**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЁТ**

**по лабораторной работе №2**

**«Основы блочного шифрования»**

*дисциплина: Математические основы защиты информации и информационной безопасности*

Вариант 8

Студент: Хиссен Али Уэддей

Группы: НПМмд-02-20

Ст/б: 10322009306

**МОСКВА**

**2021**

Оглавление

[Цель работы 3](#_Toc57465413)

[Описание алгоритма 3](#_Toc57465414)

[Реализация алгоритма (код программы) 4](#_Toc57465415)

[Результаты работы программы с различными исходными текстами, ключами 9](#_Toc57465416)

[Вывод 10](#_Toc57465417)

[Ответы на контрольные вопросы 10](#_Toc57465418)

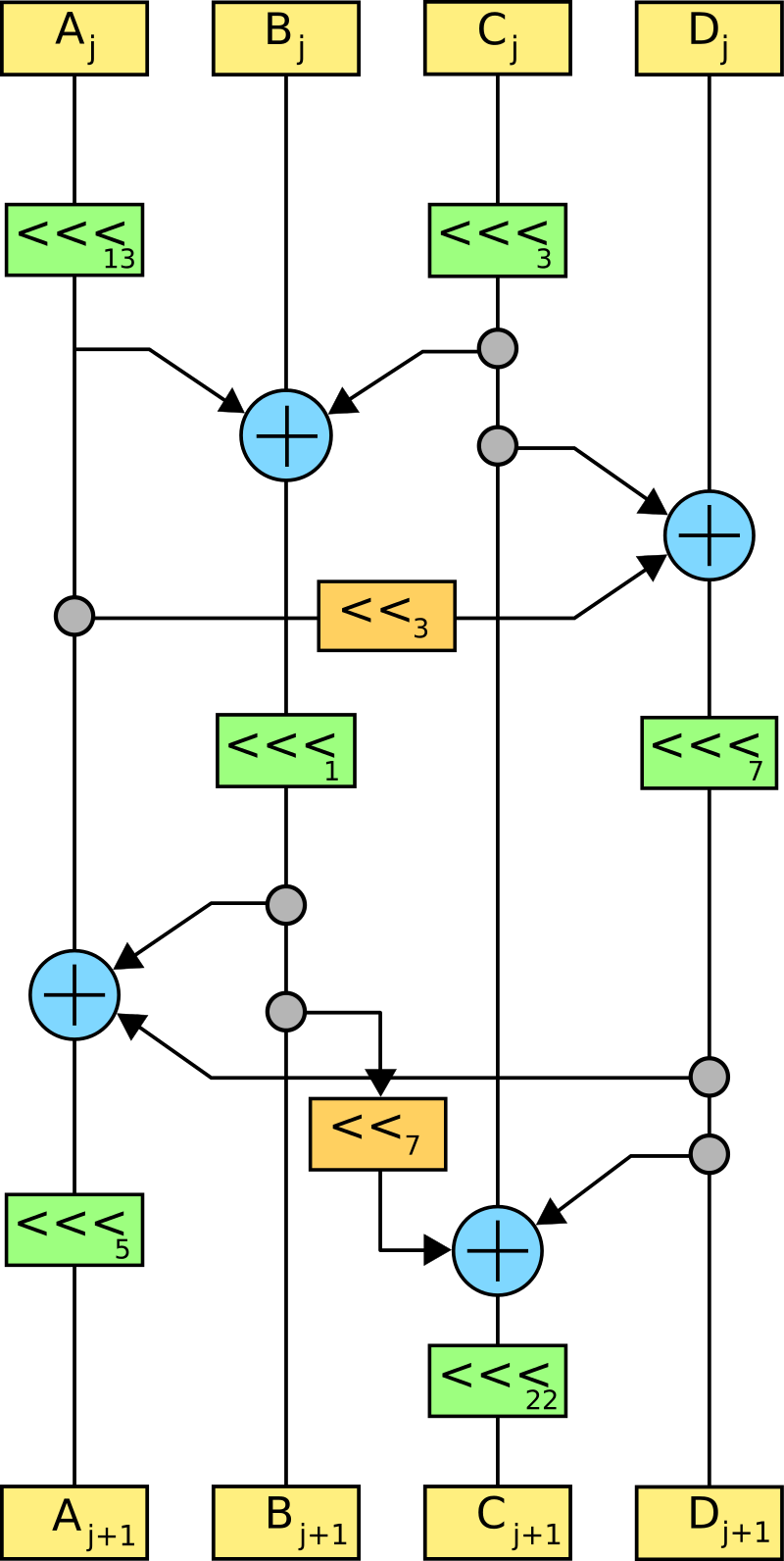
# **Цель работы**

Целью работы является изучение симметричных алгоритмов шифрования, основанных на блочных шифрах, а также реализация алгоритма шифрования/дешифрования Serpent.

