Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЕТ

по лабораторной работе №1

на тему

**Симметричная криптография. Стандарт**

**шифрования ГОСТ 28147-89**

|  |  |  |
| --- | --- | --- |
| Студент |  | Д. А. Демидова |
|  |  |  |
| Проверил |  | Е. А. Лещенко |

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#__RefHeading___Toc7947_2113241753)

[2 Результат работы программы 4](#__RefHeading___Toc7949_2113241753)

[Вывод 5](#__RefHeading___Toc7951_2113241753)

[Приложение А](#__RefHeading___Toc7953_2113241753) [(обязательное)](#__RefHeading___Toc7955_2113241753) [Блок-схема алгоритма ГОСТ 28147-89 6](#__RefHeading___Toc7957_2113241753)

[Приложение Б](#__RefHeading___Toc7959_2113241753) [(обязательное)](#__RefHeading___Toc7961_2113241753) [Листинг программного кода 7](#__RefHeading___Toc7963_2113241753)

**1 ЦЕЛЬ РАБОТЫ**

В данной лабораторной работе необходимо изучить теоретические сведения, а также реализовать программное средство шифрования и дешифрования текстовых файлов при помощи стандарта шифрования ГОСТ 28147-89 в режиме шифрования методом гаммирования.

**2 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

**Содержимое файла input.txt:**

hello my name is dasha

i'm studying in bsuir

lalalalalalalalalalala

**Содержимое файла encrypted.txt:**

­âj895]8Jpñ$e²Á{cÉkJ#¹lcÒÉ%J#¹rcÉaJ#¹rcßÉdJe¹hcÉhJO¹rcÃÉpJ#¹xcÞÉkJ#¹!cÞÉkJO¹ccÄÉpJ#¹sc½ÉiJ#¹mcÖÉiJ#¹mcÖÉiJ#¹mcÖÉiJ#¹mcÖÉiJ#¹mcÖÉiJ#

**Содержимое файла decrypted.txt:**

hello my name is dasha

i'm studying in bsuir

lalalalalalalalalalala

**ВЫВОД**

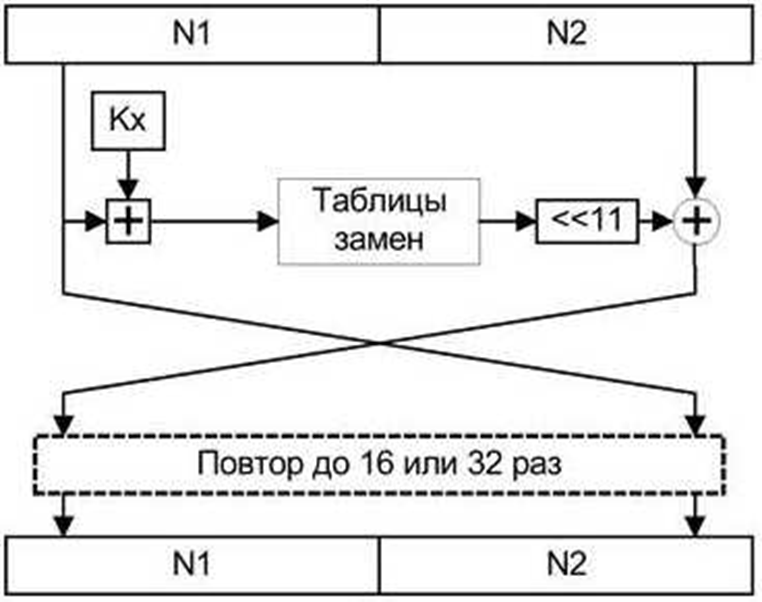
В ходе выполнения данной лабораторной работы были изучены теоретические сведения об алгоритме шифрования ГОСТ 28147-89 в режимах простой замены, гаммирования, гаммирования с обратной связью, генерации имитоприставок.

