Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №9**

**Дисциплина: Криптографические протоколы**

Работу выполнила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д. Н. Баева

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Крамаренко

**Цель работы:** реализовать программный продукт прямого преобразования DES для введенных двоичных последовательностей открытого текста и ключа. Для реализации криптоалгоритмов запрещено пользоваться встроенными библиотеками используемых языков.

**Ход работы:**

**Алгоритм DES (Data Encryption Standard)** — это симметричный блочный шифр, разработанный фирмой IBM и утвержденный правительством США в 1977 году как официальный стандарт. Вот основные аспекты алгоритма DES:

1. **Основные характеристики**:
   * Размер блока данных: 64 бита.
   * Длина ключа: 56 бит (плюс 8 бит для проверки).
   * Количество раундов: 16.
   * Основан на сети Фейстеля.
2. **Принцип работы**:
   * DES использует комбинацию нелинейных (S-блоки) и линейных (перестановки) преобразований.
   * Входные данные разбиваются на блоки по 64 бита.
   * Ключ также разбивается на подключи для каждого раунда.
3. **Сеть Фейстеля**:
   * DES использует сеть Фейстеля, где блок данных разделяется на две половины: L и R.
   * В каждом раунде выполняются следующие действия:
     + Расширение R.
     + XOR с ключом.
     + Замена с использованием S-блоков.
     + Перестановка.
     + Обновление L и R.
4. **Режимы использования**:
   * DES поддерживает несколько режимов, таких как ECB, CBC, CFB, OFB и Counter Mode.
   * Triple DES (3DES) — развитие DES, где шифрование/расшифрование выполняются троекратно.
5. **Криптостойкость**:
   * DES имеет некоторые слабости, такие как ограниченная длина ключа.
   * Несмотря на это, он был широко использован в прошлом и считается важным этапом в развитии криптографии.
6. **Применение**:
   * DES использовался для защиты данных в банковских системах, электронной коммерции и других областях.

Важно отметить, что в настоящее время DES считается устаревшим из-за ограниченной длины ключа и возможных атак.

Кратко рассмотрим каждый этап алгоритма DES:

1. **Начальная перестановка (Initial Permutation, PI)**:
   * Входные 64 бита данных переупорядочиваются согласно заданной таблице.
   * Это начальное преобразование перед началом шифрования.
2. **Циклы шифрования**:
   * В алгоритме DES 16 циклов преобразования Фейстеля.
   * Каждый цикл включает следующие шаги:
     + Расширение (Expand) половины данных.
     + XOR с ключом.
     + Замена (Substitute) с использованием S-блоков.
     + Перестановка (Permut).
     + Обновление половин данных.
3. **Конечная перестановка (Final Permutation, PI\_1)**:
   * После всех циклов выполняется конечная перестановка.
   * Полученные данные объединяются в итоговый результат.

Программный продукт состоит из функций, каждая из которых отвечает за определенную часть вышеописанного алгоритма.

Сперва идет инициализация матриц, используемых в алгоритме. Определим их.

**PI (Initial Permutation Matrix for the Data)** – это матрица перестановки, которая применяется к исходным данным перед началом шифрования. Она определяет начальное расположение битов в блоке данных. Каждый элемент матрицы указывает, на какую позицию перемещается соответствующий бит. Например, PI[1] = 58 означает, что первый бит перемещается на 58-ю позицию.

**CP\_1 (Initial Permutation Made on the Key)** – это матрица перестановки, применяемая к ключу перед началом процесса генерации подключей. Она определяет начальное расположение битов в ключе. Аналогично PI, каждый элемент CP\_1 указывает, на какую позицию перемещается соответствующий бит.

**CP\_2 (Permutation Applied on Shifted Key)** – эта матрица перестановки применяется к сдвинутому ключу для получения финального подключа. Она также определяет начальное расположение битов в ключе, но используется на более поздних этапах.

**E (Expand Matrix)** – это матрица расширения, используемая при расширении половинок блока данных перед применением функции F. Она увеличивает размер блока данных, добавляя дополнительные биты.

