Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Лабораторная работа №1. Компьютерная реализация блочных шифров на примере DES и ГОСТ 28147-89

Выполнил: cтудент гр. 853501

Яковлев А.Б.

Проверил:

Протько М.И.

Минск 2021

# Постановка задачи и описание алгоритма

**Цель:** изучить теоретическую часть об алгоритмах шифрования DES и [ГОСТ 28147-89](https://ru.wikipedia.org/wiki/%D0%93%D0%9E%D0%A1%D0%A2_28147-89), которые необходимо использовать для шифрования данных, и на ее основании создать приложение, реализующее алгоритмы DES (двойной и тройной DES) и [ГОСТ 28147-89](https://ru.wikipedia.org/wiki/%D0%93%D0%9E%D0%A1%D0%A2_28147-89).

## Алгоритм шифрования DES

DES (англ. Data Encryption Standard) — алгоритм для симметричного шифрования, разработанный фирмой IBM и утверждённый правительством США в 1977 году как официальный стандарт (FIPS 46-3). Размер блока для DES равен 64 битам. В основе алгоритма лежит сеть Фейстеля с 16 циклами (раундами) и ключом, имеющим длину 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований. Прямым развитием DES в настоящее время является алгоритм Triple DES (3DES). В 3DES шифрование/расшифровка выполняются путём троекратного выполнения алгоритма DES.

***Схема шифрования с помощью алгоритма DES:***

* ***Начальная перестановка***

Начальная перестановка и ее инверсия определяются стандартной таблицей. Если М- это произвольные 64 бита, то X = IP (M)-переставленные 64 бита. Если применить обратную функцию перестановки Y = IP-1 (X) = IP-1 (IP(M)), то получится первоначальная последовательность бит.

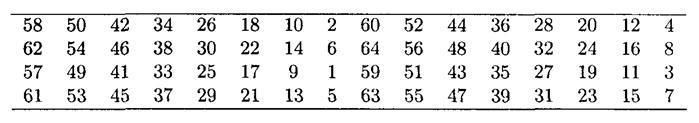


Рисунок 1 - DES. Начальная перестановка

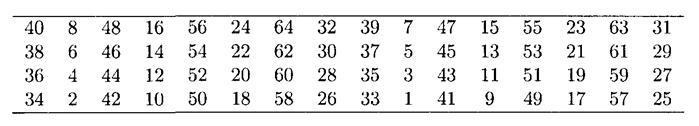


Рисунок 2 - DES. Заключительная перестановка

* ***Последовательность преобразований отдельного раунда***

Теперь рассмотрим последовательность преобразований, используемую в каждом *раунде*.

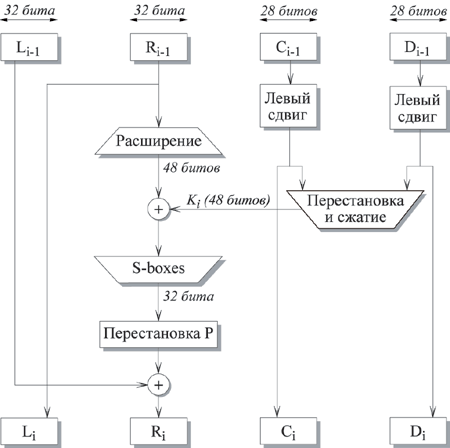


Рисунок 3 - i-ый раунд DES

64-битный входной блок проходит через 16 *раундов*, при этом на каждой итерации получается промежуточное 64-битное значение. Левая и правая части каждого промежуточного значения трактуются как отдельные 32-битные значения, обозначенные *L* и *R*. Каждую итерацию можно описать следующим образом:

Li = Ri-1

Ri = Li-1 F(Ri-1, Ki)

Где обозначает операцию XOR.

Таким образом, выход левой половины *Li* равен входу правой половины *Ri-1*. Выход правой половины *Ri*является результатом применения операции XOR к *Li-1* и функции *F*, зависящей от *Ri-1* и *Ki*.

Рассмотрим функцию *F* более подробно.

*Ri*, которое подается на вход функции *F*, имеет длину 32 бита. Вначале Ri расширяется до 48 бит, используя таблицу, которая определяет перестановку плюс расширение на 16 бит. Расширение происходит следующим образом. 32 бита разбиваются на группы по 4 бита и затем расширяются до 6 бит, присоединяя крайние биты из двух соседних групп. Например, если часть входного сообщения

. . . efgh ijkl mnop . . .

то в результате расширения получается сообщение

. . . defghi hijklm lmnopq . . .

После этого для полученного 48-битного значения выполняется операция XOR с 48-битным *подключом Ki*. Затем полученное 48-битное значение подается на вход функции подстановки, результатом которой является 32-битное значение.

Подстановка состоит из восьми *S-boxes,* каждый из которых на входе получает 6 бит, а на выходе создает 4 бита. Эти преобразования определяются специальными таблицами. Первый и последний биты входного значения *S-box* определяют номер строки в таблице, средние 4 бита определяют номер столбца. Пересечение строки и столбца определяет 4-битный выход. Например, если входом является 011011, то номер строки равен 01 (строка 1) и номер столбца равен 1101 (столбец 13). Значение в строке 1 и столбце 13 равно 5, т.е. выходом является 0101.