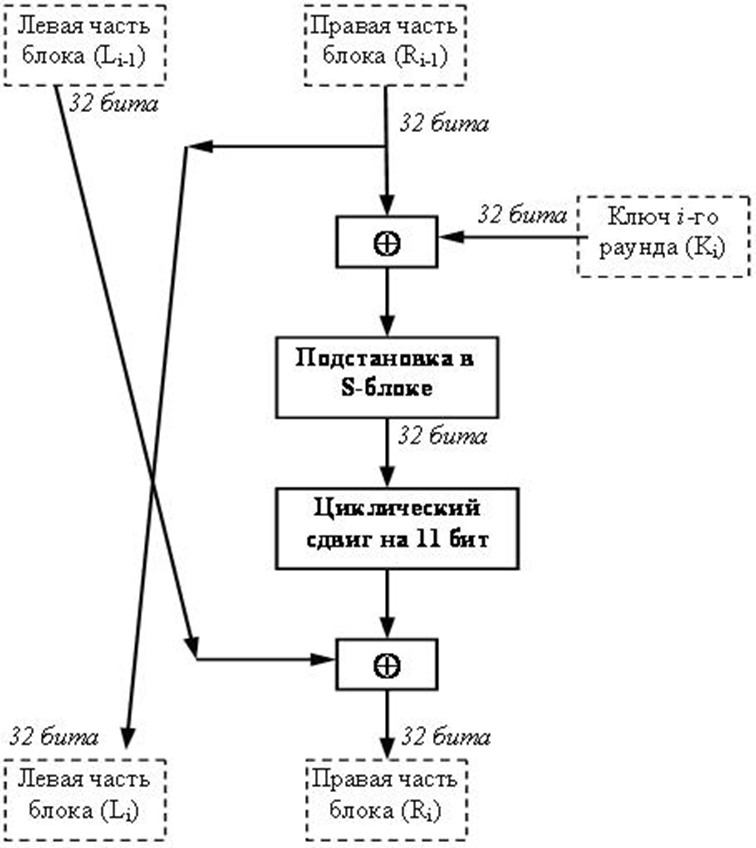
Также было реализовано консольное приложение, осуществляющее шифрование и дешифрование содержимого текстовых файлов с помощью алгоритма при помощи стандарта шифрования ГОСТ 28147-89 в режиме шифрования методом гаммирования.

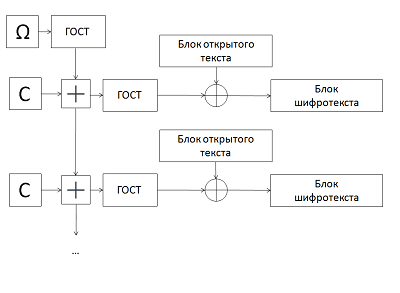
**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Блок-схема алгоритма ГОСТ 28147-89**







**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Листинг программного кода**

**Файл main.py**

from lab1.gost28147 import encode\_file, decode\_file

def main():

encode\_file('text.txt')

decode\_file('encrypted-text.txt')

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Файл gost28147.py**

from random import choice

from string import ascii\_uppercase

from lab1.text\_helpers import str\_to\_binary, text\_to\_binary, binary\_to\_text

KEY = ''.join(choice(ascii\_uppercase) for \_ in range(32)) # ключ (8х32бит)

SYNC = ''.join(choice(ascii\_uppercase) for \_ in range(8)) # синхропосылка (начальное состояние гаммы)

# константы для генерации элементов гаммы

C1 = 0x1010101

C2 = 0x1010104

BLOCK\_SIZE = 4

# таблица замен

TABLE = [

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

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

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

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

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

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

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

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

]

def prepare\_keys():

# переводим ключ в двоичку

return str\_to\_binary(KEY)

def prepare\_sync(text: str):

# переводим синхропосылку в двоичку

return str\_to\_binary(text)

def sum\_bits\_32(first: bin, second: bin):

# суммирование по модулю 2^32

int\_sum = int(first, 2) + int(second, 2)

return format(int\_sum, '32b')

def sum\_bits\_32\_1(first: bin, second: bin):

# суммирование по модулю 2^32 - 1

int\_sum = (int(first, 2) + int(second, 2)) % (2 \*\* 32 - 1)

return format(int\_sum, '32b')

def get\_enc\_key(index: int):

# высчитываем индекс, чтобы брать ключи К1-К8 циклически

# К1-К24 циклически повторяют К1-К8

# К25-К32 повторяют К8-К1

return index % 8 if index < 24 else 7 - index % 8

def move\_block\_by\_table(block: str) -> str:

# делим на 4-битовые последовательности

nums = [block[i:i + 4] for i in range(0, len(block), 4)]

# применяем таблицу замен

for i in range(len(nums)):

nums[i] = format(TABLE[i][int(nums[i], 2)], '04b')

return ''.join(nums)

def xor\_values(first: str, second: str) -> str:

size = len(first)

result = int(first, 2) ^ int(second, 2)

result = format(result, f'{size}b')

return result

def simple\_change(text\_block: list, keys: list, key\_func) -> (str, str):

left, right = text\_block[0], text\_block[1]

# 32 раунда

for i in range(32):

