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 №2

на тему

**Симметричная криптография. СТБ**

**34.101.31-2011.**

Выполнил Д.С. Шевцова

Проверил Е.А. Лещенко

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc158758843)

[1 Краткие теоретические сведения 4](#_Toc158758844)

[2 Результаты выполнения лабораторной работы](#_Toc158758845) 7

[Заключение](#_Toc158758846) 8

[Приложение А](#_Toc158758848) [(обязательное)](#_Toc158758849) [Листинг кода](#_Toc158758850) 9

## ВВЕДЕНИЕ

Целью данной лабораторной работы является реализация программных средств шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ 34.101.31-2011 в режиме простой замены.

## 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

СТБ 34.101.31 – блочный шифр с 256-битным ключом и 8 циклами преобразований, который оперирует 128-битными блоками. Криптографические алгоритмы стандарта построены на основе базовых режимов шифрования блоков данных.

Алгоритмы шифрования, описанные в стандарте:

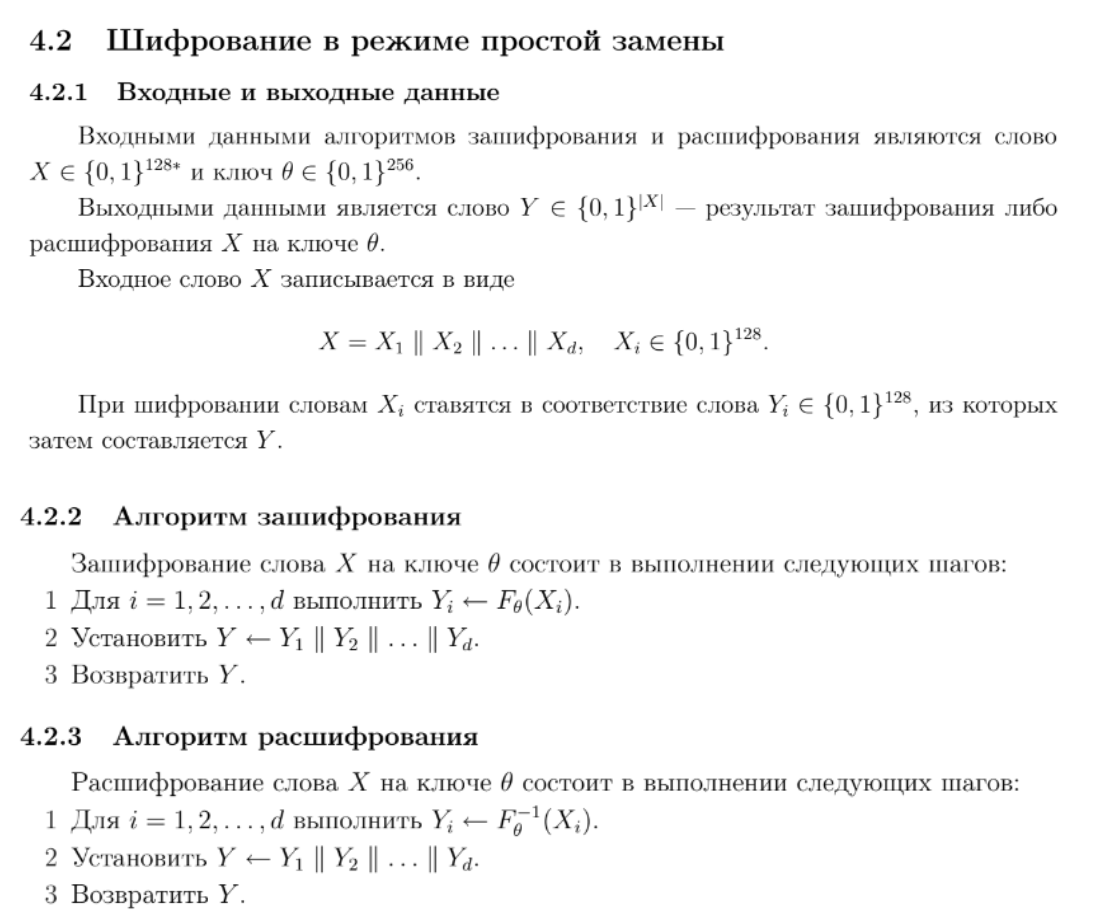
1 режим простой замены;

2 режим сцепления блоков;

3 режим гаммирования с обратной связью;

4 режим счетчика;

Каждая группа включает алгоритм зашифрования и алгоритм расшифрования с использованием секретного ключа. Предполагается, что стороны будут обмениваться сообщениями, используя один ключ, заранее распределен между ними. В режимах простой замены и сцепления блоков шифруются сообщения, которые содержат хотя бы один блок, а в режимах гаммирования с обратной связью и счётчика — сообщения произвольной длины.