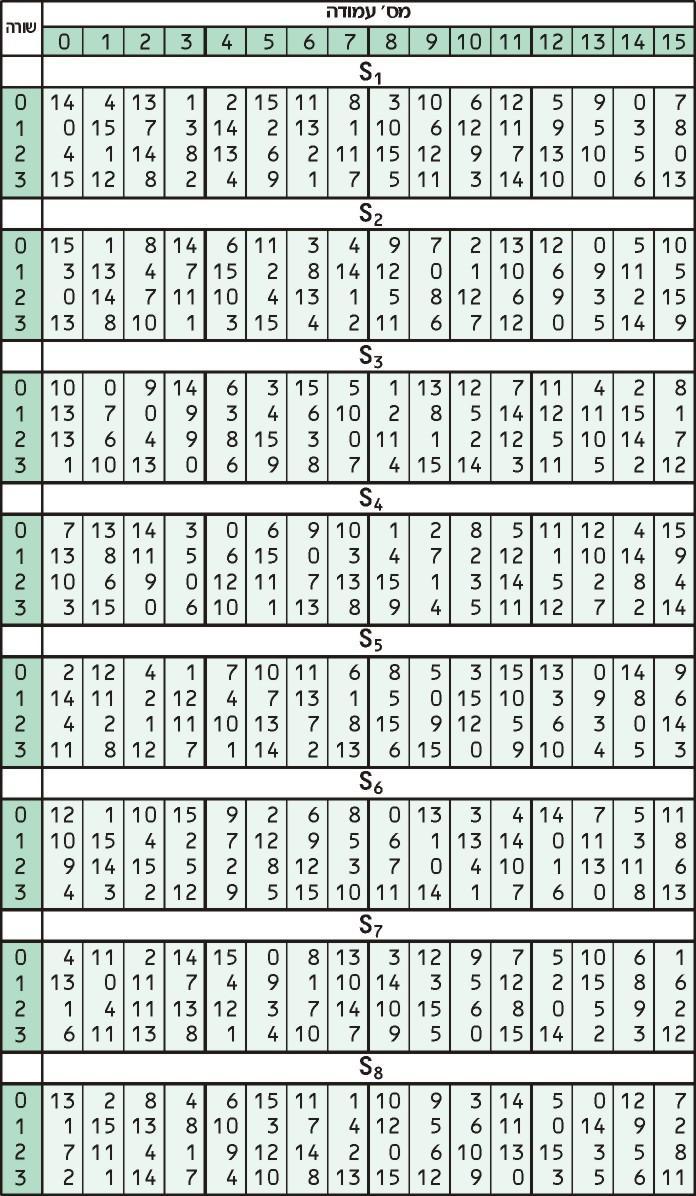


Рисунок 4 - S-boxes

Далее полученное 32-битное значение обрабатывается с помощью перестановки *Р*, целью которой является максимальное переупорядочивание бит, чтобы в следующем *раунде* шифрования с большой вероятностью каждый бит обрабатывался другим *S-box*.

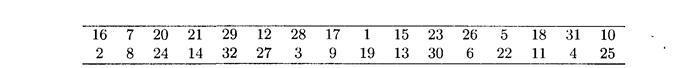


Рисунок 5 - Перестановка с помощью Р-блоков

* ***Создание подключей***

Ключ для отдельного *раунда Ki*состоит из 48 бит. Ключи *Ki* получаются по следующему алгоритму. Для 56-битного ключа, используемого на входе алгоритма (если используется 64-битный ключ, то, как видно из рис. 6 убираются биты 64, 56, 48, 40, 32, 16, 8), вначале выполняется перестановка в соответствии с таблицей Permuted Choice 1 (РС-1).

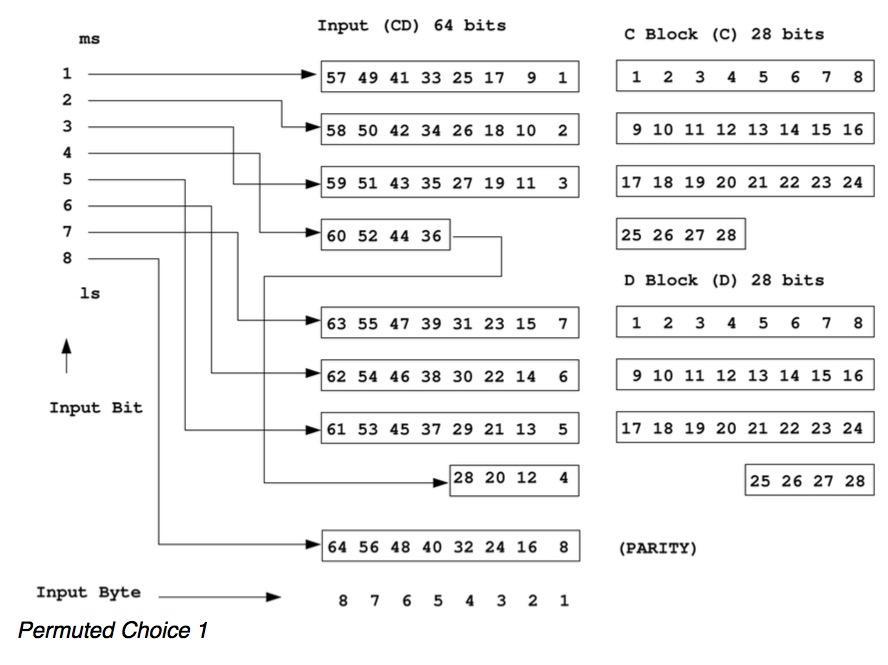


Рисунок 6 - Схема Permuted Choice

Полученный 56-битный ключ разделяется на две 28-битные части, обозначаемые как C0 и D0 соответственно. На каждом *раунде Ci* и *Di*независимо циклически сдвигаются влево на 1 или 2 бита, в зависимости от номера *цикла*.

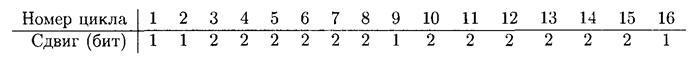


Рисунок 7 - Сдвиг ключа в зависимости от номера цикла

Полученные значения являются входом следующего *раунда*. Они также представляют собой вход в Permuted Choice 2 (РС-2), который создает 48-битное выходное значение, являющееся входом функции *F*(*Ri-1*, *Ki*).

Процесс ***дешифрования*** аналогичен процессу шифрования. На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности. *K16* используется на первом *раунде*, *K1* используется на последнем *раунде*.

## Двойной DES

Наиболее логичным способом противодействия полному перебору ключа DES выглядит многократное шифрование данных алгоритмом DES с различными ключами. Следующий алгоритм получил название Double DES (двойной DES):

где: — половины двойного ключа алгоритма Double DES, каждая из которых представляет собой обычный 56-битный ключ DES; Е — функция шифрования блока данных обычным алгоритмом DES. Если бы при двойном шифровании DES выполнялось следующее свойство:  для любых значений, то двойное шифрование не приводило бы к усилению против полного перебора ключа — всегда нашелся бы такой ключ *k*, *однократное* шифрование которым было бы эквивалентно двукратному шифрованию на ключах, а для нахождения ключа к достаточно было бы перебрать 255 ключей. Double DES действительно удваивает эффективный размер ключа— до 112 битов, а при современном развитии вычислительной техники полный перебор 112-битного ключа невозможен.

