Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра автоматизированных систем управления (АСУ)

# СИММЕТРИЧНЫЕ АЛГОРИТМЫ ШИФРОВАНИЯ

Отчет по лабораторной работе №2

По дисциплине

«Информационная безопасность»

Студент гр. 431-3

_____ Е.П. Бекиш
(подпись)

_____
(дата)

Руководитель:
Ассистент кафедры АСУ

_____ Я.В. Яблонский
(подпись)

Томск 2024

Оглавление

1 Цель работы ..... 3
2 Задание на лабораторную работу ..... 4
3 Описание алгоритма шифрования ..... 5
4 Листинг программы ..... 8
5 Примеры работы программы ..... 11
6 Вывод ..... 13

# 1 Цель работы

Познакомиться и научиться работать с симметричными алгоритмами шифрования.

## 2 Задание на лабораторную работу

Задание по варианту №4: два друга хотят обмениваться зашифрованными сообщениями, но у них нет подходящей программы. Напишите программу позволяющую шифровать и дешифровать сообщения с использованием алгоритма симметричного шифрования RC4. Входные и выходные данные запишите в файл типа .txt.

# 3 Описание алгоритма шифрования

RC4, так же известен как ARC4 или ARCFOUR – потоковый шифр, широко применяющийся в различных системах защиты информации в компьютерных сетях.

Ядро алгоритма поточных шифров состоит из функции — генератора псевдослучайных битов (гаммы), который выдаёт поток битов ключа (ключевой поток, гамму, последовательность псевдослучайных битов).

Алгоритм шифрования.

1. Функции генерируют последовательность битов ($k_i$);
2. Затем последовательность битов посредством операции «суммирование по модулю два» (xor) объединяется с открытым текстом ($m_i$). В результате получается шифрограмма ($c_i$). $c_i = m_i \oplus k_i$.

Алгоритм расшифровки.

1. Повторно создается поток битов ключа ($k_i$);
2. Поток битов складывается с шифрограммой ($c_i$) операцией xor. В силу свойств операции xor на выходе получается исходный текст ($m_i$). $m_i = c_i \oplus k_i = (m_i \oplus k_i) \oplus k_i$.

Инициализация *S-блока.*

Алгоритм также известен как «key-scheduling algorithm» или «KSA». Этот алгоритм использует ключ, подаваемый на вход пользователем, сохранённый в Key, и имеющий длину L байт. Инициализация начинается с заполнения массива S, далее этот массив перемешивается путём перестановок, определяемых ключом. Так как только одно действие выполняется над S, то должно выполняться утверждение, что S всегда содержит один набор значений, который был дан при первоначальной инициализации (*S[i] := i*) как представлено на рисунке 3.1.

```
for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := ( j + S[i] + Key[ i mod L ] ) mod 256 // n = 8 ; 2^8 = 256
    поменять местами S[i] и S[j]
endfor
```

Рисунок 3.1 – Инициализация S-блока

Генерация псевдослучайного слова *K*

Эта часть алгоритма называется генератором псевдослучайной последовательности. Генератор ключевого потока RC4 переставляет значения, хранящиеся в S. В одном цикле RC4 определяется одно *n*-битное слово K из ключевого потока. В дальнейшем ключевое слово будет сложено по модулю два с исходным текстом, которое пользователь хочет зашифровать, и получен зашифрованный текст как показано на рисунках 3.2 и 3.3.

