Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №1

По теме «Компьютерная реализация блочных шифров на примере DES и ГОСТ»

Выполнил:

студент гр. 853503

Климович А. А.

Проверил:

Протько М.И

Оглавление

[Цель работы 3](#_Toc83033966)

[Описание реализуемых алгоритмов 4](#_Toc83033967)

[1. Алгоритм шифрования DES 4](#_Toc83033968)

[2. Алгоритм шифрования ГОСТ 28147-89 7](#_Toc83033969)

[Результаты реализации алгоритмов шифрования 10](#_Toc83033970)

[Вывод 11](#_Toc83033971)

[Приложение 1. Исходный код алгоритмов 12](#_Toc83033972)

[Алгоритм шифрования DES 12](#_Toc83033973)

[Алгоритм шифрования ГОСТ 28147-89 17](#_Toc83033974)

# Цель работы

Целью данной лабораторной работы является реализация программных средств шифрования и дешифрования текстовых файлов при помощи алгоритмов DES и ГОСТ 28147-89.

# Описание реализуемых алгоритмов

## Алгоритм шифрования DES

DES является классической сетью Фейстеля с двумя ветвями.

Данные шифруются 64-битными блоками, используя 56-битный ключ.

Алгоритм преобразует за несколько раундов 64-битный вход в 64-битный выход.

Процесс шифрования состоит из четырех этапов.

Выполняется начальная перестановка (IP) 64-битного исходного текста (забеливание), во время которой биты переупорядочиваются в соответствии со стандартной таблицей.

Этап состоит из 16 раундов одной и той же функции, которая использует операции сдвига и подстановки.

Левая и правая половины выхода последней (16-й) итерации меняются местами.

Выполняется перестановка IP-1 результата, полученного на третьем этапе.

Перестановка IP-1 инверсна начальной перестановке.

**Начальная перестановка**

Начальная перестановка и ее инверсия определяются стандартной таблицей.

Если **М** - это произвольные 64 бита, то

**X = IP (M)** - переставленные 64 бита.

Изображение выглядит как текст, стол

Автоматически созданное описание

*Рисунок 1. Вспомогательная таблица начальной перестановки*

Если применить обратную функцию перестановки

**Y = IP-1 (X) = IP-1 (IP(M)),**

то получится первоначальная последовательность бит.

Изображение выглядит как стол

Автоматически созданное описание

*Рисунок 2. Вспомогательная таблица обратной перестановки*

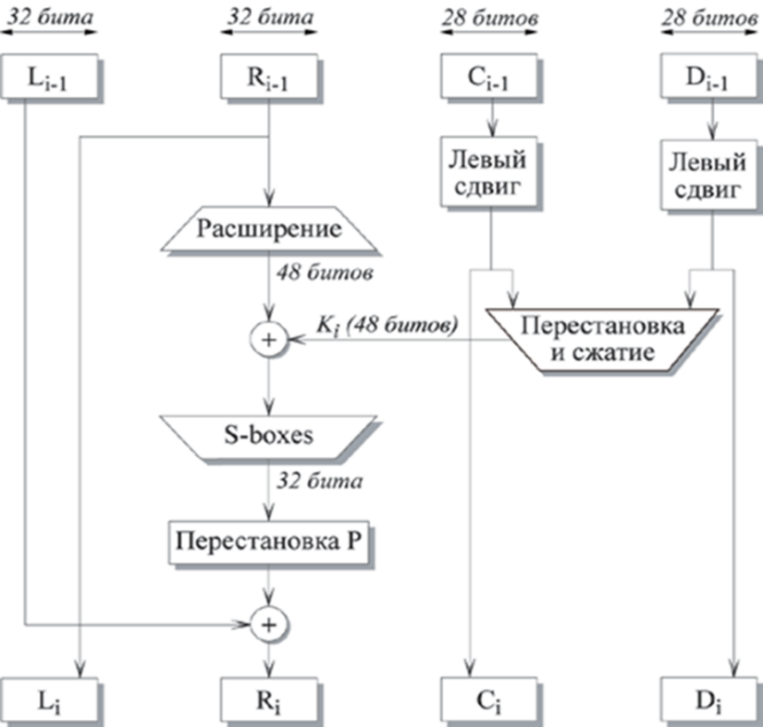
**Последовательность преобразований отдельного раунда**

