Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Лабораторная работа №2. Компьютерная реализация блочных шифров на примере СТБ П 34.101.31–2007

Выполнил: cтудент гр. 853501

Яковлев А.Б.

Проверил:

Протько М.И.

Минск 2021

# Постановка задачи и описание алгоритма

**Цель:** реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ 34.101.31-2011.

СТБ 34.101.31 – блочный шифр с 256-битным ключом и 8 циклами криптопреобразований, оперирующий с 128-битными блоками. Криптографические алгоритмы стандарта построены на основе базовых режимов шифрования блоков данных. Алгоритмы шифрования, описанные в стандарте:

* режим простой замены;
* режим сцепления блоков;
* режим гаммирования с обратной связью;
* режим счётчика;

Также в стандарте описаны:

* выработка имитовставки;
* одновременное шифрование и имитозащиты данных;
* одновременное шифрование и имитозащиты ключей;
* алгоритм хэширования;

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.

Рассмотрим алгоритм шифрования одного блока.

Входными данными алгоритмов зашифрования и расшифрования являются блок  и ключ 

Выходными данными является блок — результат зашифрования либо расшифрования слова на ключе  либо 

Входные данные для шифрования подготавливаются следующим образом:

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

Преобразование ставит в соответствие слову , слово



циклический сдвиг влево на r бит.  операция замены 8-битной входной строки подстановкой с рисунка 1.

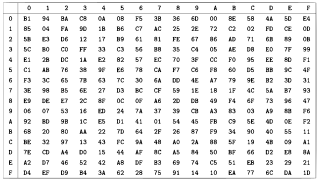


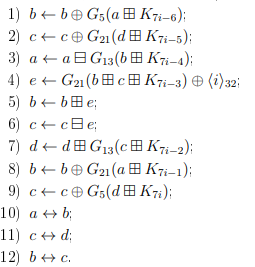
Рисунок 1 – Преобразование Н

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:

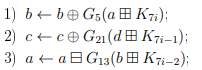


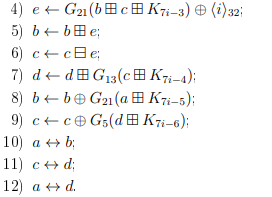
3. Установить 

4. Возвратить 

Для расшифрования блока  на ключе  выполняются следующие шаги:

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

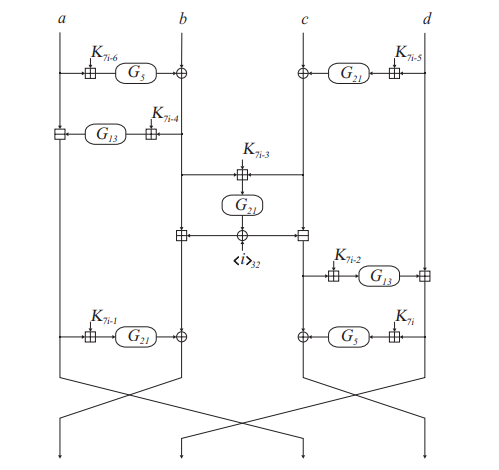




3. Установить  

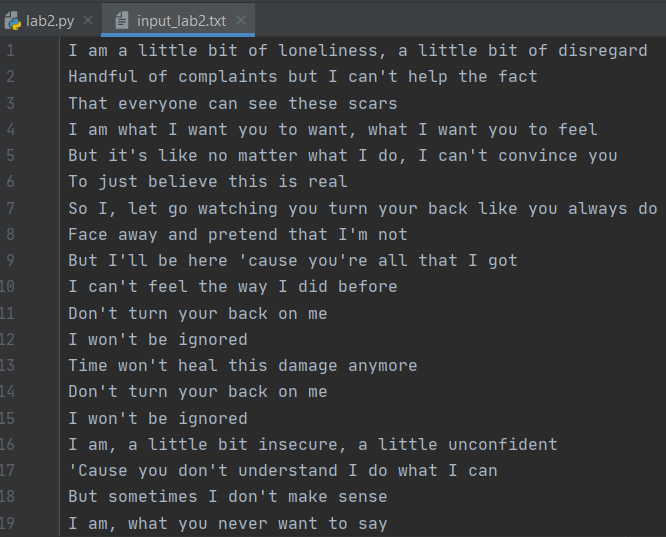
4. Возвратить  

# Блок-схемы алгоритмов

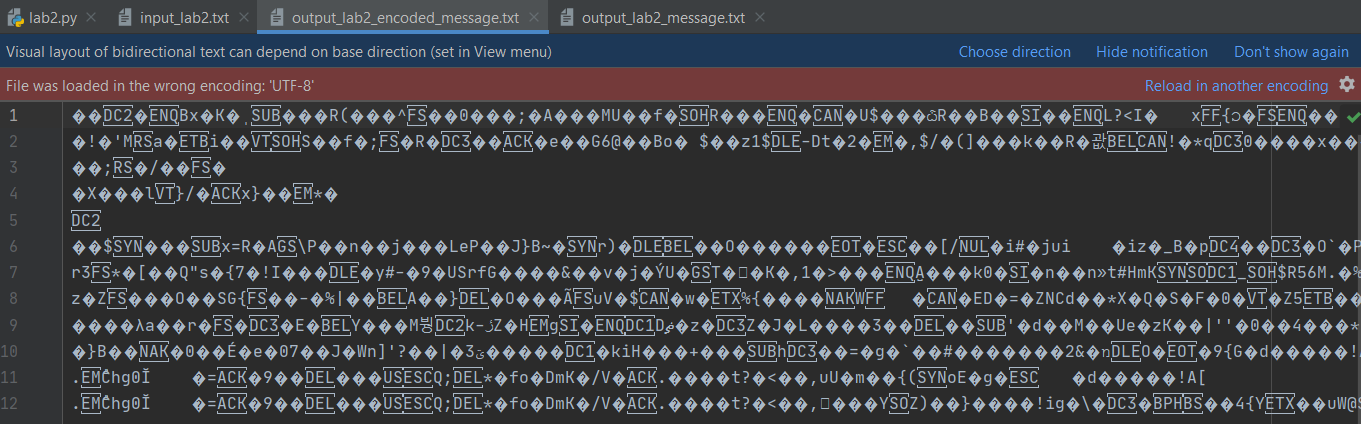


# Результаты выполнения программы

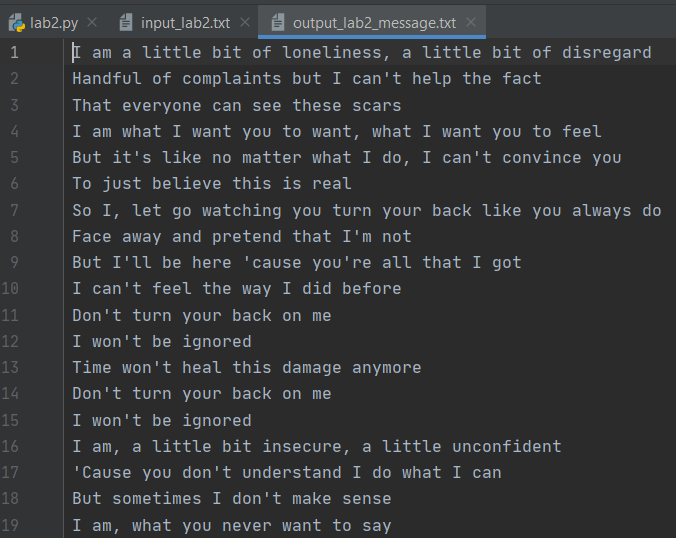
Файл с входными данными:



Зашифрованное сообщение:



Расшифрованное сообщение:



# Исходный код

import typing as tp  
  
from bitarray import bitarray  
from bitarray.util import ba2int, int2ba  
  
from block\_cipher import BlockCipher  
  
  
class Stb(BlockCipher, message\_block\_length=128):  
 \_substitution\_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 \_\_init\_\_(self, key: bitarray) -> None:  
 self.\_key = key  
  
 @staticmethod  
 def \_lambda\_function(block: bitarray) -> bitarray:  
 value = ba2int(block)  
 if value < (1 << 31):  
 return int2ba((2 \* value) % (1 << 32), length=32)  
 else:  
 return int2ba((2 \* value + 1) % (1 << 32), length=32)  
  
 @staticmethod  
 def \_lambda\_r\_function(block: bitarray, r: int) -> bitarray:  
 result = block.copy()  
 for \_ in range(r):  
 result = Stb.\_lambda\_function(result)  
 return result  
  
 @staticmethod  
 def \_apply\_substitution(substitution\_table: tp.List[tp.List[int]], block: bitarray) -> bitarray:  
 result\_block = bitarray()  
 for sub\_block\_no in range(4):  
 current\_sub\_block = block[sub\_block\_no \* 8: (sub\_block\_no + 1) \* 8]  
 substitution\_table\_row\_no = ba2int(current\_sub\_block[:4])  
 substitution\_table\_column\_no = ba2int(current\_sub\_block[4:])  
 result\_block.extend(  
 int2ba(substitution\_table[substitution\_table\_row\_no][substitution\_table\_column\_no], length=8))  
 return result\_block  
  
 @staticmethod  
 def \_g\_r\_function(substitution\_table: tp.List[tp.List[int]], block: bitarray, r: int) -> bitarray:  
 substitution\_result = Stb.\_apply\_substitution(substitution\_table, block)  
 lambda\_r\_function\_result = Stb.\_lambda\_r\_function(substitution\_result, r)  
 return lambda\_r\_function\_result  
  
 @staticmethod  
 def \_sum\_mod(first\_block: bitarray, second\_block: bitarray) -> bitarray:  
 return int2ba((ba2int(first\_block) + ba2int(second\_block)) % (1 << 32), length=32)  
  