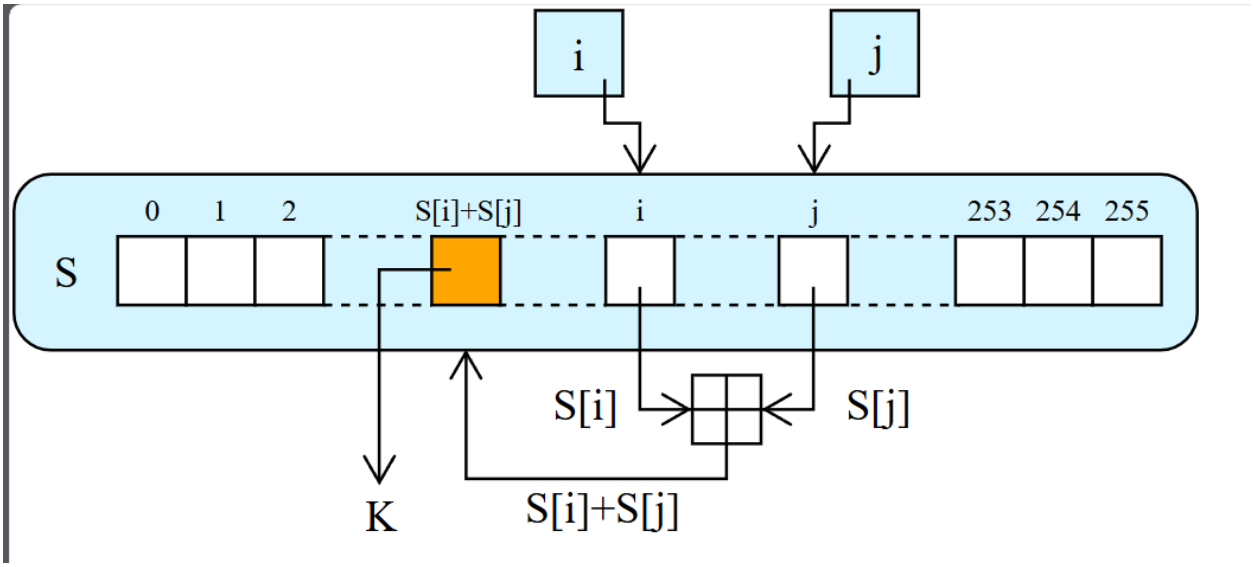


Рисунок 3.2 – Генерация псевдоключа слова K

```
i := 0
j := 0
while Цикл генерации:
    i := ( i + 1 ) mod 256
    j := ( j + S[i] ) mod 256
    поменять местами S[i] и S[j]
    t := ( S[i] + S[j] ) mod 256
    K := S[t]
    сгенерировано псевдослучайное слово K (для n = 8 будет сгенерирован один байт)
endwhile
```

Рисунок 3.3 – Алгоритм генерация псевдоключа слова K

# 4 Листинг программы

Листинг файла main.py

```
#Инициализация S-блока
def KSA(key: bytes) -> list[bytes]:
 s_block: list[int] = list(range(256))

 j: int = 0
 for i in range(256):
        j = (j + s_block[i] + key[i % len(key)]) % 256
        s_block[i], s_block[j] = s_block[j], s_block[i]

 return s_block

#Генерация псевдослучайного слова K
def PRGA(s_block: list[int]):
 i: int = 0
 j: int = 0
 while True:
        i = (i + 1) % 256
        j = (j + s_block[i]) % 256
        s_block[i], s_block[j] = s_block[j], s_block[i]
        k = s_block[(s_block[i] + s_block[j]) % 256]
        yield k

def RC4(s_block: list[bytes]) -> list[bytes]:
 key = PRGA(s_block)
 for _ in range(256):
        next(key)

 return key

def xor(text: bytes, key: bytes) -> list[bytes]:
```

```
result: list[bytes] = []
for symbol_byte in text:
        try:
                symbol_byte = ord(symbol_byte)
        except:
                pass

        result += [symbol_byte ^ next(key)]

return result



def get_byte_key() -> bytes:
return open('key.txt', 'rb').read()

def get_byte_text_from_file() -> bytes:
return open('file.txt', 'rb').read()



key_byte = get_byte_key()
key = KSA(key_byte)

text_byte = get_byte_text_from_file()

key_cipher = RC4(key[:])
cipher_byte = xor(text_byte, key_cipher)

open('cipher_text.txt', 'wb').write(bytes(cipher_byte))

key_decrypted = RC4(key[:])
decrypted_byte = xor(cipher_byte, key_decrypted)
```

```
open('decrypted_text.txt', 'wb').write(bytes(decrypted_byte))
```

## 5 Примеры работы программы

Далее подадим на вход в файл .txt два сообщения для шифрования и расшифрования. Результат работы программы можно увидеть на рисунках 5.1 – 5.7.