Рисунок . Блок-схема одного раунда шифрования алгоритма DES

Каждую итерацию можно описать следующим образом:

Li = Ri-1

Ri = Li-1⊕F(Ri-1, Ki)

**Процедура преобразования ключа**

64-битовый ключ DES уменьшается до 56-битового ключа отбрасыванием каждого восьмого бита. Эти биты используются только для контроля четности, позволяя проверять правильность ключа.

После извлечения 56-битного ключа для каждого из 16 циклов генерируется новый 48-битный подключ.

Эти подключи, К2, определяются следующим образом:

56-битный ключ делится на две 28-битные половины.

Половины циклически сдвигаются влево на один или два бита в зависимости от номера цикла.

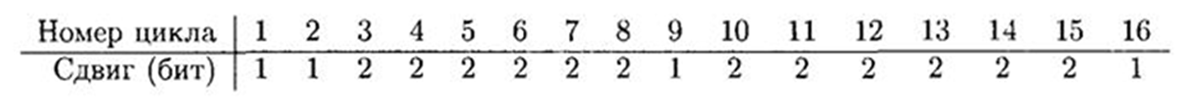


Рисунок . Последовательность циклических сдвигов

**DES-подстановка при помощи S-блоков**

После объединения сжатого блока с расширенным блоком с помощью XOR над 48-битным результатом выполняется операция подстановки.

Подстановки производятся в восьми блоках подстановки, или S-блоках (от substitution).

У каждого S-блока есть 6-битный вход и 4-битный выход, всего используется восемь различных S-блоков.

Рисунок . Таблица блоков подстановки, использованная в алгоритме

48 битов делятся на восемь 6-битных подблоков.

Каждый отдельный подблок обрабатывается отдельным S-блоком: первый подблок - первым S-блоком, второй - вторым S-блоком и так далее.

Каждый S-блок представляет собой таблицу из четырех строк и шестнадцати столбцов. Каждый элемент в блоке является 4-битным числом. По шести входным битам S-блока определяется, под какими номерами столбцов и строк следует искать выходное значение.

**Дешифрование**

Процесс дешифрования аналогичен процессу шифрования.

На входе алгоритма используется зашифрованный текст, но ключи *Ki* используются в обратной последовательности.

**Двойной и тройной DES**

Простейший способ увеличить длину ключа состоит в повторном применении *DES* с двумя разными ключами – в этом и состоит смысл двойного алгоритма DES.

В этом случае длина ключа равна 56 \* 2 = 112 бит.

Для повышения устойчивости алгоритма перед атакой под названием «встреча посередине» используется тройное последовательное шифрование тремя ключами общей длиной 168 бит.

**Итоговая блок-схема алгоритма**

Приведём итоговую блок-схему алгоритма шифрования DES за исключением процедуры преобразования ключа

