**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

по дисциплине **«Методы защиты информации»**

на тему: «**Стандарты симметричного шифрования DES и**

**ГОСТ 28147-89**»

Выполнили: студент гр. ИП-41

Шевкунова В.А.

Пикун Я.И.

Коваленко А.И.

Принял: к.т.н., доцент

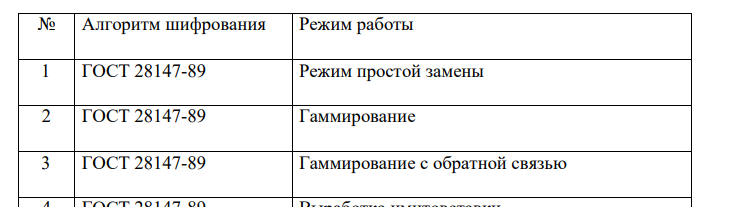
Прокопенко Дмитрий Викторович

Гомель 2023

**Цель**: изучить алгоритмы симметричного шифрования информации DES и ГОСТ 28147-89.

**Задание** I. Реализовать приложение для шифрования, позволяющее выполнять следующие действия: 1. Шифровать данные по заданному в варианте алгоритму: ‒ шифруемый текст должен храниться в одном файле, а ключ шифрования – в другом; ‒ зашифрованный текст должен сохраняться в файл; ‒ в процессе шифрования предусмотреть возможность просмотра и изменения ключа, шифруемого и зашифрованного текстов в шестнадцатеричном и символьном виде; II. Реализовать приложение для дешифрования, позволяющее выполнять следующие действия: 2. Дешифровать данные по заданному в варианте алгоритму: ‒ зашифрованный текст должен храниться в одном файле, ключ – в другом; ‒ расшифрованный текст должен сохраняться в файл; ‒ в процессе дешифрования предусмотреть возможность просмотра и изменения ключа, зашифрованного и расшифрованного текстов в шестнадцатеричном и символьном виде.

Вариант 3



**Листинг программы:**

*#возвращает массив из 8 элементов, где каждый элемент - это 32 бита*def genKeys\_GOST(*key256b*:*str*):  
 *# Преобразуем ключ в бинарную строку* keys1 = *key256b*.encode()  
 keys = []  
 *# Разделяем ключ на 8 частей по 32 бита* for i in *range*(8):  
 keys.append(keys1[:4])  
 keys1 = keys1[4:]  
 keysBin = []  
 *# Преобразуем каждую часть ключа в строку из нулей и единиц* for i in *range*(*len*(keys)):  
 binStr = ''  
 for j in keys[i]:  
 binStr += *bin*(j)[2:].zfill(8)  
 keysBin.append(binStr)  
 return keysBin  
  
*# Функция подготовки текста перед кодированием (шифрованием или дешифрованием)*def before\_coding(*text*, *flag*: *int*):  
 file = 0  
 *# Проверяем тип текста (строка или бинарные данные)* if *type*(*text*) == *str*:  
 if *flag* == 1:  
 *# Если шифруем, кодируем текст* textToInt = *text*.encode()  
 else:  
 *# Если дешифруем, преобразуем строку в бинарные данные* textToInt = b''  
 for i in *text*:  
 textToInt += *bytes*([*ord*(i)])  
 else:  
 *# Если входные данные уже в бинарном формате* textToInt = *text* file = 1  
 *print*('OK')  
  
 blocks = []  
 *# Разбиваем данные на блоки по 64 бита (8 байт)* while *len*(textToInt) > 0:  
 blocks.append(textToInt[:8])  
 textToInt = textToInt[8:]  
  
 count = 0  
 if *len*(blocks[*len*(blocks) - 1]) != 8:  
 *# Если последний блок меньше 64 бит, добавляем нули и информацию о добавленных нулях* while *len*(blocks[*len*(blocks) - 1]) != 8:  
 count += 1  
 blocks[*len*(blocks) - 1] += *bytes*([0])  
 blocks[*len*(blocks) - 1] = blocks[*len*(blocks) - 1][:-1] + *bytes*([count])  
  
 blocksBin = []  
 *# Преобразуем каждый блок в строку из нулей и единиц* for i in *range*(*len*(blocks)):  
 binStr = ''  
 for j in blocks[i]:  
 binStr += *bin*(j)[2:].zfill(8)  
 blocksBin.append(binStr)  
 return blocksBin, file  
  
