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.**

**СТАНДАРТ ШИФРОВАНИЯ СТБ 34.101.31-2011**

Студент Д. С. Кончик

Преподаватель Е. А. Лещенко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc157722973)

[2 Краткие теоретические сведения 4](#_Toc157722974)

[3 Результаты выполнения лабораторной работы 5](#_Toc157722975)

[Выводы 6](#_Toc157722976)

[Список использованных источников 7](#_Toc157722977)

[Приложение А (обязательное) Листинг кода 8](#_Toc157722978)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является реализация программных средств шифрования и дешифрования текстовых файлов при помощи стандарта шифрования СТБ 34.101.31-2011 в режиме счетчика.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

BelT — государственный стандарт симметричного шифрования и контроля целостности Республики Беларусь. Полное название стандарта — СТБ 34.101.31-2007 «Информационные технологии и безопасность. Криптографические алгоритмы шифрования и контроля целостности». Принят в качестве предварительного стандарта в 2007 году. Введён в действие в качестве окончательного стандарта в 2011 году.

BelT — блочный шифр с 256-битным ключом и 8 циклами криптопреобразований, оперирующий с 128-битными словами. Криптографические алгоритмы стандарта построены на основе базовых режимов шифрования блоков данных. Все алгоритмы стандарта делятся на 8 групп:

– алгоритмы шифрования в режиме простой замены;

– алгоритмы шифрования в режиме сцепления блоков;

– алгоритмы шифрования в режиме гаммирования с обратной связью;

– алгоритмы шифрования в режиме счётчика;

– алгоритм выработки имитовставки ;

– алгоритмы одновременного шифрования и имитозащиты данных;

– алгоритмы одновременного шифрования и имитозащиты ключей;

– алгоритм хеширования;

Первые четыре группы предназначены для обеспечения безопасного обмена сообщениями. Каждая группа включает алгоритм шифрования и алгоритм расшифрования на секретном ключе.

Стороны, располагающие общим ключом, могут организовать обмен сообщениями путём их шифрования перед отправкой и расшифрования после получения.

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.

Пятый алгоритм предназначен для контроля целостности сообщений с помощью имитовставок — контрольных слов, которые определяются с использованием секретного ключа. Стороны, располагающие общим ключом, могут организовать контроль целостности при обмене сообщениями путём добавления к ним имитовставок при отправке и проверки имитовставок при получении. Проверка имитовставок дополнительно позволяет стороне получателю убедиться в знании стороной-отправителем секретного ключа, то есть проверить подлинность сообщений.

# **3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ             РАБОТЫ**

В ходе выполнения лабораторной было реализовано программное средство шифрования и дешифрования текстовых файлов при помощи стандарта шифрования СТБ 34.101.31-2011 в режиме счетчика.

Начальный текст находится в файле text.txt. После шифрования зашифрованная информация помещается в консоль, после чего она снова дешифруется и выводится. Вывод программы представлен на рисунке 1.

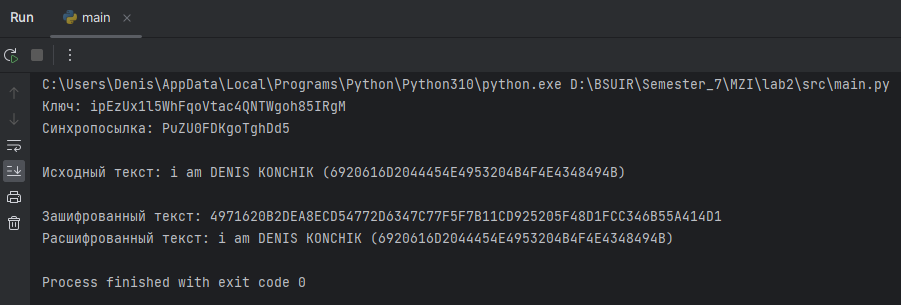


Рисунок 1 – Вывод программы

Таким образом, в ходе данной лабораторной работы было реализовано программное средство шифрования и дешифрования текстовых файлов при помощи стандарта шифрования СТБ 34.101.31-2011 в режиме счетчика**.**

# **ВЫВОДЫ**

В ходе данной лабораторной работы было реализовано программное средство шифрования и дешифрования текстовых файлов при помощи стандарта шифрования СТБ 34.101.31-2011 в режиме счетчика.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Алгоритм шифрования СТБ 34.101.31-2011 [Электронный ресурс]. – Режим доступа: https://apmi.bsu.by/assets/files/std/belt-spec27.pdf – Дата доступа: 15.09.2024.

[2] Режимы работы блочного шифра [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/otus/articles/812181/. – Дата доступа: 15.09.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

H\_TABLE = [  
 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  
]  
  
  
def hex\_to\_bin(hex\_string, size):  
 return bin(int(hex\_string, 16))[2:].zfill(size)  
  
  
def bin\_to\_hex(binary\_string):  
 number = int(binary\_string, 2)  
 width = (len(binary\_string) + 3) // 4  
  
 return f'{number:0{width}X}'  
  
  
def str\_to\_bin(text):  
 return ''.join(format(ord(char), 'b').zfill(8) for char in text)  
  
  
def str\_to\_hex(text):  
 return bin\_to\_hex(str\_to\_bin(text))  
  
  
def bin\_to\_str(binary\_string):  
 text = ''  
 for i in range(len(binary\_string) // 8):  
 bin\_number = binary\_string[i \* 8:(i + 1) \* 8]  
 number = int(bin\_number, 2)  
 text += chr(number)  
 return text  
  
  
def RotHi(u, r):  
 return u[r:] + u[:r]  
  
  
def H(u):  
 return '{0:b}'.format(H\_TABLE[int(u, 2)]).zfill(8)  
  