**S\_BOX** – это таблица замен (Substitution Boxes), используемая в функции F. S\_BOX преобразует 6-битовые блоки данных в 4-битовые блоки, используя заранее определенные значения. Всего есть 8 S-блоков, каждый собственной таблицей замен.

**P (Permut After SBox Substitution)** – эта матрица перестановки применяется после замены S-блоков (S-Box). Она определяет новое расположение битов в блоке данных после применения S-блоков. Каждый элемент матрицы указывает, на какую позицию перемещается соответствующий бит.

**PI\_1 (Final Permutation for Datas)** – эта матрица перестановки применяется к зашифрованным данным перед их окончательным выводом. Она определяет конечное расположение битов в зашифрованных данных. Аналогично другим матрицам, каждый элемент PI\_1 указывает, на какую позицию перемещается соответствующий бит.

**SHIFT** – это список, который определяет количество битов, на которое сдвигается ключ на каждом раунде. В алгоритме DES ключ сдвигается влево на определенное количество битов перед каждым раундом.

Далее идут методы, соответствующие обработке строк.

1. Функция **string\_to\_bit\_array(text)** преобразует строку в список битов.Она итерируется по каждому символу в строке text.Для каждого символа она вызывает функцию binvalue(char, 8), чтобы получить его бинарное представление в виде строки из 8 битов.Затем она преобразует эту строку в список целых чисел (битов) и добавляет его к итоговому списку array.Возвращается полученный список битов.
2. Функция **bit\_array\_to\_string(array)** выполняет обратное преобразование: она принимает список битов и возвращает строку, полученную из этих битов. метод генерирует ключи. Сначала она разделяет список битов на байты (подсписки длиной 8 битов). Затем каждый байт преобразуется в целое число, а затем в символ (используя кодировку UTF-8). Наконец, все символы объединяются в одну строку.
3. Функция **binvalue(val, bitsize)**  возвращает бинарное представление значения val в виде строки заданного размера bitsize. Если val является целым числом, оно преобразуется в бинарное представление. Если val является символом, его код ASCII преобразуется в бинарное представление. Если бинарное представление короче ожидаемого размера, добавляются нули в начало строки.
4. Функция **nsplit(s, n)** разделяет строку s на подсписки длиной n. Возвращает список подсписков.

Соответствующие функции представлены на рисунке 1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 – Функции обработки строк.

Далее следует класс **des**, реализующий шаги основные алгоритма DES. Методы класса:

1. Функция **substitute(self, d\_e)** выполняет замену блоков данных после применения S-блоков (Substitution Boxes) в алгоритме DES.

Сначала она разделяет входные данные d\_e на блоки длиной 6 битов (подсписки).

Затем для каждого блока она выполняет следующие действия: извлекает первый и последний биты блока и объединяет их в строку. Это значение используется как индекс для выбора строки в S-блоке. Затем она объединяет остальные биты блока (кроме первого и последнего) и преобразует их в число. Это значение используется как индекс для выбора столбца в S-блоке. Полученное значение из S-блока преобразуется обратно в бинарное представление длиной 4 бита. Результат добавляется к итоговому списку result.

В итоге функция возвращает список битов после замены с помощью S-блоков.

1. Метод **permut(self, block, table)** выполняет перестановку для заданного блока данных (block) с использованием указанной таблицы (table). Он переупорядочивает биты в блоке данных в соответствии с порядком, указанным в таблице. Каждый элемент в таблице указывает позицию соответствующего бита в выходных данных.Результатом является новый блок с переупорядоченными битами.
2. Метод **xpand(self, block, table)** подобно permut также выполняет перестановку для заданного блока данных (block). Он увеличивает размер блока, переупорядочивая его биты в соответствии с порядком, указанным в таблице (table). Расширенный блок используется в алгоритме DES во время выполнения функции Фейстеля.
3. Метод **xor(self, t1, t2)** выполняет побитовую операцию XOR (исключающее ИЛИ) между двумя списками битов (t1 и t2). Он возвращает новый список, где каждый бит является результатом XOR между соответствующими битами в t1 и t2.
4. Метод **shift(self, g, d, n)** выполняет циклический сдвиг битов в двух половинах ключа (g и d). Он сдвигает биты в g и d влево на n позиций. Сдвинутые половины используются в процессе генерации подключей.
5. Метод **addPadding(self)** добавляет заполнение к входным данным (self.text), чтобы их длина была кратной 8 байтам (64 бита). Заполнение состоит из одного байта (8 бит), значение которого равно количеству добавленных байтов.
6. Метод **removePadding(self, data)** удаляет заполнение из заданных данных (data). Он определяет количество байтов заполнения, основываясь на последнем байте данных (который представляет длину заполнения). Метод возвращает исходные данные без добавленного заполнения.