****

# 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

На вход подается текстовый файл с содержимым для шифрации и для последующей расшифрации. На рисунке 2.1 представлен файл с текстом для шифрации, на рисунке 2.2 зашифрованный текст, а на рисунке 3.3 результат расшифрации.

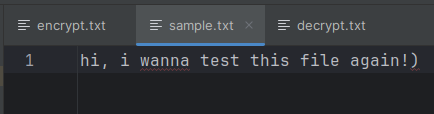


Рисунок 2.1 – Исходный текст

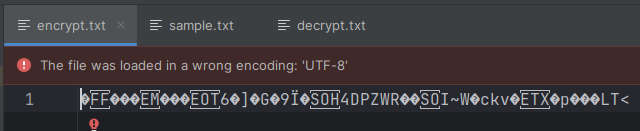


Рисунок 2.2 – Зашифрованный текст

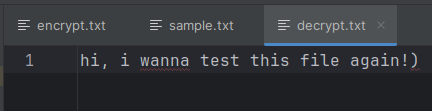


Рисунок 2.3 – Расшифрованный текст

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были реализованы программные средства шифрования и дешифрования текстовых файлов при помощи стандарта шифрования СТБ 34.101.31-2011 режиме простой замены. Также были изучены теоретические сведения в данной области.

# **ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

lab2.py

from constants import \*

class STB:

def \_\_init\_\_(self, key):

count = self.get\_key\_chunks\_counts(key)

self.tmp\_keys = []

for i in range(count):

self.tmp\_keys.append(key & 0xFFFF)

key >>= 32

if count == 4:

self.tmp\_keys.extend(self.tmp\_keys[:])

elif count == 6:

self.tmp\_keys.extend([

self.tmp\_keys[0] ^ self.tmp\_keys[1] ^ self.tmp\_keys[2],

self.tmp\_keys[3] ^ self.tmp\_keys[4] ^ self.tmp\_keys[5]

])

self.K = []

for \_ in range(8):

self.K.extend(self.tmp\_keys[:])

def get\_key\_chunks\_counts(self, key):

l = len(bin(key)[2:])

l &= (1 << 256) - 1

if 256 >= l > 192:

return 8

elif 192 >= l > 128:

return 6

elif l <= 128:

return 4

def rot\_hi(self, u):

if u < 1 << 31:

return (2 \* u) % (1 << 32)

else:

return (2 \* u + 1) % (1 << 32)

def rot\_hi\_r(self, u, r):

result = u

for \_ in range(r):

result = self.rot\_hi(result)

return result

def square\_plus(self, u, v):

return (u + v) % (1 << 32)

def square\_minus(self, u, v):

return (u - v) % (1 << 32)

def G(self, r, word):

mask = (1 << 8) - 1

final = 0

for i in range(4):

part = word & mask

word >>= 8

r = part & 0x0F

l = (part & 0xF0) >> 4

result = H[l][r]

result <<= 8 \* i

final += result

return self.rot\_hi\_r(final, r)

def encrypt\_block(self, X):

if self.get\_key\_chunks\_counts(X) != 4:

raise ValueError()

d = X & 0xFFFFFFFF

X >>= 32

c = X & 0xFFFFFFFF

X >>= 32

b = X & 0xFFFFFFFF

X >>= 32

a = X

for i in range(1, 9):

b = b ^ self.G(5, self.square\_plus(a, self.K[7\*i - 7]))

c = c ^ self.G(21, self.square\_plus(d, self.K[7\*i - 6]))

a = self.square\_minus(a, self.G(13, self.square\_plus(b, self.K[7\*i - 5])))