*# Функция для обработки данных после кодирования (шифрования или дешифрования)*def after\_coding(*file*, *block64Int*, *flag*):  
 if *file* == 0:  
 if *flag* != 1:  
 numb = *block64Int*[*len*(*block64Int*) - 1]  
 *print*('n', numb)  
 blocksText1 = *block64Int* j = 2  
 for i in *range*(numb-1):  
 if blocksText1[-j] == 0:  
 blocksText1 = blocksText1[:-1]  
 j = 1  
 else:  
 break  
 if *len*(blocksText1) < *len*(*block64Int*):  
 blocksText1 = blocksText1[:-1]  
 blocksText = ''  
 for i in blocksText1:  
 blocksText += *chr*(i)  
 else:  
 blocksText = ''  
 for i in *block64Int*:  
 blocksText += *chr*(i)  
 else:  
 if *flag* != 1:  
 numb = *block64Int*[*len*(*block64Int*) - 1]  
 *print*('n', numb)  
 blocksText = *block64Int* j = 2  
 for i in *range*(numb-1):  
 if blocksText[-j] == 0:  
 blocksText = blocksText[:-1]  
 j = 1  
 else:  
 break  
 if *len*(blocksText) < *len*(*block64Int*):  
 blocksText = blocksText[:-1]  
 else:  
 blocksText = *block64Int* return blocksText  
  
*# Функция для выполнения операции XOR над двумя строками из нулей и единиц*def xor(*a*: *str*, *b*: *str*):  
 if *len*(*a*) > *len*(*b*):  
 a = *a*.zfill(*len*(*b*))  
 elif *len*(*b*) > *len*(*a*):  
 b = *b*.zfill(*len*(*a*))  
 res = *bin*(*int*(*a*, 2) ^ *int*(*b*, 2))[2:].zfill(*len*(*a*))  
 return res  
  
*# Функция для циклического сдвига строки на n позиций*def sdvig(*str*: *str*, *n*: *int*):  
 arrTxt = *str*[*n*:] + *str*[:*n*]  
 return arrTxt  
  
*# Функция для замены 4-битных блоков по таблице S-боксов*def substitution(*N1*:*str*):  
 \_\_Sbox = [  
 [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]  
 ]  
  
 *# 8 блоков по 4 бита* blocks4b = []  
 for i in *range*(8):  
 blocks4b.append(*N1*[:4])  
 N1 = *N1*[4:].zfill(4)  
 blocksAfterSbox = ''  
 for i in *range*(8):  
 blocksAfterSbox+=*bin*(\_\_Sbox[i][*int*(blocks4b[i], 2)])[2:].zfill(4)  
 *# Выполняем циклический сдвиг на 11 позиций* return sdvig(blocksAfterSbox, 11)  
  
*# Функция для выполнения одного раунда схемы Фейстеля*def round\_feistel\_scheme(*L0*: *str*, *R0*: *str*, *key*: *str*):  
 *# RES = (N1 + Ki) mod 2 ^ 32* RES = *bin*((*int*(*L0*, 2) | *int*(*key*, 2)) % 2\*\*32)[2:]  
  
 *# RES = RES -> Sbox, << 11* RES = substitution(RES)  
 L0, R0 = xor(RES, *R0*), *L0* return *L0*, *R0  
  
# Функция для выполнения схемы Фейстеля над блоком*def feistel\_scheme(*block*:*str*, *keys*:*list*, *flag*:*int*):  
 L0 = *block*[:32]  
 R0 = *block*[32:]  
 if *flag* == 1:  
 *# Шифрование: 3 раунда прямого порядка и 1 раунд обратного порядка  
 # K0, K1, K2, K3, K4, K5, K6, K7, K0, K1, K2, K3, K4, K5, K6, K7, K0, K1, K2, K3, K4, K5, K6, K7* for i in *range*(3):  
 for j in *range*(*len*(*keys*)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, *keys*[j])  
 *# K7, K6, K5, K4, K3, K2, K1, K0* for i in *range*(*len*(*keys*)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, *keys*[*len*(*keys*) - 1 - i])  
 L0, R0 = R0, L0  
 else:  
 *# Дешифрование: 1 раунд прямого порядка и 3 раунда обратного порядка  
 # K0, K1, K2, K3, K4, K5, K6, K7* for i in *range*(*len*(*keys*)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, *keys*[i])  
 *# K7, K6, K5, K4, K3, K2, K1, K0, K7, K6, K5, K4, K3, K2, K1, K0, K7, K6, K5, K4, K3, K2, K1, K0* for i in *range*(3):  
 for j in *range*(*len*(*keys*)):  
 L0, R0 = round\_feistel\_scheme(L0, R0, *keys*[*len*(*keys*) - 1 - j])  
 L0, R0 = R0, L0  
 return L0 + R0  
  
def GOST\_gamm(*keys1*, *text*, *flag*, *vector\_init*):  
 *# Проверка наличия и корректности ключа, вектора и текста* if not *keys1* or *keys1* == '':  
 keys = genKeys\_GOST('this\_is\_a\_pasw\_for\_GOST\_28147\_89')  
 else:  
 if *len*(*keys1*) != 256:  
 *print*("Ошибка: Неправильный формат ключа!")  
 return ""  
  