## Тройной DES

Triple DES (3DES) — симметричный [блочный шифр](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80), созданный [Уитфилдом Диффи](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B8,_%D0%A3%D0%B8%D1%82%D1%84%D0%B8%D0%BB%D0%B4), [Мартином Хеллманом](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD,_%D0%9C%D0%B0%D1%80%D1%82%D0%B8%D0%BD) и Уолтом Тачманном в [1978 году](https://ru.wikipedia.org/wiki/1978_%D0%B3%D0%BE%D0%B4) на основе алгоритма [DES](https://ru.wikipedia.org/wiki/DES) с целью устранения главного недостатка последнего — малой длины ключа (56 бит), который может быть взломан методом [полного перебора](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80) ключа. Скорость работы 3DES в 3 раза ниже, чем у DES, но криптостойкость намного выше — время, требуемое для криптоанализа 3DES, может быть в миллиард раз больше, чем время, нужное для вскрытия DES. 3DES используется чаще, чем DES, который легко взламывается при помощи сегодняшних технологий (в [1998 году](https://ru.wikipedia.org/wiki/1998_%D0%B3%D0%BE%D0%B4) организация [Electronic Frontier Foundation](https://ru.wikipedia.org/wiki/Electronic_Frontier_Foundation), используя специальный компьютер *DES Cracker*, вскрыла DES за 3 дня). 3DES является простым способом устранения недостатков DES. Алгоритм 3DES построен на основе DES, поэтому для его реализации возможно использовать программы, созданные для DES. Официальное название алгоритма, используемое в стандартах - TDEA или Triple DEA (англ. Triple Data Encryption Algorithm). Однако, термин "3DES" используется более широко поставщиками, пользователями и разработчиками криптосистем.

Существует 2 основных варианта шифрования алгоритмом 3DES: 3-key Triple DES и 2-key Triple DES. Как видно из названия, принципиальное различие этих методов - количество ключей (три и два соответственно). В свою очередь, каждый из этих алгоритмов имеет по 2 разных типа: EEE (encryption-encryption-encryption) и EDE (encryption-decryption-encryption) шифрование. Начнем с трех ключей:

* DES-EEE3: Исходный текст шифруется три раза, используя разные ключи.
* DES-EDE3: Исходный текст шифруется, затем дешифруется (уже другим ключом), затем снова шифруется (третьим ключом). Наглядное представление можно увидеть ниже.
* DES-EEE2: Исходный текст шифруется три раза, однако ключи на первом и последнем шаге одинаковые.
* DES-EDE2: Исходный текст шифруется, затем дешифруется (другим ключом), затем снова шифруется (ключом, используемым при первом шифровании).

На практике, самый используемый тип 3DES шифрования - DES-EDE3.

## ГОСТ 28147-89

ГОСТ 28147-89 представляет собой симметричный 64-битовый

блочный алгоритм с 256-битовым ключом.

Этот алгоритм криптографического преобразования данных

предназначен для аппаратной и программной реализации, удовлетворяет

криптографическим требованиям и не накладывает ограничений на степень

секретности защищаемой информации.

Для зашифровывания в режиме простой замене 64-битный блок [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) сначала разбивается на две половины: T0 = (A0, B0). На i-ом цикле используется подключ Ti:





Для генерации подключей исходный 256-битный ключ разбивается на восемь 32-битных чисел: K0,…K7.

Подключи X0…X23 являются циклическим повторением K0,…K7. Подключи X24…X31 являются K0,…K7.

Результатом выполнения всех 32 раундов алгоритма является 64-битный блок [шифртекста](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82): Tш = (А32, B32).

Расшифрование осуществляется по тому же алгоритму, что и зашифрование, с тем изменением, что инвертируется порядок подключей: X0,…X7 являются K0,…K7, а X8,…X31 являются циклическим повторением K7,…K0.

Во входных и выходных данных 32-битные числа представляются как [little endian](https://ru.wikipedia.org/wiki/Little_endian).

Функция  вычисляется следующим образом:

Ai и Xi складываются по модулю 232.

Результат разбивается на восемь 4-битовых подпоследовательностей, каждая из которых поступает на вход своего узла таблицы замен (в порядке возрастания старшинства битов), называемого ниже S-блоком. Общее количество S-блоков стандарта — восемь, то есть столько же, сколько и подпоследовательностей. Каждый S-блок представляет собой перестановку чисел от 0 до 15 (конкретный вид S-блоков в стандарте не определен). Первая 4-битная подпоследовательность попадает на вход первого S-блока, вторая — на вход второго и т. д.

Если узел S-блока выглядит так:

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

и на входе S-блока 0, то на выходе будет 1, если 4, то на выходе будет 5, если на входе 12, то на выходе 6 и т. д.

Выходы всех восьми S-блоков объединяются в 32-битное слово, затем всё слово циклически сдвигается влево (к старшим разрядам) на 11 битов.

Режим простой замены имеет следующие недостатки:

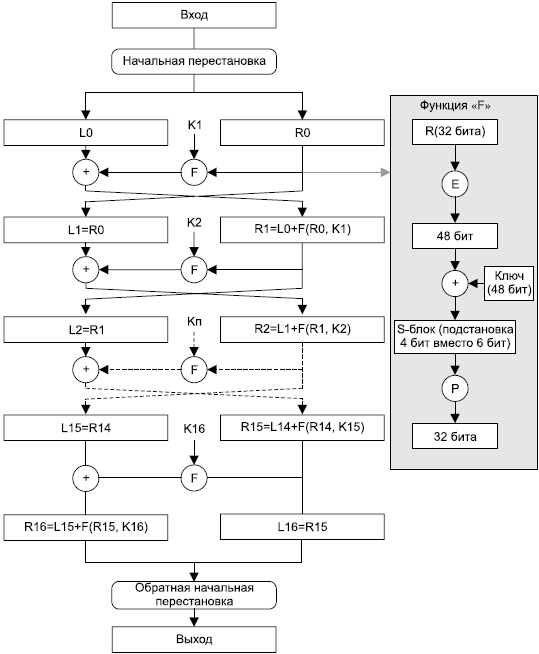
Может применяться только для шифрования открытых текстов с длиной, кратной 64 бит

При шифровании одинаковых блоков открытого текста получаются одинаковые блоки шифротекста, что может дать определенную информацию криптоаналитику.

Таким образом, применение ГОСТ 28147-89 в режиме простой замены желательно лишь для шифрования ключевых данных.

