Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 1

«**Симметричная криптография. Стандарт**

**шифрования ГОСТ 28147-89**»

Выполнила студентка гр. 053501

Криштафович К. Д.

Проверил

Лещенко Е. А.

Минск, 2023

**ВВЕДЕНИЕ**

Цель данной лабораторной работы – реализовать программные средства шифрования и дешифрования текстовых файлов при помощи стандарта шифрования ГОСТ 28147-89 в одном из следующих режимов:

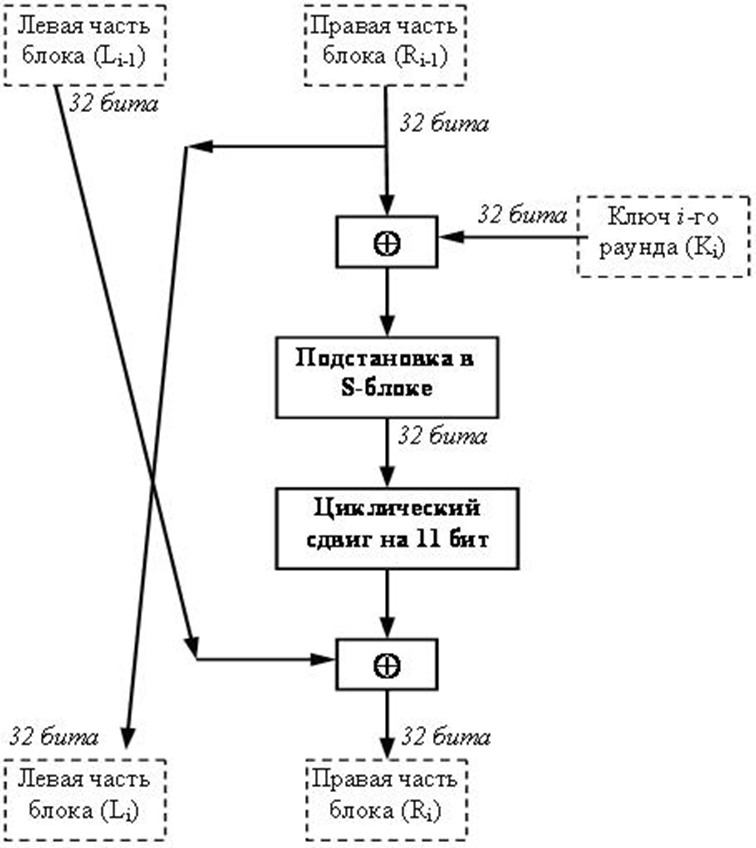
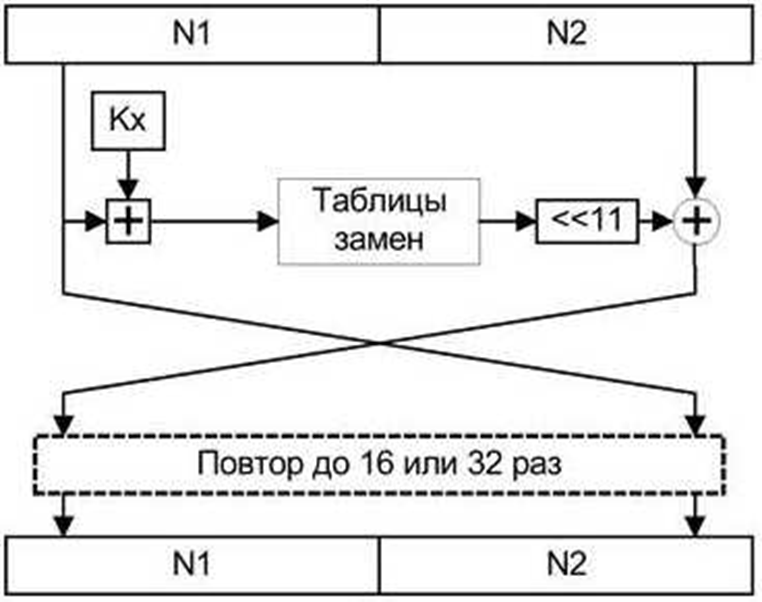
1. Простой замены;
2. Гаммирования;
3. Гаммирования с обратной связью;
4. Генерации имитоприставок.

**ЗАДАНИЕ:**

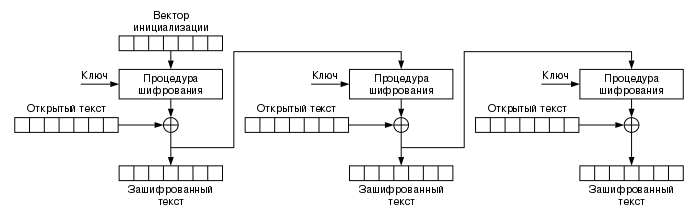
1. Изучить теоретические сведения.
2. Создать программу, читающие данные из файла и шифрующие (дешифрующие) их с помощью с помощью шифра ГОСТ 28147-89 в режиме гаммирования с обратной связью на языке Python.