Фрагмент описанных функции представлен на рисунке 2.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 2 – Функции перестановок.

Основным является метод **run(self, key, text, action=ENCRYPT, padding=False**) выполняющий шифрование и дешифрование.  Разберем, что делает каждая часть метода run в алгоритме DES:

1. **Проверка длины ключа и обрезка**:
   * Если длина ключа меньше 8 байтов, вызывается исключение.
   * Если длина ключа больше 8 байтов, ключ обрезается до 8 байтов.
2. **Установка ключа и текста**:
   * Ключ устанавливается как self.password.
   * Текст устанавливается как self.text.
3. **Добавление заполнения (padding)**:
   * Если включено заполнение (параметр padding=True) и выполняется шифрование (action == ENCRYPT), добавляется заполнение к тексту.
   * Если длина текста не кратна 8 байтам, вызывается исключение.
4. **Генерация подключей**:
   * Вызывается метод generatekeys(), который генерирует подключи для алгоритма DES.
5. **Разделение текста на блоки**:
   * Текст разделяется на блоки по 8 байтов.
   * Каждый блок преобразуется в список битов.
6. **Шифрование/дешифрование блоков**:
   * Для каждого блока выполняются следующие действия:
     + Применяется начальная перестановка (PI) к блоку.
     + Блок разделяется на половины g и d.
     + В цикле (16 раундов) выполняется:
       - Расширение (expand) половины d.
       - XOR с ключом (шифрование или дешифрование).
       - Замена (substitute) с использованием S-блоков.
       - Перестановка (P).
       - XOR с половиной g.
       - Обновление g и d.
     + Объединение d и g и применение конечной перестановки (PI\_1).
   * Полученные блоки объединяются в итоговый результат.
7. **Удаление заполнения (если необходимо)**:
   * Если включено заполнение и выполняется дешифрование, удаляется добавленное заполнение.

Фрагмент вышеописанной функции представлен на рисунке 3.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 3 – Фрагмент функции run.

12) Метод **generatekeys** выполняет генерацию подключей для алгоритма DES. Рассмотрим, что он делает:

1. **Преобразование ключа:**
   * Ключ преобразуется в список битов с помощью функции string\_to\_bit\_array.
   * Затем применяется начальная перестановка (CP\_1) к ключу.
2. **Разделение на половины:**
   * Полученный ключ разделяется на две половины: g (левая половина) и d (правая половина).
3. **Генерация подключей:**
   * В цикле (16 раундов) выполняется следующее:
     + Происходит циклический сдвиг битов в g и d на определенное количество позиций (определяется списком SHIFT).
     + Объединение g и d во временный блок tmp.
     + Применение перестановки (CP\_2) к tmp.
     + Полученный блок добавляется в список подключей (self.keys).
4. Функция **encrypt(self, key, text, padding=False)** выполняет шифрование сообщения с использованием ключа. Параметры: key: ключ (строка), text: текст для шифрования (строка) и padding: флаг, указывающий на необходимость добавления заполнения (по умолчанию False). Возвращает зашифрованный текст.
5. Функция **decrypt(self, key, text, padding=False)** выполняет дешифрование сообщения с использованием ключа. Параметры аналогичны функции encrypt. Возвращает исходный текст.

