001000101000011100101010000010000111011001110101111000010100110011000001011000111001000100001100010111010111000001010000101011010111000110010111100000100101110011101000010101010110010000001100100010110000100101000101111110000111100000010110110100000001111101010101000010111001100011100000100101101100010000000001111100101101011111010

В результате выполнения расшифрованный текст сохраняется в файл decrypted.txt. Содержимое файла представлено на рисунке 2.

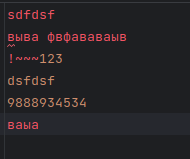


Рисунок 2 – Запись расшифрованного текста в файл decpypted.txt

# 2 ОПИСАНИЕ БЛОК-СХЕМЫ АЛГОРИТМА

Блок-схема алгоритма представлена на рисунке 3.



Рисунок 3 – Блок-схема алгоритма генерации ключей Мак-Элиса



Рисунок 4 – Блок-схема алгоритма шифрования сообщения

Генерация ключа

Принцип состоит в том, что Алиса выбирает линейный код из некоторого семейства кодов, для которого она знает эффективный алгоритм декодирования. Такой алгоритм декодирования требует не просто знания в смысле знания произвольной матрицы генератора, но требует знания параметров, используемых при указании в выбранном семействе кодов. Более конкретно, шаги следующие:

1. Алиса выбирает двоичный (n,k)-линейный код C способный (эффективно) исправлять t ошибки из некоторого большого семейства кодов. Этот выбор должен привести к эффективному алгоритму декодирования A. Пусть также G будет любой образующей матрицей для C. Любой линейный код имеет много образующих матриц, но часто есть естественный выбор для этого семейства кодов. Зная это, можно будет обнаружить A, поэтому его следует держать в секрете.
2. Алиса выбирает случайный k\*k двоичная невырожденная матрица S.
3. Алиса выбирает случайную n\*n матрица перестановок P.
4. Алиса вычисляет k\*n матрицу G=SGP.

Открытый ключ Алисы - (G, t); ее закрытый ключ: (S, P, A).

Шифрование сообщения

Предположим, Боб хочет отправить сообщение m Алисе, открытый ключ которой равен (G, t):

1. Боб кодирует сообщение m как двоичная строка длины k.
2. Боб вычисляет вектор c’=mG.
3. Боб генерирует случайный n-битовый вектор z, содержащий точно t единиц (вектор длины n и веса t)
4. Боб вычисляет зашифрованный текст как c=c’+z.

Расшифровка сообщения

После получения c, Алиса выполняет следующие шаги для расшифровки сообщения:

1. Алиса вычисляет обратное значение P (т.е. P-1).
2. Алиса вычисляет c=cP-1.
3. Алиса использует алгоритм декодирования A для декодирования c в m.
4. Алиса вычисляет m=mS-1.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы было реализовано программное средство шифрования и дешифрования текстовых файлов с использованием Криптосистемы Мак-Элиса. Этот процесс включал в себя несколько важных шагов, включая генерацию ключей, шифрование и последующую дешифрацию данных.

В итоге, выполнение данной лабораторной работы позволило нам приобрести практические навыки в области асимметричной криптографии и ознакомиться с принципами работы Криптосистемы Мак-Элиса.

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Листинг программного кода

import random

import numpy as np

H = np.array([[1, 0, 1, 0, 1, 0, 1],

[0, 1, 1, 0, 0, 1, 1],

[0, 0, 0, 1, 1, 1, 1]])

G = np.array([[1, 1, 0, 1],

[1, 0, 1, 1],

[1, 0, 0, 0],

[0, 1, 1, 1],

[0, 1, 0, 0],

[0, 0, 1, 0],

[0, 0, 0, 1]])

R = np.array([[0, 0, 1, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0],

[0, 0, 0, 0, 0, 1, 0],

[0, 0, 0, 0, 0, 0, 1]])

def random\_binary\_non\_singular\_matrix(n):

a = np.random.randint(0, 2, size=(n, n))

while np.linalg.det(a) == 0:

a = np.random.randint(0, 2, size=(n, n))

return a

S = random\_binary\_non\_singular\_matrix(4)

S\_inv = np.linalg.inv(S).astype(int)

def generate\_permutation\_matrix(n):

i = np.eye(n)

p = np.random.permutation(i)

return p.astype(int)

P = generate\_permutation\_matrix(7)

P\_inv = np.linalg.inv(P).astype(int)

G\_hat = np.transpose(np.mod((S.dot(np.transpose(G))).dot(P), 2))

# Определяет позицию ошибки в закодированных данных.

def detect\_error(err\_enc\_bits):

err\_idx\_vec = np.mod(H.dot(err\_enc\_bits), 2)

err\_idx\_vec = err\_idx\_vec[::-1]

err\_idx = int(''.join(str(bit) for bit in err\_idx\_vec), 2)

return err\_idx - 1

def hamming7\_4\_encode(p\_str):

p = np.array([int(x) for x in p\_str])

prod = np.mod(G\_hat.dot(p), 2)

return prod

def hamming7\_4\_decode(c):

prod = np.mod(R.dot(c), 2)

return prod

def flip\_bit(bits, n):

bits[n] = (bits[n] + 1) % 2

def add\_single\_bit\_error(enc\_bits):

error = [0] \* 7

idx = random.randint(0, 6)

error[idx] = 1

return np.mod(enc\_bits + error, 2)

def split\_binary\_string(str, n):

return [str[i:i + n] for i in range(0, len(str), n)]

def bits\_to\_str(bits):

# Split the binary string into 8-bit chunks

my\_chunks = [bits[i:i + 8] for i in range(0, len(bits), 8)]

# Convert each 8-bit chunk to its corresponding character

my\_chars = [chr(int(chunk, 2)) for chunk in my\_chunks]

# Concatenate the characters into a single string

my\_text = ''.join(my\_chars)

# Print the resulting text

return my\_text

if \_\_name\_\_ == '\_\_main\_\_':

with open("input.txt", "rb") as f:

text = f.read()

binary\_str = ''.join(format(x, '08b') for x in text)

# split bits into chunks of 4

split\_bits\_list = split\_binary\_string(binary\_str, 4)

enc\_msg = []

for split\_bits in split\_bits\_list:

enc\_bits = hamming7\_4\_encode(split\_bits)

# add a random bit error

err\_enc\_bits = add\_single\_bit\_error(enc\_bits)

# convert to string and append to result

str\_enc = ''.join(str(x) for x in err\_enc\_bits)

enc\_msg.append(str\_enc)

encoded = ''.join(enc\_msg)

with open("encrypt.txt", "w", encoding="utf-8") as f:

f.write(encoded)

dec\_msg = []

for enc\_bits in enc\_msg:

enc\_bits = np.array([int(x) for x in enc\_bits])

# compute c\_hat = c \* P\_inv

c\_hat = np.mod(enc\_bits.dot(P\_inv), 2)

# find the error bit

err\_idx = detect\_error(c\_hat)

# flip it

flip\_bit(c\_hat, err\_idx)

# find m\_hat

m\_hat = hamming7\_4\_decode(c\_hat)

# find m = m\_hat \* S\_inv

m\_out = np.mod(m\_hat.dot(S\_inv), 2)

str\_dec = ''.join(str(x) for x in m\_out)

dec\_msg.append(str\_dec)

dec\_msg\_str = ''.join(dec\_msg)

txt = bits\_to\_str(dec\_msg\_str)

with open("decoded.txt", "w", encoding="utf-8") as f:

f.write(text)