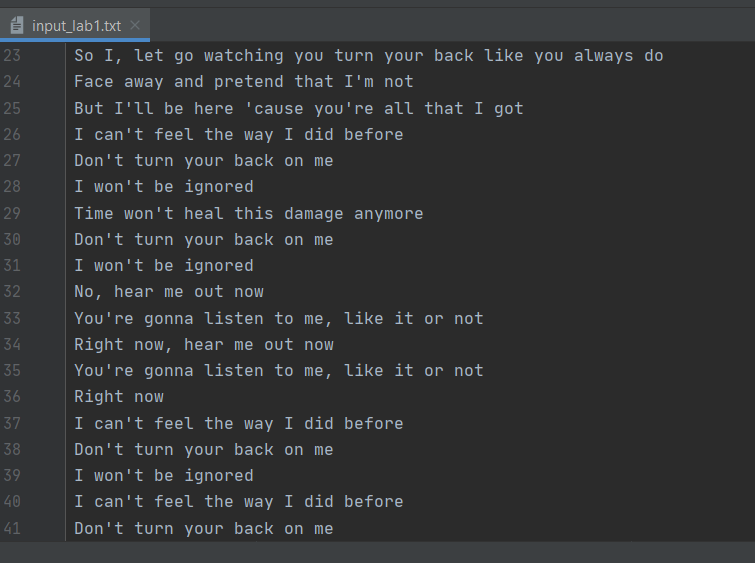
# Блок-схемы алгоритмов

Общая схема алгоритма *кодирования* для DES.



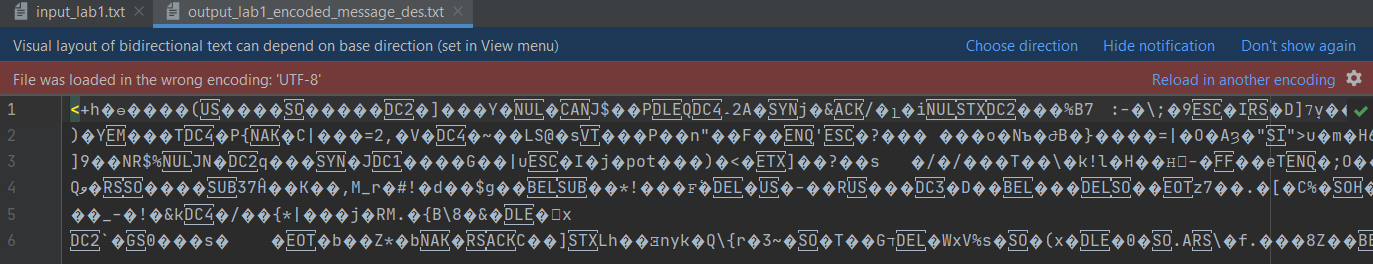
# Результаты выполнения программы

Файл с входными данными:

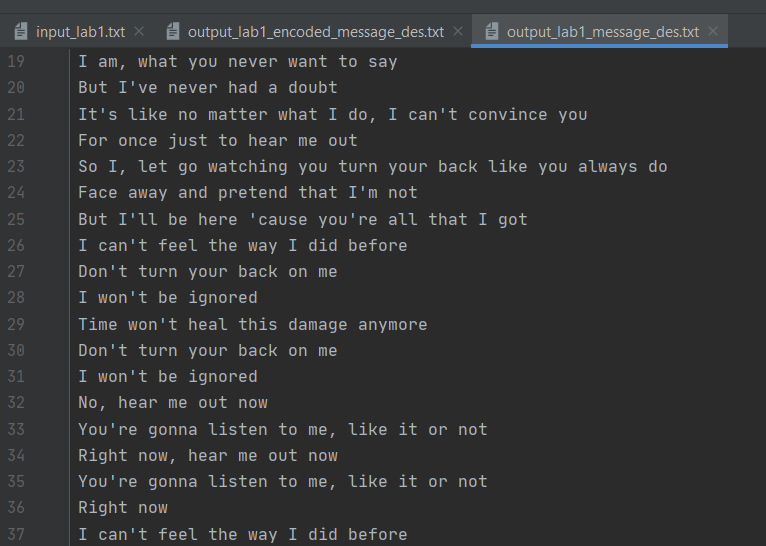


## 3.1 DES

Зашифрованное сообщение:

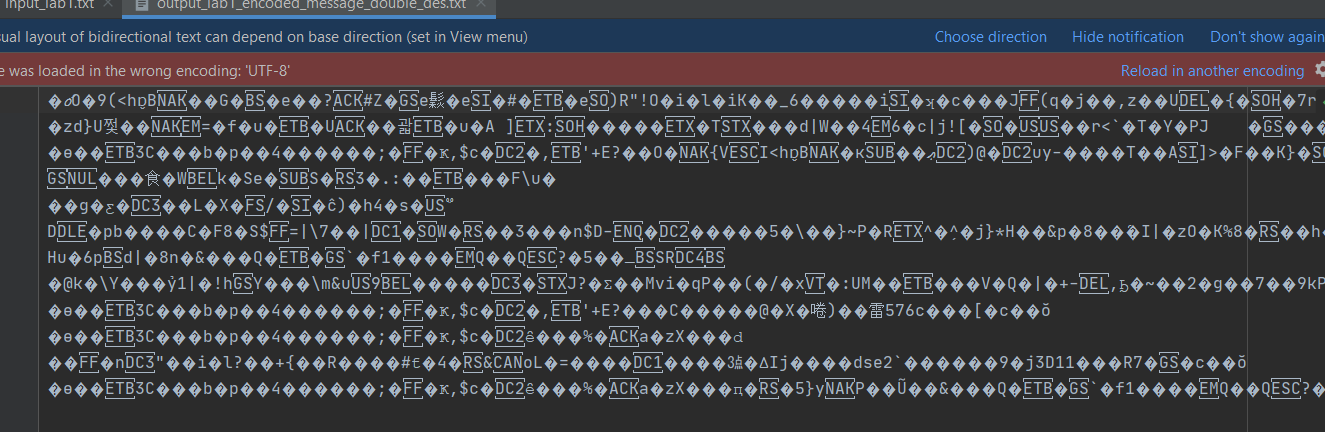


Расшифрованное сообщение:

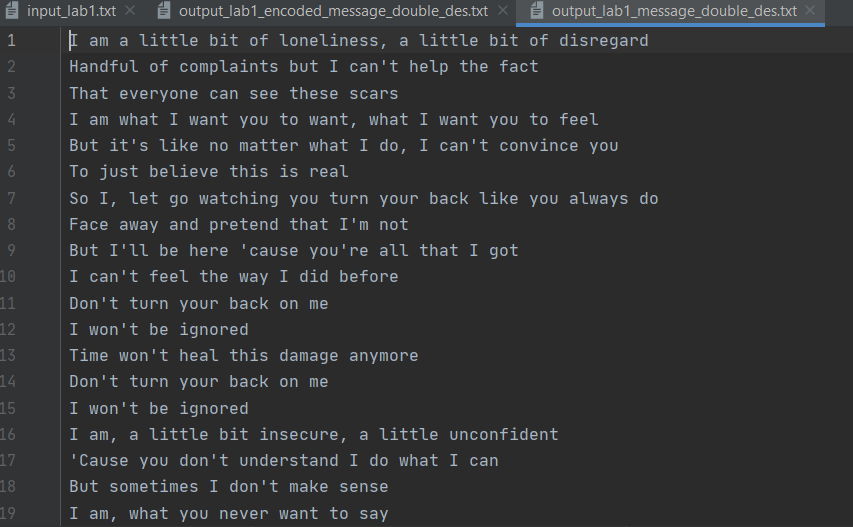


## 3.2 Double DES

Зашифрованное сообщение:

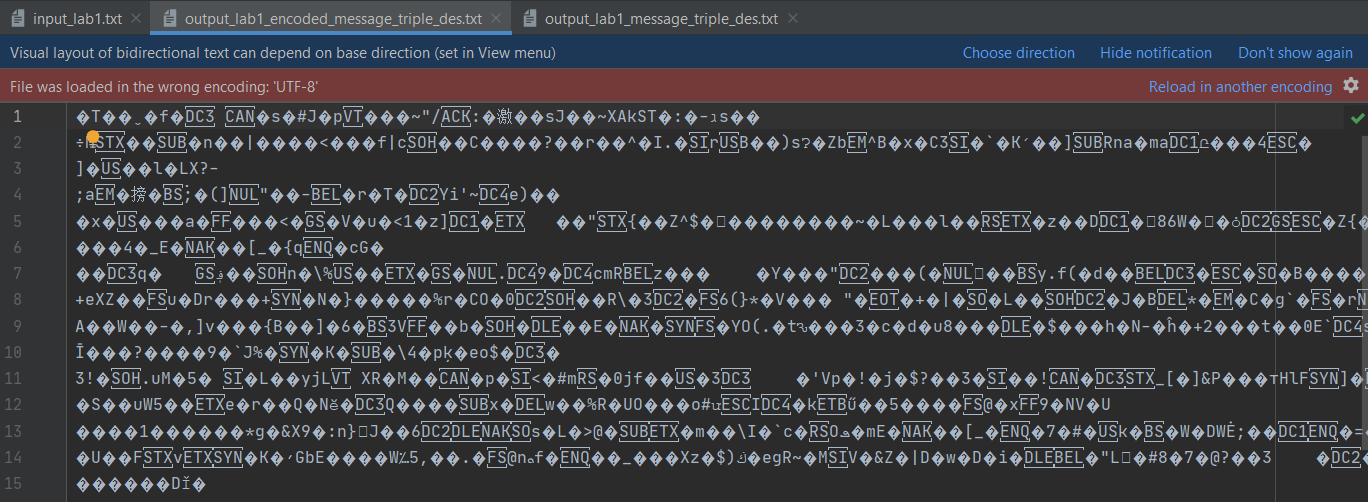


Расшифрованное сообщение:

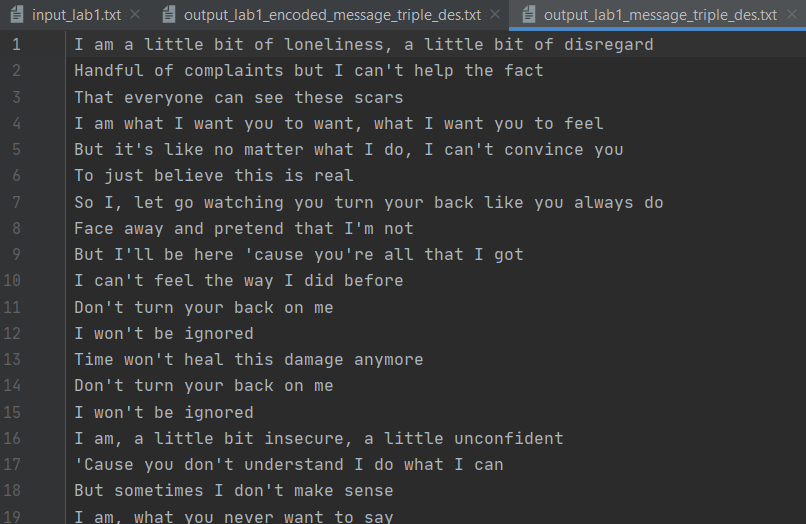


## 3.3 Triple DES

Зашифрованное сообщение:

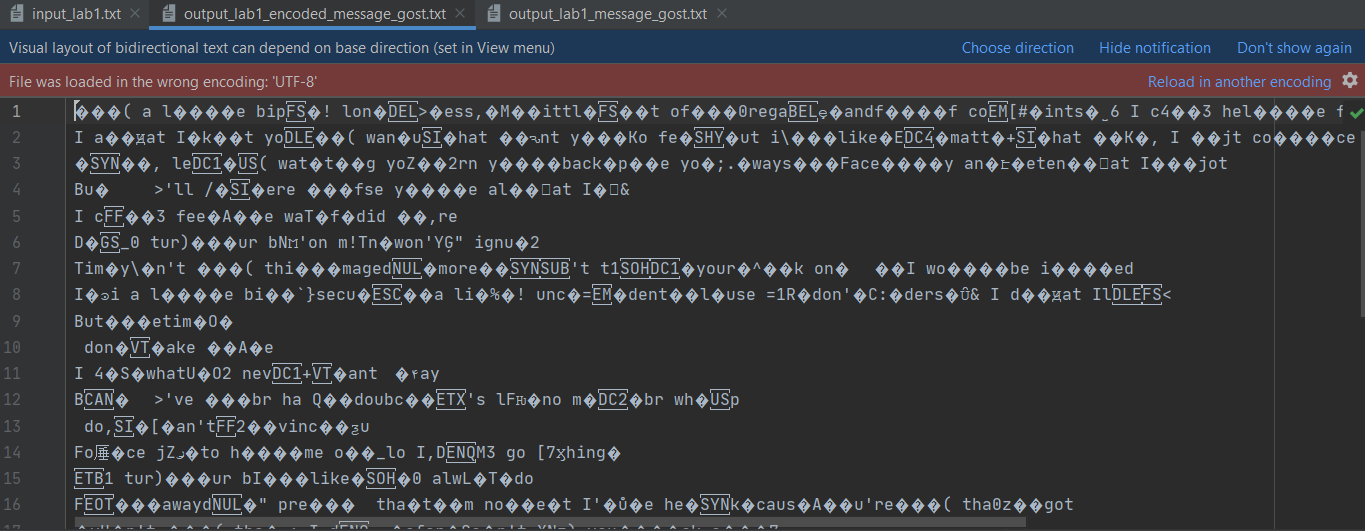


Расшифрованное сообщение:

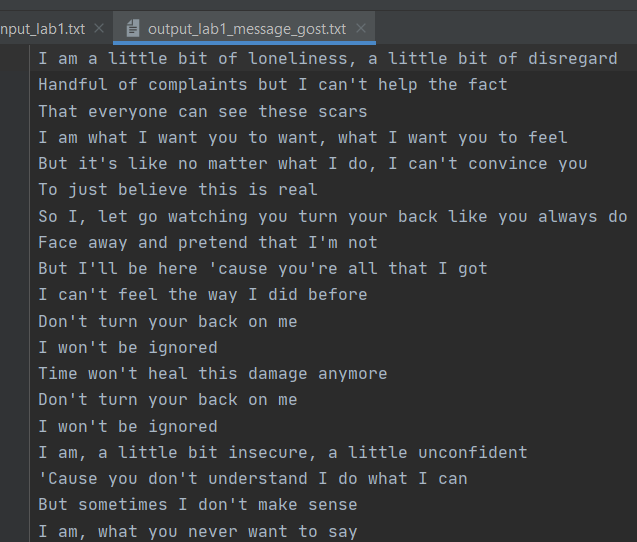


## 3.4 Гост 28147-89

Зашифрованное сообщение:



Расшифрованное сообщение:



# Исходный код

import typing as tp  
import numpy as np  
import dataclasses  
  
from bitarray import bitarray  
from bitarray.util import ba2int, int2ba  
  
from block\_cipher import BlockCipher  
from functions import left\_cycle\_shift  
  
  
@dataclasses.dataclass  
class DesCipherConstParameters:  
 ip\_permutation: tp.List[int]  
 e\_permutation: tp.List[int]  
 s\_boxes: tp.List[tp.List[tp.List[int]]]  
 p\_permutation: tp.List[int]  
 pc\_1\_permutation: tp.List[int]  
 pc\_2\_permutation: tp.List[int]  
  
  
class Des(BlockCipher, message\_block\_length=64):  
 def \_\_init\_\_(self, const\_parameters: DesCipherConstParameters, key: bitarray) -> None:  
 super().\_\_init\_\_()  
  
 self.\_ip\_permutation = const\_parameters.ip\_permutation  
 self.\_inverse\_ip\_permutation = self.\_inverse\_permutation(const\_parameters.ip\_permutation)  
  
 self.\_e\_permutation = const\_parameters.e\_permutation  
 self.\_s\_boxes = const\_parameters.s\_boxes  
 self.\_p\_permutation = const\_parameters.p\_permutation  
  
 self.\_pc\_1\_permutation = const\_parameters.pc\_1\_permutation  
 self.\_pc\_2\_permutation = const\_parameters.pc\_2\_permutation  
  
 self.\_key = key  
  
 @staticmethod  
 def generate\_parameters() -> DesCipherConstParameters:  
 ip\_permutation = np.random.permutation(64).tolist()  
 e\_permutation = np.random.randint(0, 32, 48).tolist()  
 s\_boxes = []  
  
 for \_ in range(8):  
 table = []  
 for \_ in range(4):  
 table.append(np.random.permutation(16).tolist())  
 s\_boxes.append(table)  
  
 p\_permutation = np.random.permutation(32).tolist()  
 pc\_1\_permutation = np.random.permutation(56).tolist()  
 pc\_2\_permutation = np.random.permutation(48).tolist()  
  
 return DesCipherConstParameters(ip\_permutation, e\_permutation, s\_boxes, p\_permutation, pc\_1\_permutation, ## noqa  
 pc\_2\_permutation) ## noqa  
  
 @staticmethod  
 def \_inverse\_permutation(permutation: tp.List[int]) -> tp.List[int]:  
 inverse\_permutation = [0] \* len(permutation)  
 for index, element in enumerate(permutation):  
 inverse\_permutation[element] = index  
 return inverse\_permutation  
  
 @staticmethod  
 def \_make\_permutation(block: bitarray, permutation: tp.List[int]) -> bitarray:  
 result\_block = bitarray(len(block))  
 for to\_position, from\_position in enumerate(permutation):  
 result\_block[to\_position] = block[from\_position]  
 return result\_block  
  
 @staticmethod  
 def \_make\_extension(block: bitarray, extension\_permutation: tp.List[int]) -> bitarray:  
 result\_block = bitarray(len(extension\_permutation))  
  
 for to\_position, from\_position in enumerate(extension\_permutation):  
 result\_block[to\_position] = block[from\_position]  
 return result\_block  
  
 @staticmethod  
 def \_make\_s\_box\_conversion(block: bitarray, s\_boxes: tp.List[tp.List[tp.List[int]]]) -> bitarray:  
 result\_bitarray = bitarray()  
 for i in range(8):  
 current\_six\_bits = block[i \* 6: (i + 1) \* 6]  
 s\_box\_conversion\_result = int2ba(s\_boxes[i][ba2int(current\_six\_bits[:2])][ba2int(current\_six\_bits[2:])],  
 length=4)  
 result\_bitarray += s\_box\_conversion\_result  
 return result\_bitarray  
  
 def change\_key(self, key: bitarray) -> None:  
 self.\_key = key  
  
 def \_feistel\_function(self, block\_part: bitarray, key: bitarray) -> bitarray:  
 extended\_block = self.\_make\_extension(block\_part, self.\_e\_permutation)  
 block\_key\_xor = key ^ extended\_block  
 s\_box\_conversed\_block = self.\_make\_s\_box\_conversion(block\_key\_xor, self.\_s\_boxes)  
 return self.\_make\_permutation(s\_box\_conversed\_block, self.\_p\_permutation)  
  