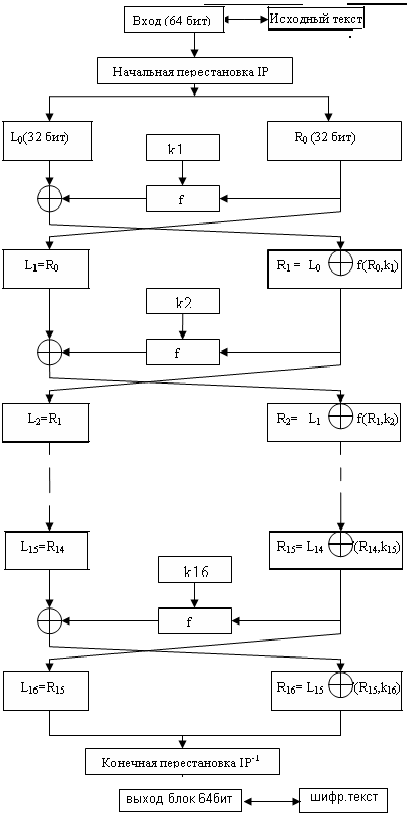


Рисунок . Блок-схема алгоритма DES

## Алгоритм шифрования ГОСТ 28147-89

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

**Схема алгоритма**

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

Изображение выглядит как текст

Автоматически созданное описаниеСначала правая половина блока и подключ раунда складываются по модулю 232.

Затем производится поблочная подстановка.

Рисунок 7. Таблица замен алгоритма ГОСТ 28147-89

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.

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 дешифрования**

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 ниже.

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.

Раунд 1 2 3 4 5 6 7 8

Подключ K0 K1 K2 K3 K4 K5 K6 K7

Раунд 9 10 11 12 13 14 15 16

Подключ K0 K1 K2 K3 K4 K5 K6 K7

Раунд 17 18 19 20 21 22 23 24

Подключ K0 K1 K2 K3 K4 K5 K6 K7

Раунд 25 26 27 28 29 30 31 32

Подключ K7 K6 K5 K4 K3 K2 K1 K0

Таблица 1. Последовательность использования подключей при шифровании

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

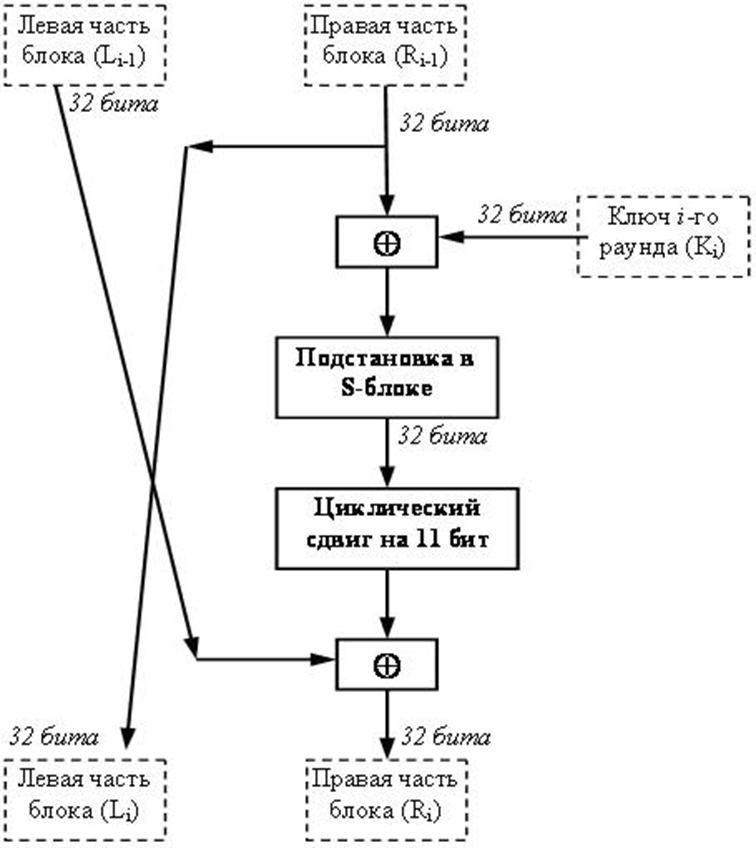


Рисунок 8. Блок-схема алгоритма шифрования ГОСТ 28147-89

# Результаты реализации алгоритмов шифрования

В ходе выполнения работы мной были реализованы двойной и тройной алгоритмы шифрования DES, а также алгоритм шифрования ГОСТ 28147-89.

На скриншоте ниже представлены результаты работы всех трёх алгоритмов.