Соответствующие функции представлены на рисунке 4.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 4 – Функция генерация ключей, шифрования и дешифрования сообщения.

В конце программы реализовано сперва преобразование ключа и открытого текста в двоичные последовательности. Ключ “SecretKey” преобразуется в список битов с помощью функции string\_to\_bit\_array. Открытый текст “Hello, world!” также преобразуется в список битов. Выводится двоичная последовательность ключа и открытого текста.

Затем выполняется преобразование обратно в строку. Полученные биты открытого текста преобразуются обратно в строку. Выводится зашифрованный текст (рисунок 5).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 – Вызов соответствующих функций.

**Результаты работы программы:**

На рисунке 6 представлен результат работы вышеописанной программы.

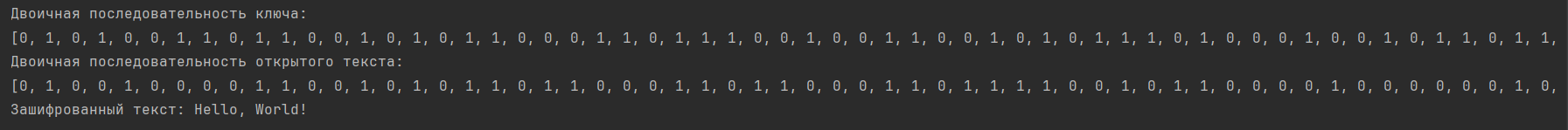


Рисунок 6 – Результат работы программы.

**Листинг программы**

Файл LW\_9.py

# реализация DES

# PI (Initial Permutation Matrix for the Data)

PI = [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]

# CP\_1 (Initial Permutation Made on the Key)

CP\_1 = [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,

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]

# CP\_2 (Permutation Applied on Shifted Key)

CP\_2 = [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]

# E (Expand Matrix)

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]

# SBOX (Substitution Boxes)

S\_BOX = [

[[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],

]

]

# P (Permut After SBox Substitution)

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]

# PI\_1 (Final Permutation for Datas)

PI\_1 = [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]

SHIFT = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

# преобразование строки в список битов

def string\_to\_bit\_array(text):

array = list()

for char in text:

binval = binvalue(char, 8)

array.extend([int(x) for x in list(binval)])

return array

# обратное преобразование: из списка битов возвращается строка, полученная из этих битов

def bit\_array\_to\_string(array):

res = ''.join([chr(int(y, 2)) for y in [''.join([str(x) for x in \_bytes]) for \_bytes in nsplit(array, 8)]])

return res

# бинарное представление значения в виде строки заданного размера

def binvalue(val, bitsize):

binval = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]

if len(binval) > bitsize:

raise "binary value larger than the expected size"

while len(binval) < bitsize:

binval = "0" + binval

return binval

# выделение подстрок в строке s длиной n

def nsplit(s, n):

return [s[k:k + n] for k in range(0, len(s), n)]

# флаги для шифровки/дешифровки

ENCRYPT = 1

DECRYPT = 0

# класс, реализующий шаги алгоритма DES

class des():

def \_\_init\_\_(self):

self.password = None

self.text = None

self.keys = list()

# замену блоков данных после применения S-блоков

def substitute(self, d\_e):

subblocks = nsplit(d\_e, 6)

result = list()

for i in range(len(subblocks)):

block = subblocks[i]

row = int(str(block[0]) + str(block[5]), 2)

column = int(''.join([str(x) for x in block[1:][:-1]]), 2)

val = S\_BOX[i][row][column]

bin = binvalue(val, 4)

result += [int(x) for x in bin]

return result

# перестановка для заданного блока данных (block) с использованием указанной таблицы (table)

def permut(self, block, table):

return [block[x - 1] for x in table]

# перестановка для заданного блока данных (block - увеличение размер блока в соответствии порядка таблицы (table)

def expand(self, block, table):

return [block[x - 1] for x in table]

# побитовая операция xor

def xor(self, t1, t2):

return [x ^ y for x, y in zip(t1, t2)]