**БЛОК-СХЕМЫ АЛГОРИТМОВ**

Схема основного шага алгоритма ГОСТ 28147-89:



Режим гаммирования с обратной связь. ГОСТ 28147-89:

****

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**ГОСТ 28147-89** представляет собой симметричный 64-битовый блочный алгоритм с 256-битовым ключом.

**Описание работы ГОСТ 28147-89**

Шифруемый блок данных разбивается на две части, которые затем обрабатываются как отдельные 32-битовые целые числа без знака.

Сначала правая половина блока и подключ раунда складываются по модулю 232.

Затем производится поблочная подстановка.

32-битовое значение, полученное на предыдущем шаге (обозначим его S), интерпретируется как массив из восьми 4-битовых блоков кода: S=(S0,S1,S2,S3,S4,S5,S6,S7).

Далее значение каждого из восьми блоков заменяется на новое, которое выбирается **по таблице замен**

В каждой строке таблицы замен записаны числа от 0 до 15 в произвольном порядке без повторений.

Значения элементов таблицы замен взяты от 0 до 15, так как в четырех битах, которые подвергаются подстановке, может быть записано целое число без знака в диапазоне от 0 до 15.

Значение блока S1 (четыре младших бита 32-разрядного числа S) заменяется на число, стоящее на позиции, номер которой равен значению заменяемого блока.

Например, в этом случае S1=0 заменится на 4, если S1=1, то оно заменится на 10 и т.д.

**Описание ГОСТ 28147-89**

После выполнения подстановки все 4-битовые блоки снова объединяются в единое 32-битное слово, которое затем циклически сдвигается на 11 битов влево.

Наконец, с помощью побитовой операции "сумма по модулю 2" результат объединяется с левой половиной, вследствие чего получается новая правая половина Ri.

Новая левая часть Li берется равной младшей части преобразуемого блока: Li= Ri-1.

Полученное значение преобразуемого блока рассматривается как результат выполнения одного раунда алгоритма шифрования.

**Процедуры шифрования и расшифрования**

ГОСТ 28147-89 является блочным шифром, поэтому преобразование данных осуществляется блоками в так называемых базовых циклах.

Базовые циклы заключаются в многократном выполнении для блока данных основного раунда, рассмотренного нами ранее, с использованием разных элементов ключа и отличаются друг от друга порядком использования ключевых элементов.

В каждом раунде используется один из восьми возможных 32-разрядных подключей.

Рассмотрим процесс создания подключей раундов.  
В ГОСТ эта процедура очень проста, особенно по сравнению с DES.  
256-битный ключ K разбивается на восемь 32-битных подключей, обозначаемых K0, K1, K2, K3, K4, K5, K6, K7. Алгоритм включает 32 раунда, поэтому каждый подключ при шифровании используется в четырех раундах в последовательности, представленной в таблице

Процесс расшифрования производится по тому же алгоритму, что и шифрование. Единственное отличие заключается в порядке использования подключей Ki. При расшифровании подключи должны быть использованы в обратном порядке, а именно, как указано в таблице

**ГОСТ 28147-89 Режимы работы**

**Режим простой замены**: все блоки шифруются независимо друг от друга с разными подключами в разных раундах. Для одинаковых блоков сообщения М блоки шифртекста будут одинаковыми.

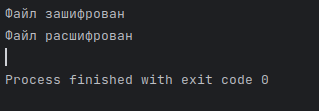
**Режим гаммирования**: В регистры N1 и N2 записывается 64-битовая синхропосылка (вектор инициализации) и шифруется с использованием СК. Результат подается на вход регистров и снова шифруется с использованием ключа. Получается «одноразовый блокнот».

**В режиме гаммирования с обратной связью** для заполнения регистров N1 и N2, начиная со 2-го блока, используется результат зашифрования предыдущего блока открытого текста.

**Работа криптосистемы в режиме гаммирования**



**РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

****

**Содержимое файла input.txt:**

Hello world

**Содержимое файла encrypted.txt:**

10151954460257647302 14860227113165849011 9195664452787785222 785422174802249546 7037017023373720651 14563474081826010458 18006394775058157994 12213200969116622331 4779502575269742811 7180168465878109043 8061910616000457815

**Содержимое файла decrypted.txt:**

Hello world

**ВЫВОДЫ**

В результате выполнения данной лабораторной работы был реализован алгоритм шифрования ГОСТ 28147-89 в режиме гаммирования с обратной связью. Были получены следующие выводы:

Преимущества алгоритма ГОСТ 28147-89: бесперспективность силовой атаки, эффективность реализации и соответственно высокое быстродействие на современных компьютерах.

