Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчет по лабораторной работе №1

“ Симметричная криптография. Стандарт шифрования ГОСТ 28147-89”

          Выполнил  
 студент гр. 053504   
 Горожанкин В.О.

          Проверил:  
 ассистент каф. информатики   
 Лещенко Евгений Александрович

Минск   2023

# СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc145506676)

[2 Теоретическая часть 4](#_Toc145506677)

[3 Программная реализация алгоритма 6](#_Toc145506678)

[4 Исходный код программы 7](#_Toc145506679)

1 ПОСТАНОВКА ЗАДАЧИ

В данной лабораторной работе необходимо реализовать программные средства шифрования и дешифрования текстовых файлов при помощи стандарта шифрования ГОСТ 28147-89. В соответствии с вариантом шифрование и дешифрование должно работать в режиме простой замены. Программное средство должно быть реализовано на языке программирования *Python*.

# 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Алгоритм ГОСТ 28147 является отечественным стандартом для алгоритмов симметричного шифрования. ГОСТ 28147 разработан в 1989 году, является блочным алгоритмом шифрования, длина блока равна 64 битам, длина ключа равна 256 битам, количество раундов равно 32. Алгоритм представляет собой классическую сеть Фейштеля. Блок делится на левую L и правую R части. После чего в каждом раунде вычисляются новые значения для этих частей по уравнениям (1) и (2). Схема алгоритма приведена на рисунке 1.

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |

Функция F проста. Сначала правая половина и i-ый подключ складываются по модулю 232. Затем результат разбивается на восемь 4-битовых значений, каждое из которых подается на вход *S-*box. ГОСТ 28147 использует восемь различных *S-*boxes, каждый из которых имеет 4-битовый вход и 4-битовый выход. Выходы всех *S-*boxes объединяются в 32-битное слово, которое затем циклически сдвигается на 11 битов влево. Наконец, с помощью XOR результат объединяется с левой половиной, в результате чего получается новая правая половина.



Рисунок 1 – 1-ый раунд ГОСТ 28147

Генерация ключей проста. 256-битный ключ разбивается на восемь 32-битных подключей. Алгоритм имеет 32 раунда, поэтому каждый подключ используется в четырех раундах по следующей схеме из таблицы 1:

Таблица 1 – Раунды алгоритма

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Раунд | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Подключ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Раунд | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Подключ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Раунд | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Подключ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Раунд | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| Подключ | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Считается, что стойкость алгоритма ГОСТ 28147 во многом определяется структурой S-boxes. Входом и выходом S-box являются 4-битные числа, поэтому каждый S-box может быть представлен в виде строки чисел от 0 до 15, расположенных в некотором порядке. Тогда порядковый номер числа будет являться входным значением S-box, а само число - выходным значением S-box.

В режиме гаммирования с обратной связью предшествующий зашифрованный блок используется в качестве входа в алгоритм; к J битам выхода алгоритма и следующему незашифрованному блоку из J битов применяется операция XOR, результатом которой является следующий зашифрованный блок из J битов. Типичные приложения - потокоориентированная передача, аутентификация.

# 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА

В качестве ключа шифрования используется строка *Travelling to far countries is always a thrilling and interesting adventure.* Программа поддерживает шифрование и дешифрование файла в зависимости от указанного флага: -*e* (зашифровать файл) и -*d* (дешифровать файл). При зашифровывании файла, представленного на рисунке 2, 3-м параметром аргумента идет файл, откуда будет считываться исходный текст, а 4-м параметром идет файл, куда будет записан зашифрованный текст. В случае дешифрования, представленного на рисунке 3, 3-м параметром идет файл с зашифрованным текстом, а 4-м параметром указывается файл, куда требуется расшифровать и записать текст. Содержимое начальных данных для шифрования, данные после шифрования и данные после шифрования представлены на рисунках 4, 5 и 6 соответственно.