Изображение выглядит как текст

Автоматически созданное описание

Как можно заметить, по итогам работы всех алгоритмов первоначальная строка совпадает с декодированной в результате цикла кодирования/декодирования, кроме того, промежуточные результаты при декодировании на второй ступени алгоритма DES совпадают с закодированной первым ключом изначальной строкой. Из этого можно сделать вывод о том, что реализации алгоритмов корректны.

# Вывод

В ходе выполнения лабораторной работы мною были реализованы программные средств шифрования и дешифрования текстовых файлов при помощи алгоритмов DES и ГОСТ 28147-89.

Во время подготовки к выполнению задания я изучил теоретический материал по темам симметричных шифров и шифров DES и ГОСТ 28147-89 – в частности, тем самым расширив и подкрепив свои знания в области методов защиты информации.

# Приложение 1. Исходный код алгоритмов

## Алгоритм шифрования DES

from collections import Counter

class DES:

    def \_\_init\_\_(self):

        self.IP = [

            58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,

            62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,

            57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,

            61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7

        ]

        self.E = [

            32, 1, 2, 3, 4, 5,

            4, 5, 6, 7, 8, 9,

            8, 9, 10, 11, 12, 13,

            12, 13, 14, 15, 16, 17,

            16, 17, 18, 19, 20, 21,

            20, 21, 22, 23, 24, 25,

            24, 25, 26, 27, 28, 29,

            28, 29, 30, 31, 32, 1

        ]

        self.S = [

            [

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

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

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

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

            ], [

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

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

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

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

            ], [

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

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

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

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

            ], [

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

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

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

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

            ], [

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

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

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

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

            ], [

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

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

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

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

            ], [

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

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

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

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

            ], [

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

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

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

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

            ]

        ]

        self.P = [

            16, 7, 20, 21, 29, 12, 28, 17,

            1, 15, 23, 26, 5, 18, 31, 10,

            2, 8, 24, 14, 32, 27, 3, 9,

            19, 13, 30, 6, 22, 11, 4, 25

        ]

        self.C = [

            57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,

            10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,

        ]

        self.D = [

            63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,

            14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4

        ]

        self.shift = [

            1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

        ]

        self.KP = [

            14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4,

            26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40,

            51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32

        ]

        self.final\_IP = [

            40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,

            38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,

            36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,

            34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25

        ]

    @staticmethod

    def add\_zeros(data, k):

        if len(data) <= k:

            zeros\_size = k - len(data)

            data2 = [0 for i in range(zeros\_size)] + data

            return data2

        elif len(data) > k:

            raise RuntimeError('{}, {}'.format(len(data), k))

    def encrypt(self, data, key):

        if type(data) != list:

            raise RuntimeError('Not list')

        data = [1] + data

        n = len(data)

        m = 0

        while m < n:

            m += 64

        data = self.add\_zeros(data, m)

        res = []

        for i in range(0, m, 64):

            res += self.encrypt\_64(data[i:i+64], key)

        return res

    def encrypt\_64(self, data, key):

        """

        :param data: 64 bit binary list //

        :param key: 56 bit key

        :return: encrypted 64 bit list

        """

        data = self.add\_zeros(data, 64)

        key = self.add\_zeros(key, 56)

        data\_IP = [0 for i in range(64)]

        for i in range(64):

            data\_IP[i] = data[self.IP[i] - 1]

        l\_i = data\_IP[:32]

        r\_i = data\_IP[32:]

        for i in range(16):

            k\_i = self.generate\_k48(key, i)

            l\_i, r\_i = self.feistel\_transform(l\_i, r\_i, k\_i)

        data\_IP = l\_i + r\_i

        ans = [0 for i in range(64)]

        for i in range(64):

            ans[i] = data\_IP[self.final\_IP[i] - 1]

        return ans

    def generate\_k48(self, k\_56, iteration):

        k\_64 = []

