Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ОТЧЕТ

по лабораторной работе

на тему

**Симметричная криптография. СТБ 34.101.31-2011**

|  | Выполнил студент группы 053501  Криштафович Карина Дмитриевна |
| --- | --- |
|  | Проверил  ассистент кафедры информатики  Лещенко Евгений Александрович |

Минск 2023

# СОДЕРЖАНИЕ

Введение3

[1 Демонстрация работы программы](#_heading=h.ct4fisxo76ob) 4

[1.1 Шифрование](#_heading=h.cgwqf0igtssc) 4

[1.2 Дешифрование](#_heading=h.azerj7ykjzed) 4

[2 Теоретические сведения](#_heading=h.gjdgxs) 5

Заключение8

Приложение [А](#_heading=h.fcdcaca7b490) [(обязательное)](#_heading=h.xcqdfzkvnwta) [Листинг программного кода](#_heading=h.drh0t9idhpqa)9

# ВВЕДЕНИЕ

Симметричная криптография — это фундаментальная область криптографии, которая занимается шифрованием и дешифрованием данных с использованием одного и того же секретного ключа. В данной лабораторной работе мы будем изучать стандарт криптографии СТБ 34.101.31-2011, который применяется в Республике Беларусь и определяет алгоритмы и принципы симметричного шифрования данных.

В ходе лабораторной работы мы рассмотрим принципы работы алгоритмов шифрования, их использование в защите данных, а также научимся реализовывать шифрование и дешифрование с использованием стандарта СТБ 34.101.31-2011. Эти навыки могут быть полезными в области информационной безопасности и защите конфиденциальных данных.

# 1 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

## **1.1 Шифрование**

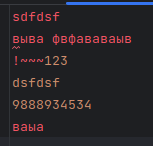


Рисунок 1 – Запись исходного текста в файл input.txt

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

251 240 162 88 153 32 168 142 194 112 19 165 168 61 84 162 171 45 112 48 215 89 108 145 45 237 35 193 161 63 192 144 99 190 117 237 151 93 121 182 67 251 223 128 247 17 69 176 54 92 129 90 216 144

## **1.2 Дешифрование**

В результате выполнения расшифрованный текст сохраняется в файл decpypted.txt, что показано на рисунке 2.

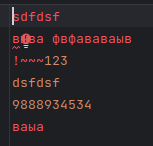


Рисунок 2 – Запись расшифрованного текста в файл decpypted.txt

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Блок-схема алгоритма на i-ом такте шифрования представлена на рисунке 3.

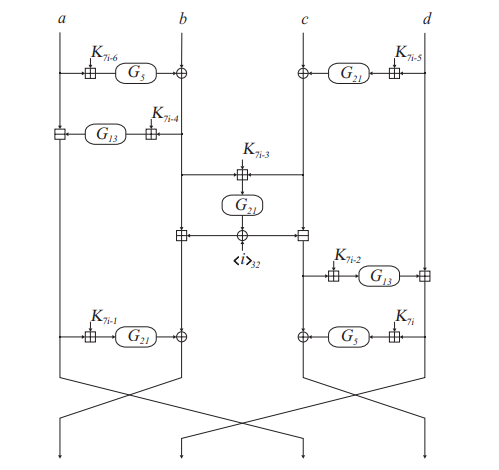


Рисунок 3 – Вычисления на i-м такте шифрования

.

Входными данными алгоритмов зашифрования и расшифрования являются блок  и ключ 

Выходными данными является блок — результат зашифрования либо расшифрования слова на ключе  либо {\displaystyle X\in \{0,1\}^{128}}

Входные данные для шифрования подготавливаются следующим образом:

* Слово  записывается в виде 
* Ключ записывается в виде  и определяются тактовые ключи

Обозначения и вспомогательные преобразования

Преобразование ставит в соответствие слову , слово



циклический сдвиг влево на r бит.

 операция замены 8-битной входной строки подстановкой с рисунка 4.