e = self.G(21, self.square\_plus(self.square\_plus(b, c), self.K[7\*i - 4])) ^ (i % (2 \*\* 32))

b = self.square\_plus(b, e)

c = self.square\_minus(c, e)

d = self.square\_plus(d, self.G(13, self.square\_plus(c, self.K[7\*i - 3])))

b = b ^ self.G(21, self.square\_plus(a, self.K[7\*i - 2]))

c = c ^ self.G(5, self.square\_plus(d, self.K[7\*i - 1]))

a, b = b, a

c, d = d, c

b, c = c, b

return (b << 96) + (d << 64) + (a << 32) + c

def decrypt\_block(self, X):

if self.get\_key\_chunks\_counts(X) != 4:

raise ValueError()

d = X & 0xFFFFFFFF

X >>= 32

c = X & 0xFFFFFFFF

X >>= 32

b = X & 0xFFFFFFFF

X >>= 32

a = X

for i in range(8, 0, -1):

b = b ^ self.G(5, self.square\_plus(a, self.K[7\*i - 1]))

c = c ^ self.G(21, self.square\_plus(d, self.K[7\*i - 2]))

a = self.square\_minus(a, self.G(13, self.square\_plus(b, self.K[7\*i - 3])))

e = self.G(21, self.square\_plus(self.square\_plus(b, c), self.K[7\*i - 4])) ^ (i % (2 \*\* 32))

b = self.square\_plus(b, e)

c = self.square\_minus(c, e)

d = self.square\_plus(d, self.G(13, self.square\_plus(c, self.K[7\*i - 5])))

b = b ^ self.G(21, self.square\_plus(a, self.K[7\*i - 6]))

c = c ^ self.G(5, self.square\_plus(d, self.K[7\*i - 7]))

a, b = b, a

c, d = d, c

a, d = d, a

return (c << 96) + (a << 64) + (d << 32) + b

def split\_message(self, message):

chunks = []

while message:

chunk = message & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

chunks.append(chunk)

message >>= 128

return chunks

def join\_message(self, chunks):

answer = 0

for chunk in chunks:

answer <<= 128

answer += chunk

return answer

def encrypt(self, message: str):

plain\_msg = int.from\_bytes(message.encode(), 'big')

chunks = self.split\_message(plain\_msg)

results = self.encrypt\_block\_plain(chunks)

answer = self.join\_message(results)

return answer.to\_bytes((answer.bit\_length() + 7) // 8, 'big')

def decrypt(self, message: bytes):

plain\_msg = int.from\_bytes(message, 'big')

chunks = reversed(self.split\_message(plain\_msg))

results = self.decrypt\_block\_plain(chunks)

answer = self.join\_message(reversed(results))

return answer.to\_bytes((answer.bit\_length() + 7) // 8, 'big').decode()

def encrypt\_block\_plain(self, chunks):

results = []

for X in chunks:

Y = self.encrypt\_block(X)

results.append(Y)

return results

def decrypt\_block\_plain(self, chunks):

results = []

for X in chunks:

Y = self.decrypt\_block(X)

results.append(Y)

return results

main.py

import sys

from constants import \*

from lab2 import STB

def main():

try:

# Открытие файла и чтение данных

with open("sample.txt", "r") as file:

data = file.read()

# Печать исходных данных

print("Data: %r\n" % data)

# Создание ключа и синхропункта

key = int.from\_bytes('erpwkwekrokpdasdasda'.encode(), 'big')

synchro = 312312412412

# Создание объекта STB и шифрование

s = STB(key)

encrypted\_text = s.encrypt(data)

# Сохранение зашифрованного сообщения в файл

with open("encrypt.txt", "wb") as file:

file.write(encrypted\_text)

print("Encrypted: %r" % encrypted\_text)

# Дешифрование из файла

with open("encrypt.txt", "rb") as file:

encrypted\_data = file.read()

decrypted\_text = s.decrypt(encrypted\_data)

# Сохранение расшифрованного сообщения в файл

with open("decrypt.txt", "w") as file:

file.write(decrypted\_text)

except Exception as e:

print(f"Ошибка: {e}", file=sys.stderr)

if \_\_name\_\_ == "\_\_main\_\_":

main()