Недостатки алгоритма: Основные проблемы ГОСТа связаны с неполнотой стандарта в части генерации ключей и таблиц замен. Считается, что у ГОСТа существуют «слабые» ключи и таблицы замен, но в стандарте не описываются критерии выбора и отсева «слабых».

**КОД ПРОГРАММЫ**

blocks = (

(4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3),

(14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9),

(5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11),

(7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3),

(6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2),

(4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14),

(13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12),

(1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12),

)

key = 18318279387912387912789378912379821879387978238793278872378329832982398023031

def bit\_length(value):

return len(bin(value)[2:])

class Crypt(object):

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

assert bit\_length(key) <= 256

self.\_key = None

self.\_subkeys = None

self.key = key

self.sbox = sbox

@property

def key(self):

return self.\_key

@key.setter

def key(self, key):

assert bit\_length(key) <= 256

# Для генерации подключей исходный 256-битный ключ разбивается на восемь 32-битных блоков: K1…K8.

self.\_key = key

self.\_subkeys = [(key >> (32 \* i)) & 0xFFFFFFFF for i in range(8)] # 8 кусков

def \_f(self, part, key):

"""Функция шифрования (выполняется в раудах)"""

assert bit\_length(part) <= 32

assert bit\_length(part) <= 32

temp = part ^ key # складываем по модулю

output = 0

# разбиваем по 4бита

# в рез-те sbox[i][j] где i-номер шага, j-значение 4битного куска i шага

# выходы всех восьми S-блоков объединяются в 32-битное слово

for i in range(8):

output |= ((self.sbox[i][(temp >> (4 \* i)) & 0b1111]) << (4 \* i))

# всё слово циклически сдвигается влево (к старшим разрядам) на 11 битов.

return ((output >> 11) | (output << (32 - 11))) & 0xFFFFFFFF

def \_decrypt\_round(self, left\_part, right\_part, round\_key):

return left\_part, right\_part ^ self.\_f(left\_part, round\_key)

def encrypt(self, msg):

# "Шифрование исходного сообщения"

def \_encrypt\_round(left, right, round\_key):

return right, left ^ self.\_f(right, round\_key)

assert bit\_length(msg) <= 64

# открытый текст сначала разбивается на две половины

# (младшие биты — rigth\_path, старшие биты — left\_path)

left\_part = msg >> 32

right\_part = msg & 0xFFFFFFFF

# Выполняем 32 рауда со своим подключом Ki

# Ключи K1…K24 являются циклическим повторением ключей K1…K8 (нумеруются от младших битов к старшим).

for i in range(24):

left\_part, right\_part = \_encrypt\_round(left\_part, right\_part, self.\_subkeys[i % 8])

# Ключи K25…K32 являются ключами K1…K8, идущими в обратном порядке.

for i in range(8):

left\_part, right\_part = \_encrypt\_round(left\_part, right\_part, self.\_subkeys[7 - i])

return (left\_part << 32) | right\_part # сливаем половинки вместе

def decrypt(self, crypted\_msg):

"""Дешифрование криптованого сообщения

Расшифрование выполняется так же, как и зашифрование, но инвертируется порядок подключей Ki."""

def \_decrypt\_round(left\_part, right\_part, round\_key):

return right\_part ^ self.\_f(left\_part, round\_key), left\_part

assert bit\_length(crypted\_msg) <= 64

left\_part = crypted\_msg >> 32

right\_part = crypted\_msg & 0xFFFFFFFF

for i in range(8):

left\_part, right\_part = \_decrypt\_round(left\_part, right\_part, self.\_subkeys[i])

for i in range(24):

left\_part, right\_part = \_decrypt\_round(left\_part, right\_part, self.\_subkeys[(7 - i) % 8])

return (left\_part << 32) | right\_part # сливаем половинки вместе

def CFB\_encrypt(self, msg):

vec\_init = 0x1234567891234567

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

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

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

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

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

return msg

def CFB\_decrypt(self, msg):

vec\_init = 0x1234567891234567

temp\_one = msg[0]

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

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

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

temp\_two = msg[i]

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

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

temp\_one = temp\_two

return msg

def main():

with open("input.txt", "r") as f:

text = f.read()

gost = Crypt(key, blocks)

s = []

for i in text:

s.append(ord(i))

cyphred = gost.CFB\_encrypt(s)

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

print(\*cyphred, file=file)

print("Файл зашифрован")

decyphred = gost.CFB\_decrypt(cyphred) # тут будет храниться расшифрованный текст

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

# объеденяем расшифрованные символы в строку и записываем в файл

file.write(bytes(decyphred))

print("Файл расшифрован")