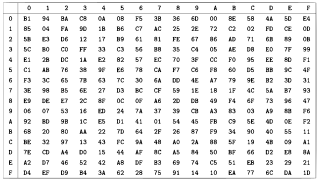


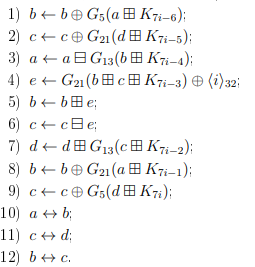
Рисунок 4 – Преобразование Н

Подстановка задается фиксированной таблицей. В таблице используется шестнадцатеричное представление слов 

 и  операции сложения и вычитания по модулю 232

Для зашифрования блока  на ключе  выполняются следующие шаги:

1. Установить 
2. Для i = 1,2,… ,8 выполнить:

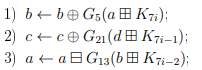


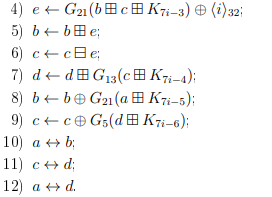
3. Установить 

4. Возвратить 

Для расшифрования блока  на ключе  выполняются следующие шаги:

1. Установить 
2. Для i = 8,7,… ,1 выполнить:





3. Установить  

4. Возвратить  

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы мы рассмотрели основные аспекты симметричной криптографии согласно стандарту СТБ 34.101.31-2011. Мы изучили основные принципы симметричных алгоритмов шифрования, их ключевые характеристики, а также роль симметрии в процессе обеспечения информационной безопасности.

Был проведен обзор стандарта СТБ 34.101.31-2011, включая его основные положения и требования. Мы изучили поддерживаемые алгоритмы шифрования и хэширования, а также требования к ключам и их генерации в соответствии с данным стандартом.

Понимание и применение стандарта СТБ 34.101.31-2011 в практике имеет важное значение для обеспечения безопасности информационных систем и данных. Этот стандарт предоставляет надежные инструменты для защиты конфиденциальной информации и обеспечения ее целостности.

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Листинг программного кода

import binascii

import random

class STB:

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

self.H = [0xB1, 0x94, 0xBA, 0xC8, 0x0A, 0x08, 0xF5, 0x3B, 0x36, 0x6D, 0x00, 0x8E, 0x58, 0x4A, 0x5D, 0xE4,

0x85, 0x04, 0xFA, 0x9D, 0x1B, 0xB6, 0xC7, 0xAC, 0x25, 0x2E, 0x72, 0xC2, 0x02, 0xFD, 0xCE, 0x0D,

0x5B, 0xE3, 0xD6, 0x12, 0x17, 0xB9, 0x61, 0x81, 0xFE, 0x67, 0x86, 0xAD, 0x71, 0x6B, 0x89, 0x0B,

0x5C, 0xB0, 0xC0, 0xFF, 0x33, 0xC3, 0x56, 0xB8, 0x35, 0xC4, 0x05, 0xAE, 0xD8, 0xE0, 0x7F, 0x99,

0xE1, 0x2B, 0xDC, 0x1A, 0xE2, 0x82, 0x57, 0xEC, 0x70, 0x3F, 0xCC, 0xF0, 0x95, 0xEE, 0x8D, 0xF1,

0xC1, 0xAB, 0x76, 0x38, 0x9F, 0xE6, 0x78, 0xCA, 0xF7, 0xC6, 0xF8, 0x60, 0xD5, 0xBB, 0x9C, 0x4F,

0xF3, 0x3C, 0x65, 0x7B, 0x63, 0x7C, 0x30, 0x6A, 0xDD, 0x4E, 0xA7, 0x79, 0x9E, 0xB2, 0x3D, 0x31,

0x3E, 0x98, 0xB5, 0x6E, 0x27, 0xD3, 0xBC, 0xCF, 0x59, 0x1E, 0x18, 0x1F, 0x4C, 0x5A, 0xB7, 0x93,

