Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Криптографические методы обеспечения информационной безопасности»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

«Основные структурные элементы блочного симметричного алгоритма DES»

Выполнил:

Чу Ван Доан, студент группы N3347

(подпись)

Проверил:

Таранов Сергей Владимирович

(отметка о выполнении)

(подпись)

Санкт-Петербург

2025 г.

Содержание

Введение

Цель: изучить основные принципы работы алгоритмы DES.

Задание

Для достижения поставленной цели необходимо решить следующие задачи:

- Проанализировать алгоритм DES
- Реализовать DES с возможностью отслеживать результаты выполнения
- раундов
- Выполнить отчёт.

Ход работы

1. Обзор DES (Data Encryption Standard)

DES (Стандарт шифрования данных — Data Encryption Standard) является первым в мире стандартом шифрования данных, предложенным Агентством национальной безопасности США (NSA) на основе усовершенствованного алгоритма Lucifer, разработанного компанией IBM и опубликованного в 1964 году. DES получил широкое распространение в США и многих других странах в течение 70-х, 80-х и 90-х годов, пока не был заменён более современным стандартом AES (Advanced Encryption Standard) в 2002 году.

Входными данными для DES является блок размером 64 бита, и выход также представляет собой блок размером 64 бита. Длина шифровального ключа составляет 56 бит, однако изначально он имеет длину 64 бита — биты, находящиеся на позициях, кратных 8, удаляются и используются для проверки чётности.

2. Алгоритм

DES — это блочный шифр, он обрабатывает каждый блок открытого текста с фиксированной длиной 64 бита. Перед тем как пройти через 16 основных раундов, данные, подлежащие шифрованию, разбиваются на блоки по 64 бита, и каждый из этих блоков поочерёдно проходит 16 раундов шифрования DES.

- **Входные данные (Input):** открытый текст $M = m_1 m_2 \dots m_{64}$ — это блок из 64 бит, ключ шифрования $K = k_1 k_2 \dots k_{64}$ — 64-битный ключ.
- **Выходные данные (Output):** зашифрованный текст $C = c_1 c_2 \dots c_{64}$ — блок из 64 бит.
- **Шаг 1: Генерация подключей.** Используя алгоритм генерации подключей из ключа K, получаем 16 подключей: $K_1, K_2, \dots K_{16}$
- **Шаг 2: Использование начальной перестановки IP (Initial Permutation),** чтобы переставить биты сообщения M. Полученный результат делится на две половины

$$L_0 = m_{63} m_{62} \dots m_{32},$$
$$E_0 = m_{31} m_{30} \dots m_0.$$

- **Шаг 3: Для i от 1 до 16 выполняем:**

  Вычисляем L_i и R_i по формулам:

  $L_i = R_i - 1$

  $R_i = L_i - 1\ XOR\ f(R_i - 1,\ K_i)$,

  где $f(R_i - 1, K_i) = R(S(E(R_i - 1)\ XOR\ K_i))$

- **Шаг 4: Меняем местами блоки L_{16} и R_{16},** получаем блок $R_{16}L_{16} = b_1 b_2 \ldots b_{64}$.
- **Шаг 5: Используем финальную перестановку FP (Final Permutation)** — это инверсия начальной перестановки IP. Получаем зашифрованный текст:

  $C = IP - 1(b_1 b_2 \ldots b_{64}$

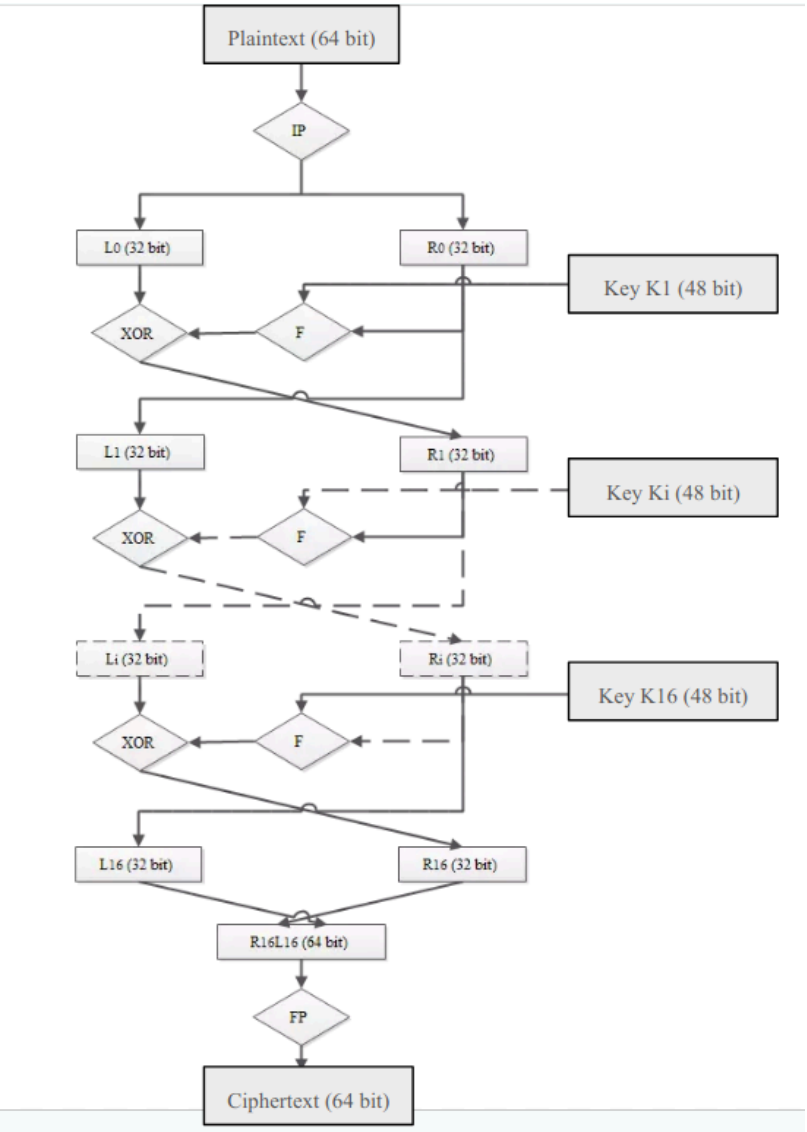


Рисунок 1 - процесс шифрования блока

3. Процесс генерации подключей

16 раундов алгоритма DES выполняются по одному и тому же алгоритму, но с разными 16 подключами.

Все подключи генерируются из основного ключа DES с помощью алгоритма генерации подключей.