 def \_encryption\_cycle(self, block\_left\_part: bitarray, block\_right\_part: bitarray,  
 key: bitarray) -> tp.Tuple[bitarray, bitarray]:  
 new\_block\_right\_part = block\_left\_part ^ self.\_feistel\_function(block\_right\_part, key)  
 return block\_right\_part, new\_block\_right\_part  
  
 def \_decryption\_cycle(self, block\_left\_part: bitarray, block\_right\_part: bitarray,  
 key: bitarray) -> tp.Tuple[bitarray, bitarray]:  
 new\_block\_left\_part = block\_right\_part ^ self.\_feistel\_function(block\_left\_part, key)  
 return new\_block\_left\_part, block\_left\_part  
  
 def \_make\_keys(self, left\_key\_part: bitarray, right\_key\_part: bitarray, round\_no: int) -> \  
 tp.Tuple[bitarray, bitarray, bitarray]:  
  
 new\_left\_key\_part: bitarray = left\_cycle\_shift(left\_key\_part, round\_no % 2 + 1)  
 new\_right\_key\_part: bitarray = left\_cycle\_shift(right\_key\_part, round\_no % 2 + 1)  
 feistel\_key = self.\_make\_extension(new\_left\_key\_part + new\_right\_key\_part, self.\_pc\_2\_permutation)  
  
 return new\_left\_key\_part, new\_right\_key\_part, feistel\_key  
  
 def \_encrypt\_block(self, block: bitarray) -> bitarray:  
 initial\_block\_permutation = self.\_make\_permutation(block, self.\_ip\_permutation)  
 block\_current\_left\_part = initial\_block\_permutation[:32]  
 block\_current\_right\_part = initial\_block\_permutation[32:]  
  
 initial\_key\_permutation = self.\_make\_permutation(self.\_key, self.\_pc\_1\_permutation)  
 key\_current\_left\_part = initial\_key\_permutation[:28]  
 key\_current\_right\_part = initial\_key\_permutation[28:]  
  
 for round\_no in range(16):  
 key\_current\_left\_part, key\_current\_right\_part, feistel\_key = self.\_make\_keys(key\_current\_left\_part,  
 key\_current\_right\_part,  
 round\_no)  
 block\_current\_left\_part, block\_current\_right\_part = self.\_encryption\_cycle(block\_current\_left\_part,  
 block\_current\_right\_part,  
 feistel\_key)  
 return self.\_make\_permutation(block\_current\_left\_part + block\_current\_right\_part,  
 self.\_inverse\_ip\_permutation)  
  
 def \_decrypt\_block(self, block: bitarray) -> bitarray:  
 initial\_key\_permutation = self.\_make\_permutation(self.\_key, self.\_pc\_1\_permutation)  
 key\_current\_left\_part = initial\_key\_permutation[:28]  
 key\_current\_right\_part = initial\_key\_permutation[28:]  
 inverse\_feistel\_keys = []  
 initial\_block\_permutation = self.\_make\_permutation(block, self.\_ip\_permutation)  
 block\_current\_left\_part = initial\_block\_permutation[:32]  
 block\_current\_right\_part = initial\_block\_permutation[32:]  
  
 for round\_no in range(16):  
 key\_current\_left\_part, key\_current\_right\_part, feistel\_key = self.\_make\_keys(key\_current\_left\_part,  
 key\_current\_right\_part,  
 round\_no)  
 inverse\_feistel\_keys.append(feistel\_key)  
  
 for feistel\_key in reversed(inverse\_feistel\_keys):  
 block\_current\_left\_part, block\_current\_right\_part = self.\_decryption\_cycle(block\_current\_left\_part,  
 block\_current\_right\_part,  
 feistel\_key)  
 return self.\_make\_permutation(block\_current\_left\_part + block\_current\_right\_part,  
 self.\_inverse\_ip\_permutation)  
  
  
def main() -> None:  
 key = bitarray(56)  
 encoder = Des(Des.generate\_parameters(), key)  
 with open('input\_files/input\_lab1.txt', 'r') as f:  
 message = bitarray()  
 message.frombytes(f.read().encode('UTF-8'))  
 encoded\_message = encoder.encrypt(message)  
 with open('output\_files/output\_lab1\_encoded\_message\_des.txt', 'wb') as f:  
 encoded\_message.tofile(f)  
 decoded\_message = encoder.decrypt(encoded\_message)  
 with open('output\_files/output\_lab1\_message\_des.txt', 'wb') as f:  
 decoded\_message.tofile(f)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

from bitarray import bitarray  
  
from lab1\_des import Des, DesCipherConstParameters  
from block\_cipher import BlockCipher  
  
  
class DoubleDes(BlockCipher, message\_block\_length=64):  
 def \_\_init\_\_(self, const\_parameters: DesCipherConstParameters, first\_key: bitarray, second\_key: bitarray) -> None:  
 self.\_des\_cipher = Des(const\_parameters, first\_key)  
 self.\_first\_key = first\_key  
 self.\_second\_key = second\_key  
  
 def \_decrypt\_block(self, message\_block: bitarray) -> bitarray:  
 self.\_des\_cipher.change\_key(self.\_first\_key)  
 firstly\_encrypted\_message\_block = self.\_des\_cipher.\_encrypt\_block(message\_block) # noqa  
  
 self.\_des\_cipher.change\_key(self.\_second\_key)  
 secondly\_encrypted\_message\_block = self.\_des\_cipher.\_encrypt\_block(firstly\_encrypted\_message\_block) # noqa  
 return secondly\_encrypted\_message\_block  
  
 def \_encrypt\_block(self, message\_block: bitarray) -> bitarray:  
 self.\_des\_cipher.change\_key(self.\_second\_key)  
 firstly\_decrypted\_message\_block = self.\_des\_cipher.decrypt(message\_block)  
  
 self.\_des\_cipher.change\_key(self.\_first\_key)  
 secondly\_decrypted\_message\_block = self.\_des\_cipher.decrypt(firstly\_decrypted\_message\_block)  
 return secondly\_decrypted\_message\_block  
  