0xE9, 0xDE, 0xE7, 0x2C, 0x8F, 0x0C, 0x0F, 0xA6, 0x2D, 0xDB, 0x49, 0xF4, 0x6F, 0x73, 0x96, 0x47,

0x06, 0x07, 0x53, 0x16, 0xED, 0x24, 0x7A, 0x37, 0x39, 0xCB, 0xA3, 0x83, 0x03, 0xA9, 0x8B, 0xF6,

0x92, 0xBD, 0x9B, 0x1C, 0xE5, 0xD1, 0x41, 0x01, 0x54, 0x45, 0xFB, 0xC9, 0x5E, 0x4D, 0x0E, 0xF2,

0x68, 0x20, 0x80, 0xAA, 0x22, 0x7D, 0x64, 0x2F, 0x26, 0x87, 0xF9, 0x34, 0x90, 0x40, 0x55, 0x11,

0xBE, 0x32, 0x97, 0x13, 0x43, 0xFC, 0x9A, 0x48, 0xA0, 0x2A, 0x88, 0x5F, 0x19, 0x4B, 0x09, 0xA1,

0x7E, 0xCD, 0xA4, 0xD0, 0x15, 0x44, 0xAF, 0x8C, 0xA5, 0x84, 0x50, 0xBF, 0x66, 0xD2, 0xE8, 0x8A,

0xA2, 0xD7, 0x46, 0x52, 0x42, 0xA8, 0xDF, 0xB3, 0x69, 0x74, 0xC5, 0x51, 0xEB, 0x23, 0x29, 0x21,

0xD4, 0xEF, 0xD9, 0xB4, 0x3A, 0x62, 0x28, 0x75, 0x91, 0x14, 0x10, 0xEA, 0x77, 0x6C, 0xDA, 0x1D]

key = [self.list\_to\_int(key[i:i+4]) for i in range(0, len(key), 4)]

self.k = [key[i % 8] for i in range(56)]

# RotHi operation

def circularleftshift(self, value, k):

bitlength = 32

return (value << (k % bitlength) & 2 \*\* bitlength - 1) ^ (value >> (bitlength - k) % bitlength)

# Represent 32-bit number as list of bytes

def int\_to\_list(self, x):

return [x >> i & 0xff for i in [24, 16, 8, 0]]

# Represent list of bytes as 32-bit number

def list\_to\_int(self, x):

l = [24, 16, 8, 0]

return sum([x[i] << l[i] for i in range(4)])

def modsub(self, x, y):

mod = 2\*\*32

return (x - y) % mod

def modadd(self, \*x):

mod = 2\*\*32

res = 0

for el in x:

res = (res + self.reverse(el)) % mod

return self.reverse(res)

# H-transformation: replacing byte with another value from table

def htransformation(self, x):

return self.H[x]

# G-transformation

def gtransformation(self, x, k):

res = self.list\_to\_int([self.htransformation(i) for i in self.int\_to\_list(x)])

return self.reverse(self.circularleftshift(self.reverse(res), k))

def reverse(self, x):

l = self.int\_to\_list(x)

l.reverse()

return self.list\_to\_int(l)

def encryption(self, m):

a, b, c, d = [self.list\_to\_int(m[i:i+4]) for i in range(0, len(m), 4)]

for i in range(8):

b = b ^ self.gtransformation(self.modadd(a, self.k[7\*i+0]), 5)

c = c ^ self.gtransformation(self.modadd(d, self.k[7\*i+1]), 21)

a = self.reverse(self.modsub(self.reverse(a),

self.reverse(self.gtransformation(self.modadd(b, self.k[7\*i+2]), 13))))

e = (self.gtransformation(self.modadd(b, c, self.k[7\*i+3]), 21)) ^ self.reverse(i+1)

b = self.modadd(b, e)

