

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
1 Демонстрация работы программы.....	4
2 Описание работы алгоритма	5
Заключение	7
Приложение А (обязательное) Листинг программного кода	8

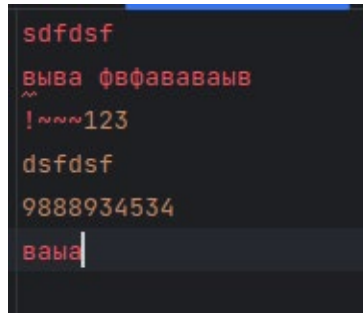
ВВЕДЕНИЕ

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.



```
sdfdsf
выва фвфававав
!~123
dsfdsf
9888934534
вава
```

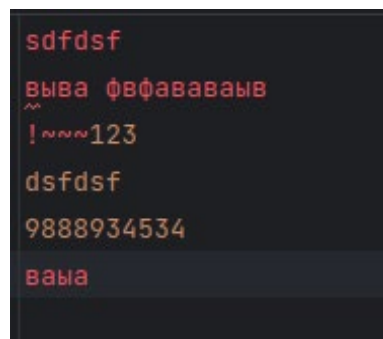
Рисунок 1 – Запись исходного текста в файл input.txt

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



```
111011000101001011110000001110011000011100101010001000100010110011010110111011011000000100011100
0010111100100001000011111101100011011001010100001010101110001011100000010100011111011000101100001
00100001100010001110111000000100111111010111000010010011110011110100000101011111101100011011111
010100010001010110011111000010010100100000100000111001000001010010111101101011111000000101011000
100000010110010000111001111111000111100001001010111100000001011110101010001110010011000110100
000011010111111101100001001111001011111110101101101111011010010011010110100111110001101100001010
1010100110101101111100101110010111000001001110001111100010101101100100000101101100010111011001100
0101011011001000101000011100101010000010000111011001110101111000010100110011000001011000111001000
10000110001011101011100000101000010101101011100011001011110000010010111001110100001010101100100
0000110010001011000010010100010111111000011110000001011011010000000111110101010100001011100110001
110000010010110110001000000000111110010110101111010
```

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



```
sdfdsf
выва фвфававав
!~123
dsfdsf
9888934534
вава
```

Рисунок 2 – Запись расшифрованного текста в файл decrypted.txt

2 ОПИСАНИЕ БЛОК-СХЕМЫ АЛГОРИТМА

Блок-схема алгоритма представлена на рисунке 3.



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

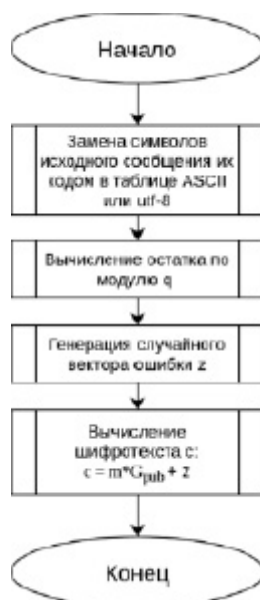


Рисунок 4 – Блок-схема алгоритма шифрования сообщения

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 $k \times k$ двоичная невырожденная матрица S .
3. Алиса выбирает случайную $n \times n$ матрица перестановок P .
4. Алиса вычисляет $k \times n$ 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  $\hat{c} = c * P_{inv}$ 
    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  $\hat{m}$ 
    m_hat = hamming7_4_decode(c_hat)
    # find  $m = \hat{m} * S_{inv}$ 
    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)

```