# циклический сдвиг битов

def shift(self, g, d, n):

return g[n:] + g[:n], d[n:] + d[:n]

# добавление заполнений во входных данных

def addPadding(self):

pad\_len = 8 - (len(self.text) % 8)

self.text += pad\_len \* chr(pad\_len)

# удаление заполнений из заданных данных

def removePadding(self, data):

pad\_len = ord(data[-1])

return data[:-pad\_len]

def run(self, key, text, action=ENCRYPT, padding=False):

# проверка длины ключа и обрезка

if len(key) < 8:

raise "Ключ должен иметь длину 8 байт"

elif len(key) > 8:

key = key[:8]

self.password = key

self.text = text

# добавление заполнений (в случае шифрования)

if padding and action == ENCRYPT:

self.addPadding()

elif len(self.text) % 8 != 0:

raise "Размер данных должен быть кратен 8"

# генерация подключей

self.generatekeys()

# разделение текста на блоки

text\_blocks = nsplit(self.text, 8)

result = list()

# для каждого блока

for block in text\_blocks:

# преобразование в список байтов

block = string\_to\_bit\_array(block)

# начальная перестановка PI

block = self.permut(block, PI)

# разделение блоков на половины g и d

g, d = nsplit(block, 32)

tmp = None

# в цикле (в течении 16 раундов)

for i in range(16):

# расширение половины d

d\_e = self.expand(d, E)

# шифрование/дешифрование

# выполнение операции XOR с ключом

if action == ENCRYPT:

tmp = self.xor(self.keys[i], d\_e)

else:

tmp = self.xor(self.keys[15 - i], d\_e)

# замена с использованием S-блоков

tmp = self.substitute(tmp)

# перестановка (P)

tmp = self.permut(tmp, P)

# XOR с половиной g

tmp = self.xor(g, tmp)

# обновление g и d

g = d

d = tmp

# объединение g и d и применение конечной перестановки (PI\_1)

result += self.permut(d + g, PI\_1)

# преобразование в строку

final\_res = bit\_array\_to\_string(result)

# удаление заполнений (в случае дешифрования)

if padding and action == DECRYPT:

return self.removePadding(final\_res)

else:

return final\_res

# генерация подключей (16)

def generatekeys(self):

self.keys = []

# преобразование в список битов

key = string\_to\_bit\_array(self.password)

# начальная перестановка (CP\_1) ключа

key = self.permut(key, CP\_1)

# выделение половин g и d

g, d = nsplit(key, 28)

# в цикле(16 раундов)

for i in range(16):

# циклический сдвиг битов в g и d

g, d = self.shift(g, d, SHIFT[i])

# объединение g и d

tmp = g + d

self.keys.append(self.permut(tmp, CP\_2))

# функция шифрования

def encrypt(self, key, text, padding=False):

return self.run(key, text, ENCRYPT, padding)

# функция дешифрования

def decrypt(self, key, text, padding=False):

return self.run(key, text, DECRYPT, padding)

# открытый текст

plaintext = "Hello, World!"

# ключ

key = "SecretKey"

# преобразование в двоичные последовательности

key\_bits = string\_to\_bit\_array(key)

print("Двоичная последовательность ключа:")

print(key\_bits)

plaintext\_bits = string\_to\_bit\_array(plaintext)

print("Двоичная последовательность открытого текста:")

print(plaintext\_bits)

# преобразование обратно в строку

ciphertext = bit\_array\_to\_string(plaintext\_bits)

print("Зашифрованный текст:", ciphertext)

# # Ввод двоичных последовательностей ключа и открытого текста

# key\_binary = input("Введите двоичную последовательность ключа: ")

# plaintext\_binary = input("Введите двоичную последовательность открытого текста: ")

#

# # Преобразование в список битов

# key\_bits = [int(bit) for bit in key\_binary]

# plaintext\_bits = [int(bit) for bit in plaintext\_binary]

#

# # Преобразование обратно в строку

# ciphertext = bit\_array\_to\_string(plaintext\_bits)

# print("Зашифрованный текст:", ciphertext)