Определила вариант с помощью формулы , где — номер студ.билета, N— количество заданий. Так как Sn=1032209306, а N=9, то номер варианта лабораторной работы 8.

# **Описание алгоритма**

Serpent имеет размер блока 128 бит и возможные длины ключа 128, 192 или 256 бит. Алгоритм представляет собой 32-раундовую сеть Файстеля, работающую с блоком из четырёх 32-битных слов. Serpent был разработан так, что все операции могут быть выполнены параллельно, используя 32 1-битных «потока».



Индексы бит пробегают значения от 0 до 31 для 32-битных слов, от 0 до 127 для 128-битных блоков, от 0 до 255 для 256-битных ключей и так далее. Для внутренних вычислений все биты величин представлены в прямом порядке.

Serpent шифрует открытый текст длиной 128 бит в шифротекст длиной так же 128 бит за 32 раунда с помощью 33 подключей длиной 128 бит. Длина используемого ключа может принимать различные значения, 128, 192 или 256 бит. Шифрование состоит из следующих основных шагов. Во-первых, начальная перестановка. Во-вторых, 32 раунда, каждый из которых состоит из операции смешивания с 128-битным ключом (побитовое логическое исключающее «или»), табличная замена (S-box) и линейное преобразование. В последнем раунде линейное преобразование заменяется дополнительным наложением ключа. Последний этап - конечная перестановка.

Начальная и конечная перестановки не имеют какой-либо криптографической значимости. Они используются для упрощения оптимизированной реализации алгоритма и повышения вычислительной эффективности.

# **Реализация алгоритма (код программы)**

class Serpent:

def \_\_init\_\_(self, key=None):

"""Serpent."""

if key:

self.set\_key(key)

def set\_key(self, key):

"""Init."""

key\_len = len(key)

if key\_len % 4:

# XXX: add padding?

raise KeyError, "key not a multiple of 4"

if key\_len > 32:

# XXX: prune?

raise KeyError, "key\_len > 32"

self.key\_context = [0] \* 140

key\_word32 = [0] \* 32

i = 0

while key:

key\_word32[i] = struct.unpack("<L", key[0:4])[0]

key = key[4:]

i += 1

set\_key(self.key\_context, key\_word32, key\_len)

#print(map(hex,self.key\_context))

import struct

import sys

WORD\_BIGENDIAN = 0

if sys.byteorder == 'big':

WORD\_BIGENDIAN = 1

def rotr32(x, n):

return (x >> n) | ((x << (32 - n)) & 0xFFFFFFFF)

def rotl32(x, n):

return ((x << n) & 0xFFFFFFFF) | (x >> (32 - n))

def byteswap32(x):

return ((x & 0xff) << 24) | (((x >> 8) & 0xff) << 16) | \

(((x >> 16) & 0xff) << 8) | ((x >> 24) & 0xff)

def set\_key(l\_key, key, key\_len):

key\_len \*= 8

if key\_len > 256:

return False

i = 0

lk = (key\_len + 31) / 32

while i < lk:

l\_key[i] = key[i]

if WORD\_BIGENDIAN:

l\_key[i] = byteswap32(key[i])

i += 1

if key\_len < 256:

while i < 8:

l\_key[i] = 0

i += 1

i = key\_len / 32

lk = 1 << (key\_len % 32)

l\_key[i] = (l\_key[i] & (lk - 1)) | lk

for i in xrange(132):

lk = l\_key[i] ^ l\_key[i + 3] ^ l\_key[i + 5] ^ l\_key[i + 7] ^ 0x9e3779b9 ^ i

l\_key[i + 8] = ((lk << 11) & 0xFFFFFFFF) | (lk >> 21)

key = l\_key

t1 = (~a) % 0x100000000;

t2 = a ^ d;

t3 = b ^ t2;

t4 = t1 | t2;

t5 = c ^ t4;

f = b ^ t5;

t13 = (~t5) % 0x100000000;

t7 = t2 | f;

t8 = d ^ t7;

t9 = t5 & t8;

g = t3 ^ t9;

t11 = t5 ^ t8;

e = g ^ t11;

t14 = t3 & t11;

h = t13 ^ t14

key[4 \* 5 + 8] = e

key[4 \* 5 + 9] = f

key[4 \* 5 + 10] = g

key[4 \* 5 + 11] = h

a = key[4 \* 6 + 8]

b = key[4 \* 6 + 9]

c = key[4 \* 6 + 10]

d = key[4 \* 6 + 11]

t1 = (~a) % 0x100000000;

t2 = a ^ b;

t3 = a ^ d;