        for i in range(8):

            k\_64 += k\_56[i \* 7:i \* 7 + 7]

            if Counter(k\_56[i \* 7:i \* 7 + 7])[1] % 2 == 0:

                k\_64 += [1]

            else:

                k\_64 += [0]

        cd\_56 = [0 for i in range(56)]

        for i in range(28):

            cd\_56[i] = k\_64[self.C[i] - 1]

        j = 0

        for i in range(28, 56):

            cd\_56[i] = k\_64[self.D[j] - 1]

            j += 1

        k\_48 = [0 for i in range(48)]

        for i in range(48):

            k\_48[i] = cd\_56[self.KP[i] - 1]

        self.C = self.C[self.shift[iteration]:] + self.C[:self.shift[iteration]]

        self.D = self.D[self.shift[iteration]:] + self.D[:self.shift[iteration]]

        return k\_48

    def feistel\_transform(self, l\_prev, r\_prev, k\_i):

        l\_i = r\_prev

        r\_i = [0 for i in range(32)]

        frk = self.f(r\_prev, k\_i)

        for i in range(32):

            r\_i[i] = l\_prev[i] ^ frk[i]

        return l\_i, r\_i

    def f(self, r\_prev, k\_i):

        r\_48 = [0 for i in range(48)]

        for i in range(48):

            r\_48[i] = r\_prev[self.E[i] - 1]

        for i in range(48):

            r\_48[i] = r\_48[i] ^ k\_i[i]

        b\_32 = []

        for i in range(8):

            b\_i = r\_48[i \* 6:i \* 6 + 6]

            a = 2 \* b\_i[0] + b\_i[-1]

            b = 8 \* b\_i[1] + 4 \* b\_i[2] + 2 \* b\_i[3] + b\_i[4]

            b\_4 = bin(self.S[i][a \* 16 + b])[2:]

            if len(b\_4) < 4:

                b\_4 = '0' \* (4 - len(b\_4)) + b\_4

            b\_32 += [int(i) for i in b\_4]

        ans = [0 for i in range(32)]

        for i in range(32):

            ans[i] = b\_32[self.P[i] - 1]

        return ans

    def decrypt(self, data, key):

        if type(data) != list:

            raise RuntimeError('Not a list')

        if len(data) % 64:

            raise RuntimeError('Wrong size')

        n = len(data)

        res = []

        for i in range(0, n, 64):

            res += self.decrypt\_64(data[i:i+64], key)

        while res[0] != 1:

            res = res[1:]

        return res[1:]

    def decrypt\_64(self, data, key):

        """

        :param data: 64 bit binary string // in a list??

        :param key: 56 bit key

        :return: encrypted 64 bit string

        """

        data = self.add\_zeros(data, 64)

        key = self.add\_zeros(key, 56)

        data\_IP = [0 for i in range(64)]

        for i in range(64):

            data\_IP[i] = data[self.IP[i] - 1]

        l\_i = data\_IP[:32]

        r\_i = data\_IP[32:]

        keys = []

        for i in range(16):

            k\_i = self.generate\_k48(key, i)

            keys.append(k\_i)

        for i in reversed(range(16)):

            l\_i, r\_i = self.feistel\_transform\_dec(l\_i, r\_i, keys[i])

        data\_IP = l\_i + r\_i

        ans = [0 for i in range(64)]

        for i in range(64):

            ans[i] = data\_IP[self.final\_IP[i] - 1]

        return ans

    def feistel\_transform\_dec(self, l\_prev, r\_prev, k\_i):

        r\_i = l\_prev

        l\_i = [0 for i in range(32)]

        flk = self.f(l\_prev, k\_i)

        for i in range(32):

            l\_i[i] = r\_prev[i] ^ flk[i]

        return l\_i, r\_i

## Алгоритм шифрования ГОСТ 28147-89

class GOST:

    def \_\_init\_\_(self):

        self.S = [

            [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]