# получаем ключи

key = keys[key\_func(i)]

# сумма по модулю 2^32 левой части блока с ключом

enc = sum\_bits\_32(left, key)[-32:]

# делим на 4-битовые последовательности, применяем таблицу замен

enc = move\_block\_by\_table(enc)

# объединяем выходы таблицы замен в одно 32-битное слово

enc = int(enc, 2)

# сдвигаем слово влево на 11 бит

enc = format((enc << 11), '32b')[-32:]

# левая часть = ксор левой и правой частей

enc = xor\_values(enc, right)

# меняем местами части

if i < 31:

right = left

left = enc

else:

right = enc

return left, right

def change\_gamma(gamma: list) -> (str, str):

# суммирование с константами по модулю

left, right = gamma[0], gamma[1]

parsed\_c1 = format(C1, '32b')

parsed\_c2 = format(C2, '32b')

new\_right = sum\_bits\_32(right, parsed\_c2)[-32:]

new\_left = sum\_bits\_32\_1(left, parsed\_c1)[-32:]

return new\_left, new\_right

def get\_gamma(gamma: list, keys: list) -> (str, str):

if len(gamma) == 0:

sync = prepare\_sync(SYNC)

# шифруем синхропосылку алгоритмом простой замены

# и записываем результат в две части

left\_block, right\_block = simple\_change(sync, keys, get\_enc\_key)

else:

left\_block, right\_block = gamma[0], gamma[1]

# суммируем половинки с константами

left\_block, right\_block = change\_gamma([left\_block, right\_block])

# снова прогоняем половинки через алгоритм простой замены

left\_block, right\_block = simple\_change([left\_block, right\_block], keys, get\_enc\_key)

left\_block = left\_block.replace(' ', '0')

right\_block = right\_block.replace(' ', '0')

return left\_block, right\_block

def encode\_text(text: str, gamma: list, decode=False) -> (str, str):

# получаем массив ключей 8х32бита

keys = prepare\_keys()

new\_text = ''.join(text\_to\_binary(text, decode))

# получаем гамму

gamma = get\_gamma(gamma, keys)

gamma = f'{gamma[0]}{gamma[1]}'

# обработка случаев, когда длина блока не кратна 8/4

if decode and len(text) % 8 != 0:

gamma = gamma[:8 \* len(text)]

elif len(text) % 4 != 0:

gamma = gamma[:16 \* len(text)]

# сумма по модулю 2 с гаммой (тут происходит как шифрование,

# так и дешифрование, т.к. операция обратна себе)

encoded\_text = xor\_values(new\_text, gamma)

return encoded\_text.replace(' ', '0'), gamma

def encode\_file(filename: str, bit\_size=None, decode=False):

gamma = []

if decode is True:

bit\_size = 16

res\_file\_name = f'lab1/decrypted-{filename}' if decode else f'lab1/encrypted-{filename}'

with open(res\_file\_name, 'w') as output\_file:

with open(f'lab1/{filename}', 'r') as input\_file:

block\_size = BLOCK\_SIZE

if decode:

block\_size \*= 2

# обрабатываем файл чанками

text = input\_file.read(block\_size)

while text:

result, gamma = encode\_text(text, gamma, decode)

res = binary\_to\_text(result, bit\_size)

output\_file.write(res)

text = input\_file.read(block\_size)

input\_file.close()

output\_file.close()

def decode\_file(filename: str):

encode\_file(filename, decode=True)

**Файл text\_helpers.py**

def str\_to\_binary(s: str) -> list:

# берем аски код каждого символа в строке и

# переводим в двоичку (каждый символ длины 8 бит)

symbols = [

[format(ord(j), '08b') for j in s[i:i + 4]]

for i in range(len(s) - 4, -4, -4)

]

bin\_repr = [''.join(symbol) for symbol in symbols]

return bin\_repr

def text\_to\_binary(text: str, decode=False) -> list:

result = []

if decode:

bits = '08b'

else:

bits = '16b'

for txt in text:

for t in txt:

result.append(format(ord(t), bits))

if decode and len(result) > 1:

new\_result = []

for i in range(0, len(result), 2):

new\_result.append(result[i] + result[i + 1])

result = new\_result

return [sym.replace(' ', '0') for sym in result]

def binary\_to\_text(binary: str, bit\_size: str) -> str:

result\_str = ''

if bit\_size is None:

bit\_size = 8

for i in range(0, len(binary), bit\_size):

result\_str += chr(int(binary[i:i + bit\_size], 2))

return result\_str