t4 = c ^ t1;

t5 = t2 | t3;

e = t4 ^ t5;

t7 = d & e;

t8 = t2 ^ e;

t10 = t1 | e;

f = t7 ^ t8;

t11 = t2 | t7;

t12 = t3 ^ t10;

t14 = b ^ t7;

g = t11 ^ t12;

t15 = f & t12;

h = t14 ^ t15

key[4 \* 6 + 8] = e

key[4 \* 6 + 9] = f

key[4 \* 6 + 10] = g

key[4 \* 6 + 11] = h

a = key[4 \* 7 + 8]

b = key[4 \* 7 + 9]

c = key[4 \* 7 + 10]

|  |  |
| --- | --- |
|  | 'ADD': lambda x, y: (x + y) % 2 \*\* 256, |
|  | 'MUL': lambda x, y: (x \* y) % 2 \*\* 256, |
|  | 'SUB': lambda x, y: (x - y) % 2 \*\* 256, |
|  | 'DIV': lambda x, y: x / y, |
|  | 'EXP': lambda x, y: pow(x, y, 2 \*\* 256), |
|  | 'AND': lambda x, y: x & y, |
|  | 'OR': lambda x, y: x | y, |
|  | 'XOR': lambda x, y: x ^ y |
|  | } |

def serpent\_cbc\_encrypt(key, data, iv='\x00'\*16):

out = ""

last = iv

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

temp = data[i\*16:(i+1)\*16]

to\_encode = ""

for j in range(4):

temp1 = struct.unpack\_from('<I', temp[j\*4:])[0]

temp2 = struct.unpack\_from('<I', last[j\*4:])[0]

to\_encode += struct.pack('<I',((temp1 ^ temp2) & 0xffffffff))

last= Serpent(key).encrypt(to\_encode)

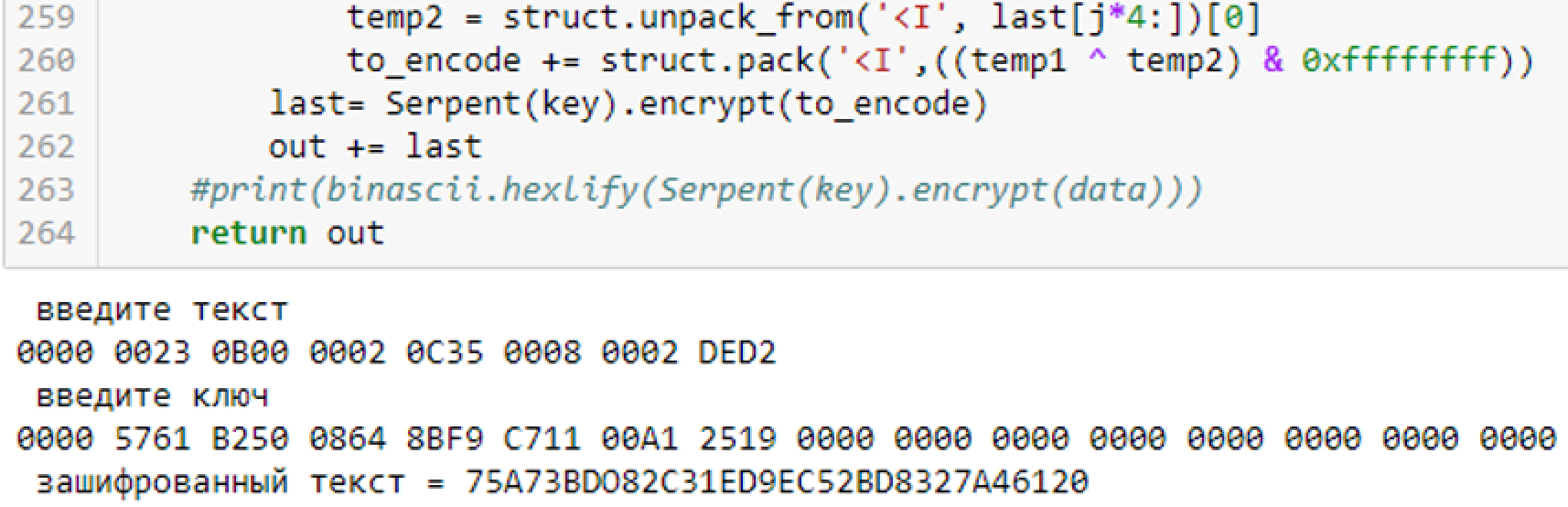
out += last

#print(binascii.hexlify(Serpent(key).encrypt(data)))