        ]

    @staticmethod

    def add\_zeros(data, k):

        if len(data) <= k:

            zeros\_size = k - len(data)

            data2 = [0 for i in range(zeros\_size)] + data

            return data2

        elif len(data) > k:

            raise RuntimeError('{}, {}'.format(len(data), k))

    @staticmethod

    def get\_8\_keys(key):

        if len(key) != 256:

            raise RuntimeError('Key size is {} not 256'.format(len(key)))

        keys = []

        for i in range(0, 256, 32):

            keys.append(key[i:i+32])

        return keys

    def encrypt(self, data, key):

        if type(data) != list:

            raise RuntimeError('Not list')

        key = self.add\_zeros(key, 256)

        keys = self.get\_8\_keys(key)

        data = [1] + data

        n = len(data)

        m = 0

        while m < n:

            m += 64

        data = self.add\_zeros(data, m)

        res = []

        for i in range(0, m, 64):

            res += self.encrypt\_64(data[i:i+64], keys)

        return res

    @staticmethod

    def get\_x(keys):

        x = []

        for i in range(24):

            x.append(keys[i % 8])

        for i in range(8):

            x.append(keys[7 - i])

        if len(x) != 32:

            raise RuntimeError('invalid x size')

        return x

    @staticmethod

    def get\_x\_dec(keys):

        x = []

        for i in range(8):

            x.append(keys[i])

        for i in range(24):

            x.append(keys[(7 - i) % 8])

        if len(x) != 32:

            raise RuntimeError('invalid x size')

        return x

    def encrypt\_64(self, data, keys):

        """

        :param data: 64 bit binary list //

        :param keys: list of size 8 with bit keys

        :return: encrypted 64 bit list

        """

        a\_i = data[:32]

        b\_i = data[32:]

        x = self.get\_x(keys)

        for i in range(32):

            k\_i = x[i]

            a\_i, b\_i = self.transform(a\_i, b\_i, k\_i)

        return a\_i + b\_i

    def transform(self, l\_prev, r\_prev, k\_i):

        r\_i = l\_prev

        l\_i = [0 for \_ in range(32)]

        frk = self.f(l\_prev, k\_i)

        for i in range(32):

            l\_i[i] = r\_prev[i] ^ frk[i]

        return l\_i, r\_i

    def f(self, l, k\_i):

        l\_int = int("".join(str(i) for i in l), 2)

        k\_int = int("".join(str(i) for i in k\_i), 2)

        l\_k\_mod = (l\_int + k\_int) % (2 \*\* 32)

        res = self.add\_zeros([int(i) for i in "{0:b}".format(l\_k\_mod)], 32)

        seq = []

        for i in range(0, 32, 4):

            seq.append(res[i:i+4])

        res = []

        for i in range(8):

            s\_int = int("".join(str(i) for i in seq[i]), 2)

            s\_int = self.S[i][s\_int]

            seq[i] = self.add\_zeros([int(i) for i in "{0:b}".format(s\_int)], 4)

            res += seq[i]

        res = res[11:] + res[:11]

        if len(res) != 32:

            raise RuntimeError('res in f should have len 32')

        return res

    def decrypt(self, data, key):

        if type(data) != list:

            raise RuntimeError('Not a list')

        if len(data) % 64:

            raise RuntimeError('Wrong size')

        key = self.add\_zeros(key, 256)

        keys = self.get\_8\_keys(key)

        n = len(data)

        res = []

        for i in range(0, n, 64):

            res += self.decrypt\_64(data[i:i+64], keys)

        while res[0] != 1:

            res = res[1:]

        return res[1:]

    def decrypt\_64(self, data, keys):

        a\_i = data[:32]

        b\_i = data[32:]

        x = self.get\_x\_dec(keys)

        for i in range(32):

            k\_i = x[i]

            b\_i, a\_i = self.transform(b\_i, a\_i, k\_i)

        return a\_i + b\_i