 for i in *keys1*:  
 if i not in ('0', '1'):  
 *print*("Ошибка: Неверный формат ключа!")  
 return ""  
  
 keys = [0, 0, 0, 0, 0, 0, 0, 0]  
 for i in *range*(8):  
 keys[i] = *keys1*[:32]  
 keys1 = *keys1*[32:]  
  
 if not *vector\_init* or *vector\_init* == '':  
 vector\_init = '1110011110010101110010111001010010000111001010011100101001001011'  
 else:  
 if *len*(*vector\_init*) != 64:  
 *print*("Ошибка: Неправильный формат вектора инициализации!")  
 return ""  
  
 for i in *vector\_init*:  
 if i not in ('0', '1'):  
 *print*("Ошибка: Неверный формат вектора инициализации!")  
 return ""  
  
 if not *text*:  
 *print*("Ошибка: Введите текст!")  
 return ""  
  
 *print*("Ключ: ", keys)  
 *print*("Вектор: ", *vector\_init*)  
 *print*("Текст: ", *text*)  
  
 blocksBin, file = before\_coding(*text*, *flag*)  
  
 normal = 0  
  
 if *flag* == 1:  
 N = *vector\_init* block64Int = b''  
 for i in *range*(*len*(blocksBin)):  
 gamma = feistel\_scheme(N, keys, 1)  
 block64b = xor(gamma, blocksBin[i])  
 N = block64b  
 for j in *range*(8):  
 block64Int += *bytes*([*int*(block64b[:8], 2)])  
 block64b = block64b[8:]  
 else:  
 N = *vector\_init* block64Int = b''  
 for i in *range*(*len*(blocksBin)):  
 gamma = feistel\_scheme(N, keys, 1)  
 block64b = xor(gamma, blocksBin[i])  
 for j in *range*(8):  
 block64Int += *bytes*([*int*(block64b[:8], 2)])  
 block64b = block64b[8:]  
 N = blocksBin[i]  
  
 blocksText = after\_coding(file, block64Int, *flag*)  
 if normal == 0:  
 return blocksText  
def save\_to\_file(*text*, *file\_path*):  
 with *open*(*file\_path*, 'w', *encoding*='utf-8') as file:  
 file.write(*text*)  
  
  
def decrypt(*text*, *key*, *vector\_init*):  
 flag = 0 *# Дешифрование* result = GOST\_gamm(*key*, *text*, flag, *vector\_init*)  
 return result  
  
def main():  
 *# Задайте пути к файлам с ключом и вектором инициализации* key\_file\_path = 'key.txt'  
 vector\_file\_path = 'vector.txt'  
 text\_path = "input.txt"  
  
 *# Считываем данные, ключ и вектор инициализации из файлов* with *open*(key\_file\_path, 'r') as key\_file:  
 key = key\_file.read().strip()  
 with *open*(vector\_file\_path, 'r') as vector\_file:  
 vector\_init = vector\_file.read().strip()  
 with *open*(text\_path, 'r') as text\_file:  
 text = text\_file.read().strip()  
  
 flag = 1 *# Шифрование  
 print*("Шифрование: ")  
 encrypted\_text = GOST\_gamm(key, text, flag, vector\_init)  
 save\_to\_file(encrypted\_text, 'encrypted.txt')  
  
 *print*("Дешифрование: ")  
 decrypted\_text = decrypt(encrypted\_text, key, vector\_init)  
 save\_to\_file(decrypted\_text, 'decrypted.txt')  
  
 *print*("Исходный текст: ", text)  
 *print*("Зашифрованный текст: ", encrypted\_text)  
 *print*("Дешифрованный текст: ", decrypted\_text)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Результат работы.**

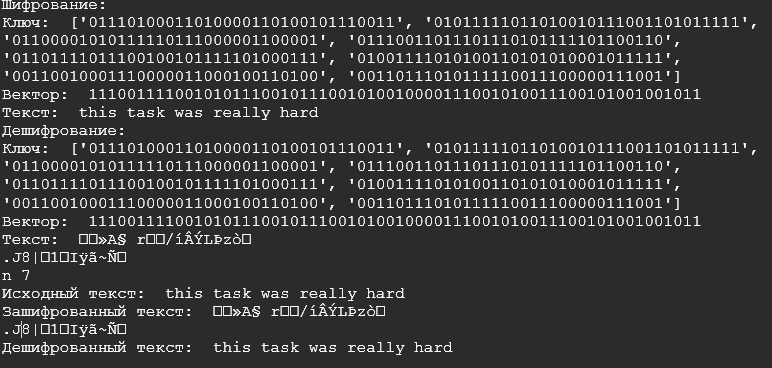
****

Рисунок 1 – Результат выполнения программы

**Вывод:** при выполнении работы были изучены алгоритмы симметричного шифрования информации ГОСТ 28147-89.