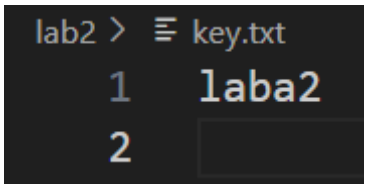


Рисунок 5.1 – Ключ шифрования

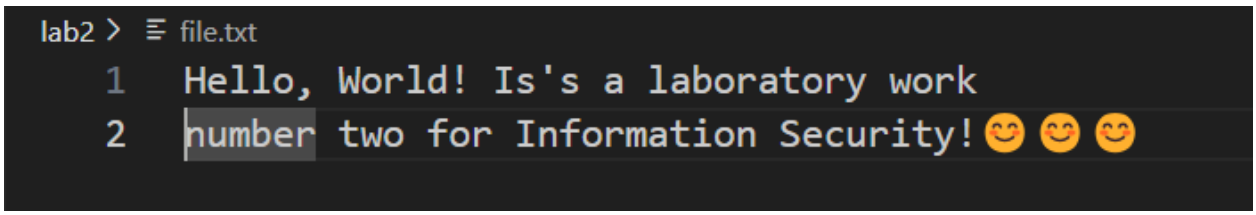


Рисунок 5.2 – Содержимое file.txt первого сообщения


Рисунок 5.3 – Содержимое cipher_text.txt первого сообщения

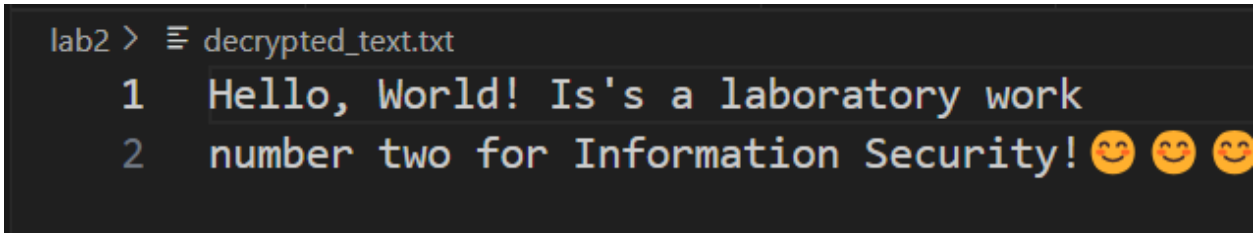


Рисунок 5.4 – Содержимое decrypted_text.txt первого сообщения

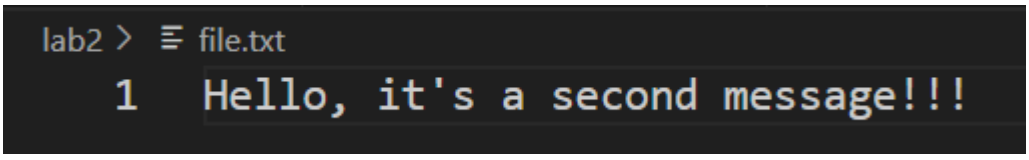


Рисунок 5.5 – Содержимое file.txt второго сообщения


Рисунок 5.6 – Содержимое cipher_text.txt второго сообщения

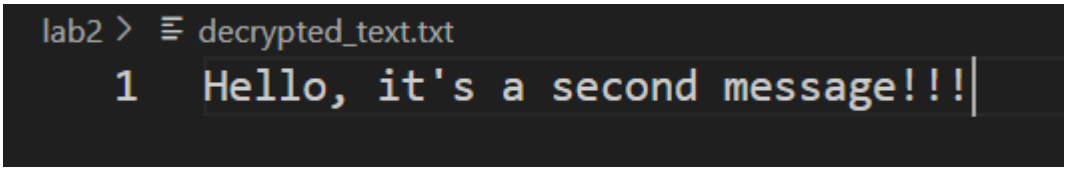


Рисунок 5.7 – Содержимое decrypted_text.txt второго сообщения

# 6 Вывод

В результате выполнения лабораторной работы я познакомился и научился работать с симметричными алгоритмами шифрования, а также реализовала алгоритм RC4.