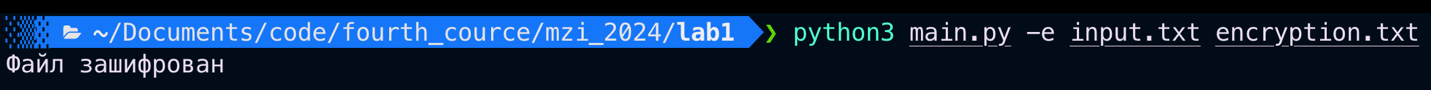


Рисунок 2 – Команда для шифрования файла

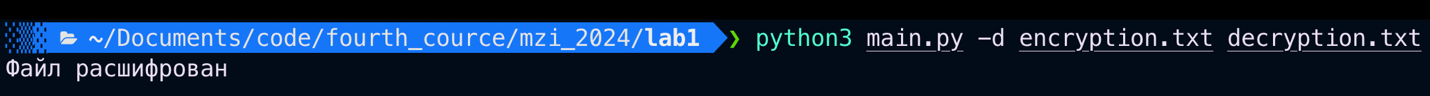


Рисунок 3 – Команда для дешифрования файла

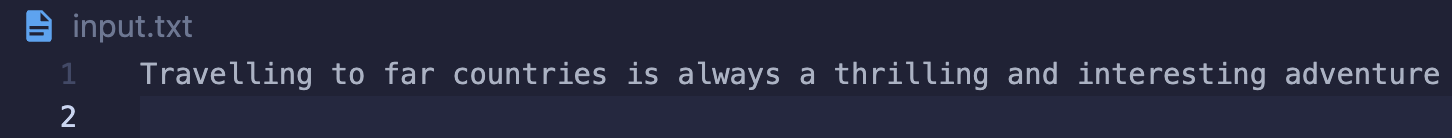


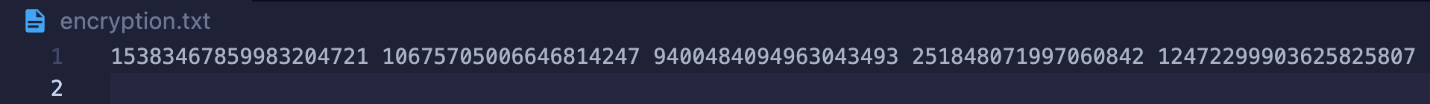
Рисунок 4 –Данные для шифрования  
  


Рисунок 5 – Данные после шифрования

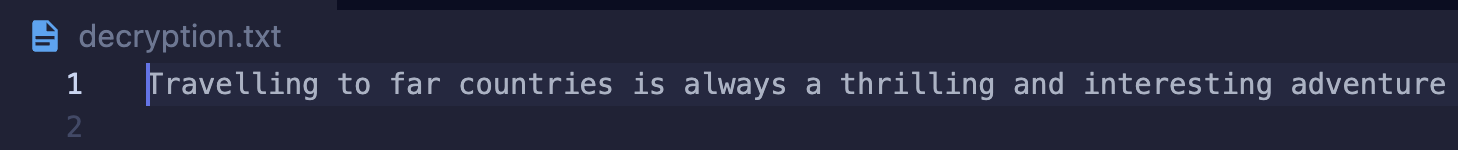


Рисунок 6 – Данные после дешифрования

# 4 ИСХОДНЫЙ КОД ПРОГРАММЫ

Исходный код программы приведен в листинге 1.

Листинг 1 – Исходный код программы

Файл *simple\_replacement.py:*

# получаем длину в битах

def bit\_length(value):

return len(bin(value)[2:]) # удаляем '0b' в начале

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

Файл *main.py:*

import sys

from encryption import \*

def main(argv=None):

# Блоки сдвигов при шифровании

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),

)

# ключ

key = 18318279387912387912789378912379821879387978238793278872378329832982398023031

# Если аргументов не 4, название файла и 3 аргумента, выходим

if len(sys.argv) != 4:

print("Нужно указать 3 аргумента, первый -s или -d для шифрования или расшифровки\n" +

"второй и третий это имена файлов, из какого брать тексь и куда сохранять")

return

# если не указан аргумент шифровки-дешифровки выходим

if sys.argv[1] != '-s' and sys.argv[1] != '-d':

print("Укажите опции -s или -d для шифрования или расшифровки")

return

# если требуется зашифровать

if sys.argv[1] == '-s':

cyphred = [] # тут будет хранится зашифрованный текст

gost = Crypt(key, blocks)

try:

s = []

# Читаем из файла текст и шифруем каждую букву

with open(sys.argv[2], 'rb') as file:

byte = file.read(1)

while byte:

s.append(ord(byte))

byte = file.read(1)

for x in s:

cyphred.append(gost.encrypt(x))

except: # если не удалось открыть файл, выходим

print(f"Не удалось открыть файл {sys.argv[2]}")

return

try:

# записываем зашифрованный текст в файл

with open(sys.argv[3], 'w') as file:

print(\*cyphred, file=file)

print("Файл зашифрован")

except:

print(f"Не удалось открыть файл {sys.argv[3]}")

return

# если требуется расшифроать

if sys.argv[1] == '-d':

decyphred = [] # тут будет храниться расшифрованный текст

gost = Crypt(key, blocks)

try:

with open(sys.argv[2]) as file:

s = file.read()

for x in s.split():

# расшифровываем текст из файла и добавляем его в список

decyphred.append(gost.decrypt(int(x)))

except:

print(f"Не удалось открыть файл {sys.argv[2]}")

return

try:

with open(sys.argv[3], 'wb') as file:

# объеденяем расшифрованные символы в строку и записываем в файл

file.write(bytes(decyphred))

print("Файл расшифрован")

except:

print(f"Не удалось открыть файл {sys.argv[3]}")

return

if \_\_name\_\_ == "\_\_main\_\_":

main()