  
def G(u, r):  
 H\_chunks = []  
 for i in range(4):  
 H\_chunks.append(H(u[8 \* i:8 \* (i + 1)]))  
 H\_u = ''.join(H\_chunks)  
 return RotHi(H\_u, r)  
  
  
def U\_32(number):  
 return '{0:b}'.format(int(number % 2 \*\* 32)).zfill(32)  
  
  
def plus\_32(u, v):  
 return U\_32(int(u, 2) + int(v, 2))  
  
  
def minus\_32(u, v):  
 return U\_32(int(u, 2) - int(v, 2))  
  
  
def xor\_32(u, v):  
 return '{0:b}'.format(int(u, 2) ^ int(v, 2)).zfill(32)  
  
  
def xor(u, v):  
 return '{0:b}'.format(int(u, 2) ^ int(v, 2)).zfill(128)  
  
  
def U(number):  
 return '{0:b}'.format(int(number % 2 \*\* 128)).zfill(128)  
  
  
def plus(u, v):  
 return U(int(u, 2) + int(v, 2))  
  
  
def encrypt\_block(block, key):  
 X = list(chunks(block, 32))  
 theta = list(chunks(key, 32))  
 K = []  
 for i in range(56):  
 K.append(theta[i % len(theta)])  
  
 a, b, c, d = X[0], X[1], X[2], X[3]  
 for i in range(1, 9):  
 b = xor\_32(b, G(plus\_32(a, K[7 \* i - 7]), 5))  
 c = xor\_32(c, G(plus\_32(d, K[7 \* i - 6]), 21))  
 a = minus\_32(a, G(plus\_32(b, K[7 \* i - 5]), 13))  
 e = xor\_32(G(plus\_32(b, plus\_32(c, K[7 \* i - 4])), 21), U\_32(i))  
 b = plus\_32(b, e)  
 c = minus\_32(c, e)  
 d = plus\_32(d, G(plus\_32(c, K[7 \* i - 3]), 13))  
 b = xor\_32(b, G(plus\_32(a, K[7 \* i - 2]), 21))  
 c = xor\_32(c, G(plus\_32(d, K[7 \* i - 1]), 5))  
 a, b = b, a  
 c, d = d, c  
 b, c = c, b  
  
 return b + d + a + c  
  
  
def decrypt\_block(block, key):  
 X = list(chunks(block, 32))  
 theta = list(chunks(key, 32))  
 K = []  
 for i in range(56):  
 K.append(theta[i % len(theta)])  
  
 a, b, c, d = X[0], X[1], X[2], X[3]  
 for i in range(8, 0, -1):  
 b = xor\_32(b, G(plus\_32(a, K[7 \* i - 1]), 5))  
 c = xor\_32(c, G(plus\_32(d, K[7 \* i - 2]), 21))  
 a = minus\_32(a, G(plus\_32(b, K[7 \* i - 3]), 13))  
 e = xor\_32(G(plus\_32(b, plus\_32(c, K[7 \* i - 4])), 21), U\_32(i))  
 b = plus\_32(b, e)  
 c = minus\_32(c, e)  
 d = plus\_32(d, G(plus\_32(c, K[7 \* i - 5]), 13))  
 b = xor\_32(b, G(plus\_32(a, K[7 \* i - 6]), 21))  
 c = xor\_32(c, G(plus\_32(d, K[7 \* i - 7]), 5))  
 a, b = b, a  
 c, d = d, c  
 a, d = d, a  
  
 return c + a + d + b  
  
  
def chunks(string, size):  
 for i in range(0, len(string), size):  
 yield string[i:i + size]  
  
  
def get\_padding(text):  
 padding\_len = 16 - ((len(text) // 8) % 16)  
 return padding\_len \* '{0:b}'.format(padding\_len).zfill(8)  
  
  
def remove\_padding(text):  
 padding\_len = int(text[-8:], 2)  
 return text[:-padding\_len \* 8]  
  
  
def ctr\_encrypt(text, key, s):  
 text += get\_padding(text)  
 X = list(chunks(text, 128))  
  
 encrypted\_parts = []  
  
 counter = encrypt\_block(s, key)  
 for block in X:  
 counter = plus(counter, U(1))  
 encrypted\_parts.append(xor(block, encrypt\_block(counter, key)))  
  
 return ''.join(encrypted\_parts)  
  
  
def ctr\_decrypt(text, key, s):  
 X = list(chunks(text, 128))  
  
 decrypted\_parts = []  
  
 counter = encrypt\_block(s, key)  
 for block in X:  
 counter = plus(counter, U(1))  
 decrypted\_parts.append(xor(block, encrypt\_block(counter, key)))  
  
 return bin\_to\_str(remove\_padding(''.join(decrypted\_parts)))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with open('text.txt', 'r') as file:  
 t = file.read()  
  
 key = "ipEzUx1l5WhFqoVtac4QNTWgoh85IRgM"  
 print(f"Ключ: {key}")  
  
 S = "PuZU0FDKgoTghDd5"  
 print(f"Синхропосылка: {S}")  
 print()  
  
 print(f"Исходный текст: {t} ({str\_to\_hex(t)})")  
 print()  
  
 e\_ctr = ctr\_encrypt(str\_to\_bin(t), str\_to\_bin(key), str\_to\_bin(S))  
 print(f"Зашифрованный текст: {bin\_to\_hex(e\_ctr)}")  
  
 d\_ctr = ctr\_decrypt(e\_ctr, str\_to\_bin(key), str\_to\_bin(S))  
 print(f"Расшифрованный текст: {d\_ctr} ({str\_to\_hex(d\_ctr)})")