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](#_Toc178426681)

[2 Задание на лабораторную работу 4](#_Toc178426682)

[3 Описание алгоритма шифрования 5](#_Toc178426683)

[4 Листинг программы 8](#_Toc178426684)

[5 Примеры работы программы 11](#_Toc178426685)

[6 Вывод 13](#_Toc178426686)

# Цель работы

Познакомиться и научиться работать с симметричными алгоритмами шифрования.

# 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.

# 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.

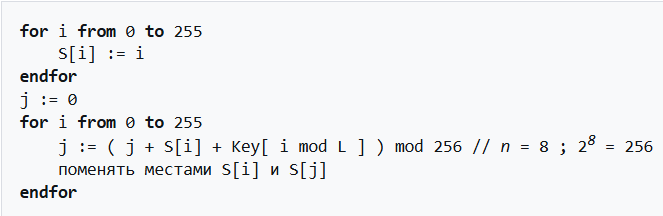
1. Функции генерируют последовательность битов ();
2. Затем последовательность битов посредством операции «суммирование по модулю два» (xor) объединяется с открытым текстом (). В результате получается шифрограмма (). .

Алгоритм расшифровки.

1. Повторно создается поток битов ключа ();
2. Поток битов складывается с шифрограммой () операцией xor. В силу свойств операции xor на выходе получается исходный текст (). .

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.

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.

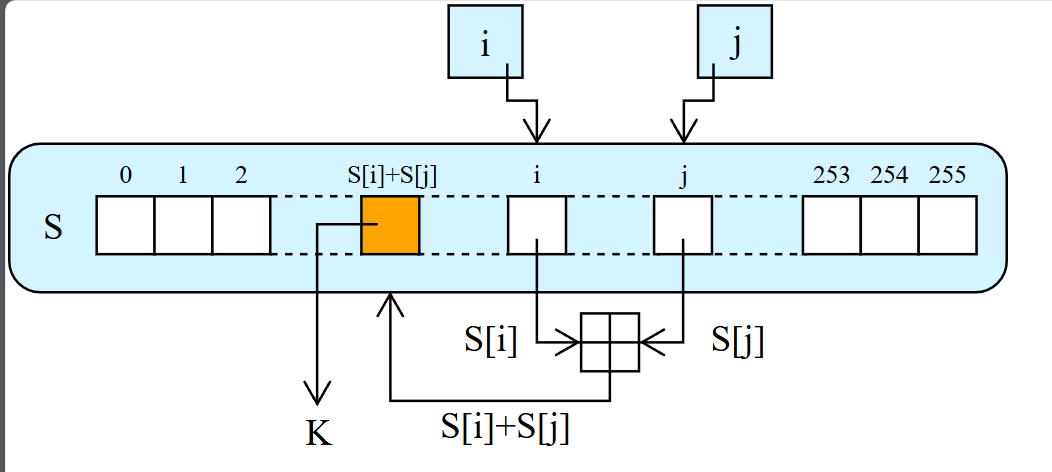
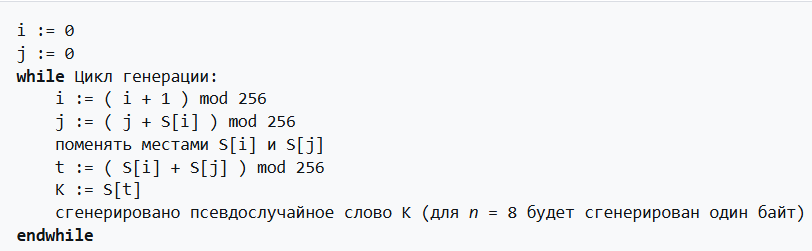
Рисунок 3.2 – Генерация псевдоключа слова K

Рисунок 3.3 – Алгоритм генерация псевдоключа слова K 

# 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)

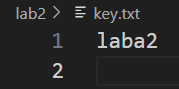
open('cipher\_text.txt', 'wb').write(bytes(cipher\_byte))

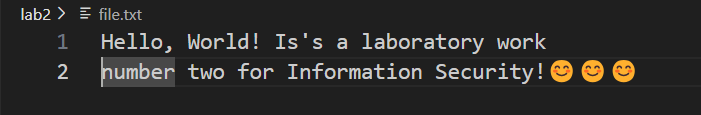
key\_decrypted = RC4(key[:])

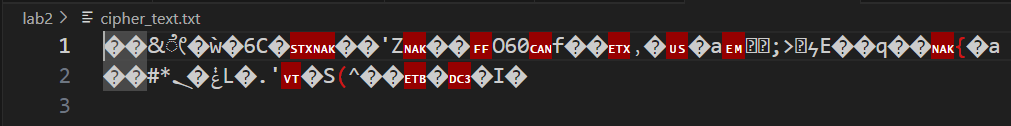
decrypted\_byte = xor(cipher\_byte, key\_decrypted)

open('decrypted\_text.txt', 'wb').write(bytes(decrypted\_byte))

# 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

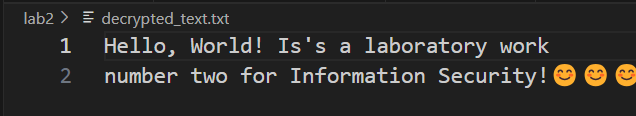
Рисунок 5.3 – Содержимое cipher\_text.txt первого сообщения

Рисунок 5.4 – Содержимое decrypted\_text.txt первого сообщения

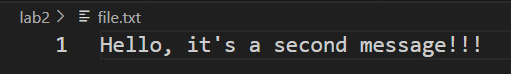
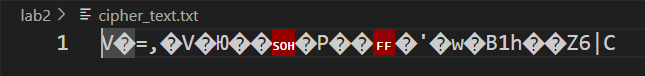
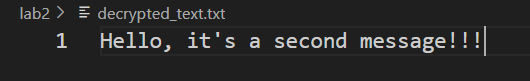
Рисунок 5.5 – Содержимое file.txt второго сообщения

Рисунок 5.6 – Содержимое cipher\_text.txt второго сообщения

Рисунок 5.7 – Содержимое decrypted\_text.txt второго сообщения

# Вывод

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