c = self.reverse(self.modsub(self.reverse(c), self.reverse(e)))

d = self.modadd(d, self.gtransformation(self.modadd(c, self.k[7\*i+4]), 13))

b = b ^ self.gtransformation(self.modadd(a, self.k[7\*i+5]), 21)

c = c ^ self.gtransformation(self.modadd(d, self.k[7\*i+6]), 5)

a, b = b, a

c, d = d, c

b, c = c, b

a = self.int\_to\_list(a)

b = self.int\_to\_list(b)

c = self.int\_to\_list(c)

d = self.int\_to\_list(d)

return b + d + a + c

def decryption(self, m):

a, b, c, d = [self.list\_to\_int(m[i:i+4]) for i in range(0, len(m), 4)]

for i in reversed(range(8)):

b = b ^ self.gtransformation(self.modadd(a, self.k[7\*i+6]), 5)

c = c ^ self.gtransformation(self.modadd(d, self.k[7\*i+5]), 21)

a = self.reverse(self.modsub(self.reverse(a),

self.reverse(self.gtransformation(self.modadd(b, self.k[7\*i+4]), 13))))

e = (self.gtransformation(self.modadd(b, c, self.k[7\*i+3]), 21)) ^ self.reverse(i+1)

b = self.modadd(b, e)

c = self.reverse(self.modsub(self.reverse(c), self.reverse(e)))

d = self.modadd(d, self.gtransformation(self.modadd(c, self.k[7\*i+2]), 13))

b = b ^ self.gtransformation(self.modadd(a, self.k[7\*i+1]), 21)

c = c ^ self.gtransformation(self.modadd(d, self.k[7\*i+0]), 5)

a, b = b, a

c, d = d, c

a, d = d, a

a = self.int\_to\_list(a)

b = self.int\_to\_list(b)

c = self.int\_to\_list(c)

d = self.int\_to\_list(d)

return c + a + d + b

def CFB\_encrypt(self, msg):

vec\_init = 0x1234567891234567

msg[0] = msg[0] ^ vec\_init

msg[0] = self.encryption(msg[0])

for i in range(1, len(msg)):

msg[i] = msg[i] ^ msg[i - 1]

msg[i] = self.encryption(msg[i])

return msg

def CFB\_decrypt(self, msg):

vec\_init = 0x1234567891234567

temp\_one = msg[0]

msg[0] = self.decryption(msg[0])

msg[0] = msg[0] ^ vec\_init

for i in range(1, len(msg)):

temp\_two = msg[i]

msg[i] = self.decryption(msg[i])

msg[i] = msg[i] ^ temp\_one

temp\_one = temp\_two

return msg

def char\_to\_int(ch):

if ord(ch) < 140:

return ord(ch)

else:

return ord(ch) - 900

def int\_to\_char(i):

if i < 140:

return chr(i)

else:

return chr(i + 900)

def main():

with open("input.txt", "r", encoding='utf-8') as file:

text = file.read()

count = 0

while len(text) % 16:

text += '0'

count += 1

encrypted\_result = []

decrypted\_result = []

for i in range(len(text) // 16):

arr\_text = [char\_to\_int(item) for item in text[16 \* i: 16 \* (i + 1)]]

random\_bytes = bytes([random.randint(0, 255) for \_ in range(32)])

hex\_str = random\_bytes.hex()

key = list(binascii.unhexlify(hex\_str))

my\_stb = STB(key)

encrypted = my\_stb.CFB\_encrypt(arr\_text)

encrypted\_result.extend(encrypted)

decrypted\_result.extend(my\_stb.CFB\_decrypt(encrypted))

encrypted\_result = encrypted\_result[:len(encrypted\_result) - count]

decrypted\_result = decrypted\_result[:len(decrypted\_result) - count]

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

for item in encrypted\_result:

f.write(str(item) + ' ')

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

for item in decrypted\_result:

f.write(int\_to\_char(item))