 @staticmethod  
 def \_diff\_mod(first\_block: bitarray, second\_block: bitarray) -> bitarray:  
 return int2ba((ba2int(first\_block) - ba2int(second\_block)) % (1 << 32), length=32)  
  
 def \_get\_key\_block(self, tact\_key\_no: int) -> bitarray:  
 block\_no = tact\_key\_no % 8  
 return self.\_key[block\_no \* 32: (block\_no + 1) \* 32]  
  
 def \_encrypt\_block(self, message\_block: bitarray) -> bitarray:  
 a = message\_block[:32]  
 b = message\_block[32:64]  
 c = message\_block[64:96]  
 d = message\_block[96:]  
  
 for i in range(8):  
 b = b ^ self.\_g\_r\_function(self.\_substitution\_table, self.\_sum\_mod(a, self.\_get\_key\_block(7 \* i - 6)), 5)  
 c = c ^ self.\_g\_r\_function(self.\_substitution\_table, self.\_sum\_mod(d, self.\_get\_key\_block(7 \* i - 5)), 21)  
 a = self.\_diff\_mod(a, self.\_g\_r\_function(self.\_substitution\_table,  
 self.\_sum\_mod(b, self.\_get\_key\_block(7 \* i - 4)), 13))  
  
 e = self.\_g\_r\_function(self.\_substitution\_table,  
 self.\_sum\_mod(self.\_sum\_mod(b, c), self.\_get\_key\_block(7 \* i - 3)), 21) ^ int2ba(  
 i + 1, length=32)  
 b = self.\_sum\_mod(b, e)  
 c = self.\_diff\_mod(c, e)  
  
 d = self.\_sum\_mod(d, self.\_g\_r\_function(self.\_substitution\_table,  
 self.\_sum\_mod(c, self.\_get\_key\_block(7 \* i - 2)), 13))  
 b = b ^ self.\_g\_r\_function(self.\_substitution\_table, self.\_sum\_mod(a, self.\_get\_key\_block(7 \* i - 1)), 21)  
 c = c ^ self.\_g\_r\_function(self.\_substitution\_table, self.\_sum\_mod(d, self.\_get\_key\_block(7 \* i)), 5)  
  
 a, b = b, a  
 c, d = d, c  
 b, c = c, b  
 return b + d + a + c  
  
 def \_decrypt\_block(self, message\_block: bitarray) -> bitarray:  
 a = message\_block[:32]  
 b = message\_block[32:64]  
 c = message\_block[64:96]  
 d = message\_block[96:]  
  
 for i in range(7, -1, -1):  
 b = b ^ self.\_g\_r\_function(self.\_substitution\_table, self.\_sum\_mod(a, self.\_get\_key\_block(7 \* i)), 5)  
 c = c ^ self.\_g\_r\_function(self.\_substitution\_table, self.\_sum\_mod(d, self.\_get\_key\_block(7 \* i - 1)), 21)  
 a = self.\_diff\_mod(a, self.\_g\_r\_function(self.\_substitution\_table,  
 self.\_sum\_mod(b, self.\_get\_key\_block(7 \* i - 2)), 13))  
  
 e = self.\_g\_r\_function(self.\_substitution\_table,  
 self.\_sum\_mod(self.\_sum\_mod(b, c), self.\_get\_key\_block(7 \* i - 3)), 21) ^ int2ba(  
 i + 1, length=32)  
 b = self.\_sum\_mod(b, e)  
 c = self.\_diff\_mod(c, e)  
  
 d = self.\_sum\_mod(d, self.\_g\_r\_function(self.\_substitution\_table,  
 self.\_sum\_mod(c, self.\_get\_key\_block(7 \* i - 4)), 13))  
 b = b ^ self.\_g\_r\_function(self.\_substitution\_table, self.\_sum\_mod(a, self.\_get\_key\_block(7 \* i - 5)), 21)  
 c = c ^ self.\_g\_r\_function(self.\_substitution\_table, self.\_sum\_mod(d, self.\_get\_key\_block(7 \* i - 6)), 5)  
  
 a, b = b, a  
 c, d = d, c  
 a, d = d, a  
 return c + a + d + b  
  
  
def main() -> None:  
 key = bitarray(256)  
 encoder = Stb(key)  
 with open('input\_files/input\_lab2.txt', 'r') as f:  
 message = bitarray()  
 message.frombytes(f.read().encode('UTF-8'))  
 encoded\_message = encoder.encrypt(message)  
 with open('output\_files/output\_lab2\_encoded\_message.txt', 'wb') as f:  
 encoded\_message.tofile(f)  
 decoded\_message = encoder.decrypt(encoded\_message)  
 with open('output\_files/output\_lab2\_message.txt', 'wb') as f:  
 decoded\_message.tofile(f)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Вывод

В результате выполнения работы была реализована программа, позволяющая шифровать и расшифровывать сообщения при помощи шифра СТБ 34.101.31.