  
def main() -> None:  
 keys = bitarray(56), bitarray(56)  
 encoder = DoubleDes(Des.generate\_parameters(), \*keys)  
 with open('input\_files/input\_lab1.txt', 'r') as f:  
 message = bitarray()  
 message.frombytes(f.read().encode('UTF-8'))  
 encoded\_message = encoder.encrypt(message)  
 with open('output\_files/output\_lab1\_encoded\_message\_double\_des.txt', 'wb') as f:  
 encoded\_message.tofile(f)  
 decoded\_message = encoder.decrypt(encoded\_message)  
 with open('output\_files/output\_lab1\_message\_double\_des.txt', 'wb') as f:  
 decoded\_message.tofile(f)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

from bitarray import bitarray  
  
from lab1\_des import Des, DesCipherConstParameters  
from block\_cipher import BlockCipher  
  
  
class TripleDes(BlockCipher, message\_block\_length=64):  
 def \_\_init\_\_(self, const\_parameters: DesCipherConstParameters, first\_key: bitarray, second\_key: bitarray) -> None:  
 self.\_des\_cipher = Des(const\_parameters, first\_key)  
 self.\_first\_key = first\_key  
 self.\_second\_key = second\_key  
  
 def \_encrypt\_block(self, message\_block: bitarray) -> bitarray:  
 self.\_des\_cipher.change\_key(self.\_first\_key)  
 firstly\_encrypted\_message\_block = self.\_des\_cipher.\_encrypt\_block(message\_block)  
  
 self.\_des\_cipher.change\_key(self.\_second\_key)  
 firstly\_decrypted\_message\_block = self.\_des\_cipher.\_decrypt\_block(firstly\_encrypted\_message\_block)  
  
 self.\_des\_cipher.change\_key(self.\_first\_key)  
 secondly\_encrypted\_message\_block = self.\_des\_cipher.\_encrypt\_block(firstly\_decrypted\_message\_block)  
  
 return secondly\_encrypted\_message\_block  
  
 def \_decrypt\_block(self, message\_block: bitarray) -> bitarray:  
 self.\_des\_cipher.change\_key(self.\_first\_key)  
 firstly\_decrypted\_message\_block = self.\_des\_cipher.\_decrypt\_block(message\_block)  
  
 self.\_des\_cipher.change\_key(self.\_second\_key)  
 firstly\_encrypted\_message\_block = self.\_des\_cipher.\_encrypt\_block(firstly\_decrypted\_message\_block)  
  
 self.\_des\_cipher.change\_key(self.\_first\_key)  
 secondly\_decrypted\_message\_block = self.\_des\_cipher.\_decrypt\_block(firstly\_encrypted\_message\_block)  
  
 return secondly\_decrypted\_message\_block  
  
  
def main() -> None:  
 keys = bitarray(56), bitarray(56)  
 encoder = TripleDes(Des.generate\_parameters(), \*keys)  
 with open('input\_files/input\_lab1.txt', 'r') as f:  
 message = bitarray()  
 message.frombytes(f.read().encode('UTF-8'))  
 encoded\_message = encoder.encrypt(message)  
 with open('output\_files/output\_lab1\_encoded\_message\_triple\_des.txt', 'wb') as f:  
 encoded\_message.tofile(f)  
 decoded\_message = encoder.decrypt(encoded\_message)  
 with open('output\_files/output\_lab1\_message\_triple\_des.txt', 'wb') as f:  
 decoded\_message.tofile(f)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

import typing as tp  
  
from bitarray import bitarray  
from bitarray.util import int2ba, ba2int  
  
from block\_cipher import BlockCipher  
from functions import left\_cycle\_shift  
  
  
class Gost2814789(BlockCipher, message\_block\_length=64):  
 \_default\_s\_block = [[10, 4, 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]]  
  
 def \_\_init\_\_(self, key: bitarray, s\_block: tp.Optional[tp.List[tp.List[int]]] = None) -> None:  
 self.\_key = key  
 self.\_s\_block = self.\_default\_s\_block if s\_block is None else s\_block  
  
 def \_compute\_substitution(self, message\_block: bitarray) -> bitarray:  
 substituted\_message\_block = bitarray()  
 for i in range(8):  
 value = ba2int(message\_block[i \* 4: (i + 1) \* 4])  
 substituted\_message\_block.extend(int2ba(self.\_s\_block[i][value], length=4))  
 return substituted\_message\_block  
  
 def \_compute\_round(self, message\_block: bitarray, subkey: bitarray) -> bitarray:  
 right\_part\_sum = int2ba((ba2int(message\_block[32:]) + ba2int(subkey)) % (2 \*\* 32), length=32)  
 next\_message\_block = left\_cycle\_shift(self.\_compute\_substitution(right\_part\_sum), 11) ^ message\_block[:32]  
 next\_message\_block.extend(message\_block[32:])  
 return next\_message\_block  
  
 def \_encrypt\_block(self, message\_block: bitarray) -> bitarray:  
 encrypted\_block = message\_block.copy()  
 for round\_series\_no in range(7):  
 for round\_no in range(8):  
 if round\_series\_no == 3:  
 subkey\_no = round\_no  
 else:  
 subkey\_no = round\_no  
 encrypted\_block = self.\_compute\_round(encrypted\_block.copy(), self.\_key[subkey\_no \* 32: (subkey\_no + 1) \* 32])  
 return encrypted\_block  
  
 def \_decrypt\_block(self, message\_block: bitarray) -> bitarray:  
 decrypted\_block = message\_block.copy()  
 for round\_series\_no in range(7):  
 for round\_no in range(8):  
 if round\_series\_no == 0:  
 subkey\_no = 7 - round\_no  
 else:  
 subkey\_no = 7 - round\_no  
 decrypted\_block = self.\_compute\_round(decrypted\_block.copy(), self.\_key[subkey\_no \* 32: (subkey\_no + 1) \* 32])  
 return decrypted\_block  
  
  
def main() -> None:  
 key = bitarray(256)  
 encoder = Gost2814789(key)  
 with open('input\_files/input\_lab1.txt', 'r') as f:  
 message = bitarray()  
 message.frombytes(f.read().encode('UTF-8'))  
 encoded\_message = encoder.encrypt(message)  
 with open('output\_files/output\_lab1\_encoded\_message\_gost.txt', 'wb') as f:  
 encoded\_message.tofile(f)  
 decoded\_message = encoder.decrypt(encoded\_message)  
 with open('output\_files/output\_lab1\_message\_gost.txt', 'wb') as f:  
 decoded\_message.tofile(f)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Вывод

В ходе лабораторной работы были реализованы блочные шифры DES (двойной и тройной DES) и ГОСТ 28147-89.