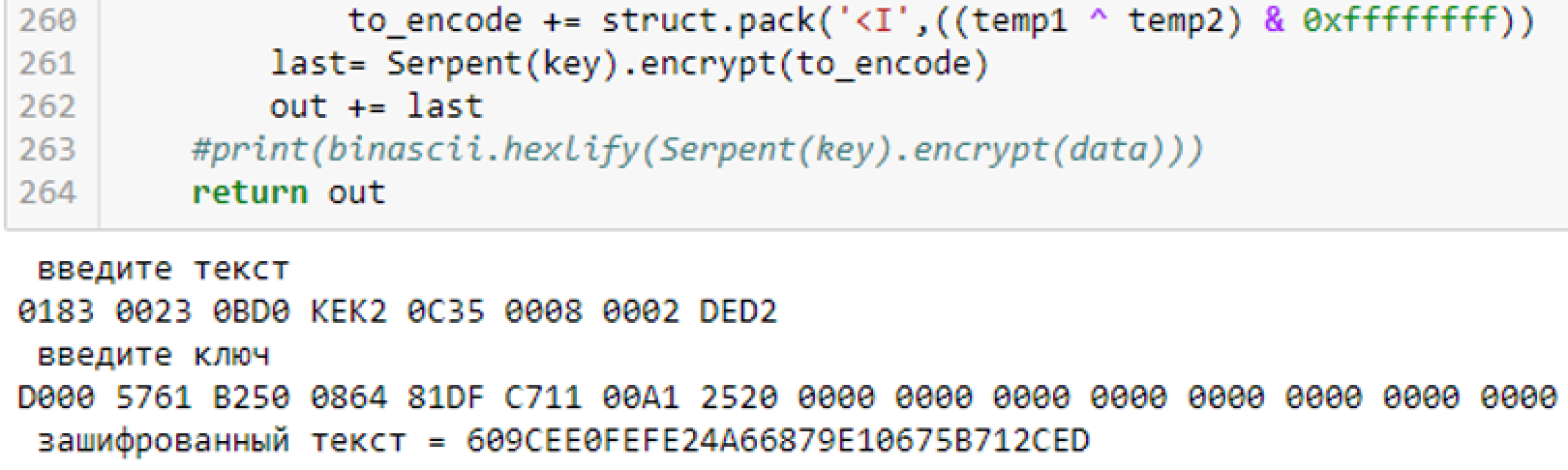
return out

# **Результаты работы программы с различными исходными текстами, ключами**

Пример 1. Зашифруем, а затем дешифруем текст “0000 0023 0B00 0002 0C35 0008 0002 DED2”. Используем ключ “0000 5761 B250 0864 8BF9 C711 00A1 2519 0000 0000 0000 0000 0000 0000 0000 0000”



Пример 2*.* Зашифруем, а затем дешифруем текст “0183 0023 0BD0 KEK2 0C35 0008 0002 DED2”. Используем ключ “D000 5761 B250 0864 81DF C711 00A1 2520 0000 0000 0000 0000 0000 0000 0000 0000”



# **Вывод**

В ходе выполнения лабораторной работы были изучены симметричные алгоритмы шифрования, основанные на блочных шифрах. Также реализован алгоритм шифрования/дешифрования Serpent.

# **Ответы на контрольные вопросы**

**1. Что называется ключом шифрования?**

Ключ шифрования – это информация, используемая криптографическим алгоритмом при шифровании/расшифровке сообщений.

**2. Какие шифры называются шифрами на открытом ключе и какие - шифрами на секретном ключе?**

В шифрах на секретном ключе один и тот же ключ используется как для шифрования, так и для дешифрования данных. В шифрах на открытом ключе при шифровании и дешифровании используются разные ключи: открытый и общеизвестный – при шифровании, закрытый, секретный – при дешифровании.

**3. Чем отличается поточное шифрование от блочного?**

Поточный шифр способен обрабатывать информацию побитно, т. е. подобная схема может, получив порцию из произвольного количества бит (может быть, даже одного), зашифровать/дешифровать ее и передать для дальнейшей обработки другим модулям. В блочных шифрах преобразования могут применяться только над информацией строго определенного объема.

**4. Чему равна длина секретного ключа алгоритма DES (с учетом контрольных разрядов, без учета контрольных разрядов)?**

56 бит без учета контрольных разрядов, 64 бита с учетом контрольных разрядов.

**5. Чему может быть равна длина секретных ключей алгоритма Rijndael?**

Алгоритм поддерживает размер ключа, кратный 32 битам

**6. Какая архитектура лежит в основе алгоритма DES?**

Алгоритм представляет собой классическую сеть Фейстеля из 16 раундов с добавлением входной и выходной перестановки бит.

**7. Какой параметр определяет криптостойкость симметричного алгоритма?**

Основным параметром криптостойкости является длина секретного ключа.