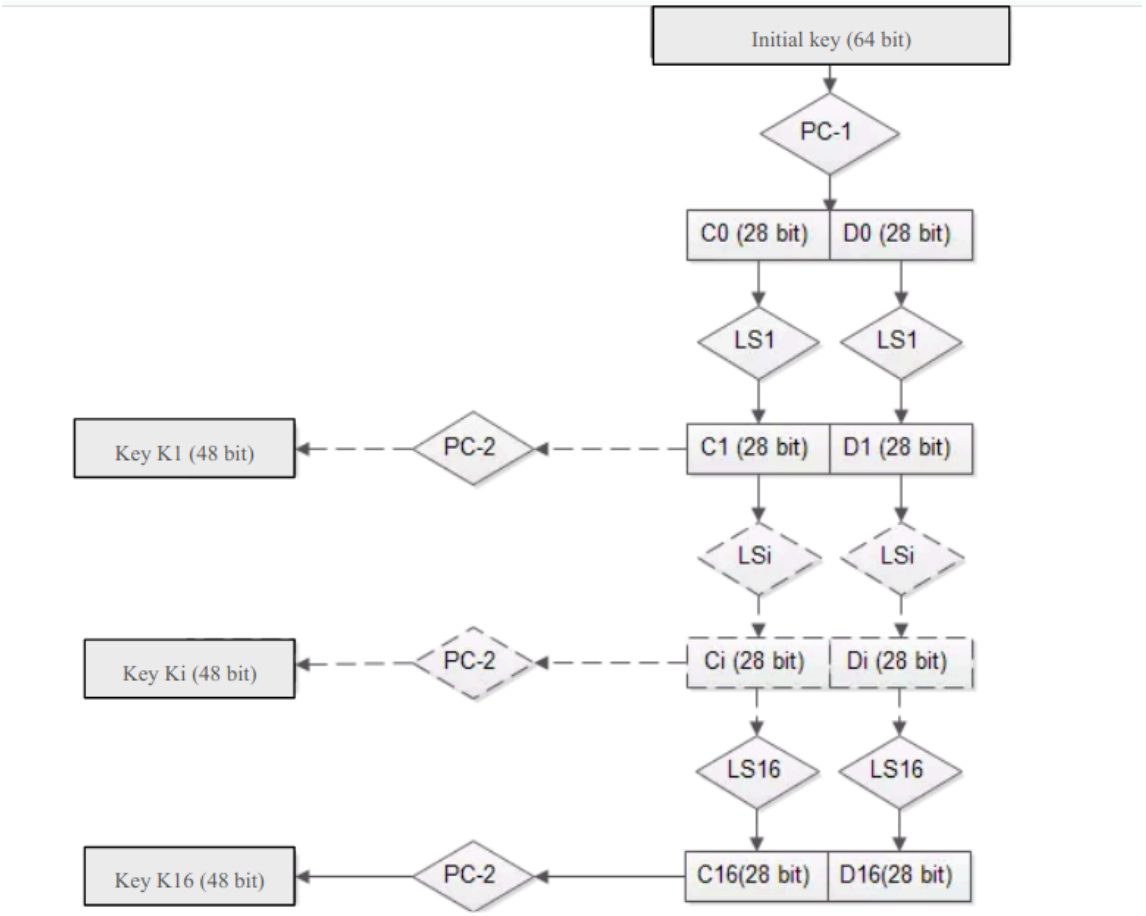


Рисунок 2 - Процесс генерации подключей

Начальный ключ K1 — это строка длиной 64 бита. Каждый 8-й бит в каждом байте используется для проверки на ошибки, в результате чего получается 56-битная строка. После удаления битов проверки эта 56-битная строка подвергается перестановке. Обе операции выполняются с помощью перестановочной матрицы PC-1 (Permuted Choice 1).

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

Рисунок 3 -таблица PC-1

Затем результат после применения PC-1 делится на две части:
C0 — первые 28 битов, D0 — последние 28 битов.
Каждая часть обрабатывается независимо:

$C_i = LS_i(C_{i-1})$
$D_i = LS_i(D_{i-1})$, при $1 \leq i \leq 16$.

Здесь LSi обозначает циклический сдвиг битов влево на 1 или 2 позиции в зависимости от значения i.

Раунд	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Сдвигов	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Наконец, используется фиксированная перестановка PC-2 (Permuted Choice 2), чтобы переставить 56-битную строку CiDi и получить подключ Ki длиной 48 бит.

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

Рисунок 4 - таблица PC-2

4. Процесс шифрования DES

4.1. Этап 1

Для заданного открытого текста x, создаётся новая строка x′ путём перестановки битов x согласно начальной перестановке 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

Рисунок 5 - таблица IP

Затем строка x' делится на две части: L_0 и R_0.

$x' = IP(x) = L_0R_0$, где L_0 — первые 32 бита, а R_0 — последние 32 бита.

4.2. Этап 2

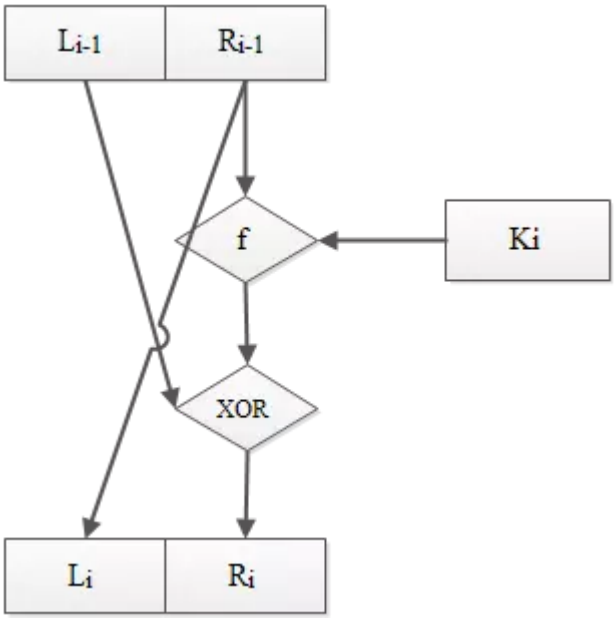


Рисунок 6 - Схема одного раунда в алгоритме DES

Вычисления выполняются 16 раз с использованием определённой функции.

В каждом раунде ($1 \leq i \leq 16$) значения L_i и R_i рассчитываются по следующим правилам:

- $L_i = R_i - 1$
- $R_i = L_i - 1 \ XOR \ f(R_i - 1, K_i)$

Здесь K_i — подключ, сгенерированный в процессе создания ключей, а `f` — функция, которая будет описана далее.

4.3. Этап 3

Применяя финальную перестановку FP к битовой строке R16L16, мы получаем зашифрованное сообщение y: **y = FP(R16L16)**.

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	59	17	57	25

Рисунок 7 - Таблица конечной перестановки FP (Final Permutation)

5. Дешифрование DES

Процесс дешифрования DES аналогичен процессу шифрования. Единственное различие заключается в следующем:

$L_i = R_{i-1}$,

$R_i = L_{i-1} \text{ XOR } f(R_{i-1}, K_{16-i+1})$.

Таким образом, ключ K, используемый в функции F, будет применяться в обратном порядке — от K16 до K1.

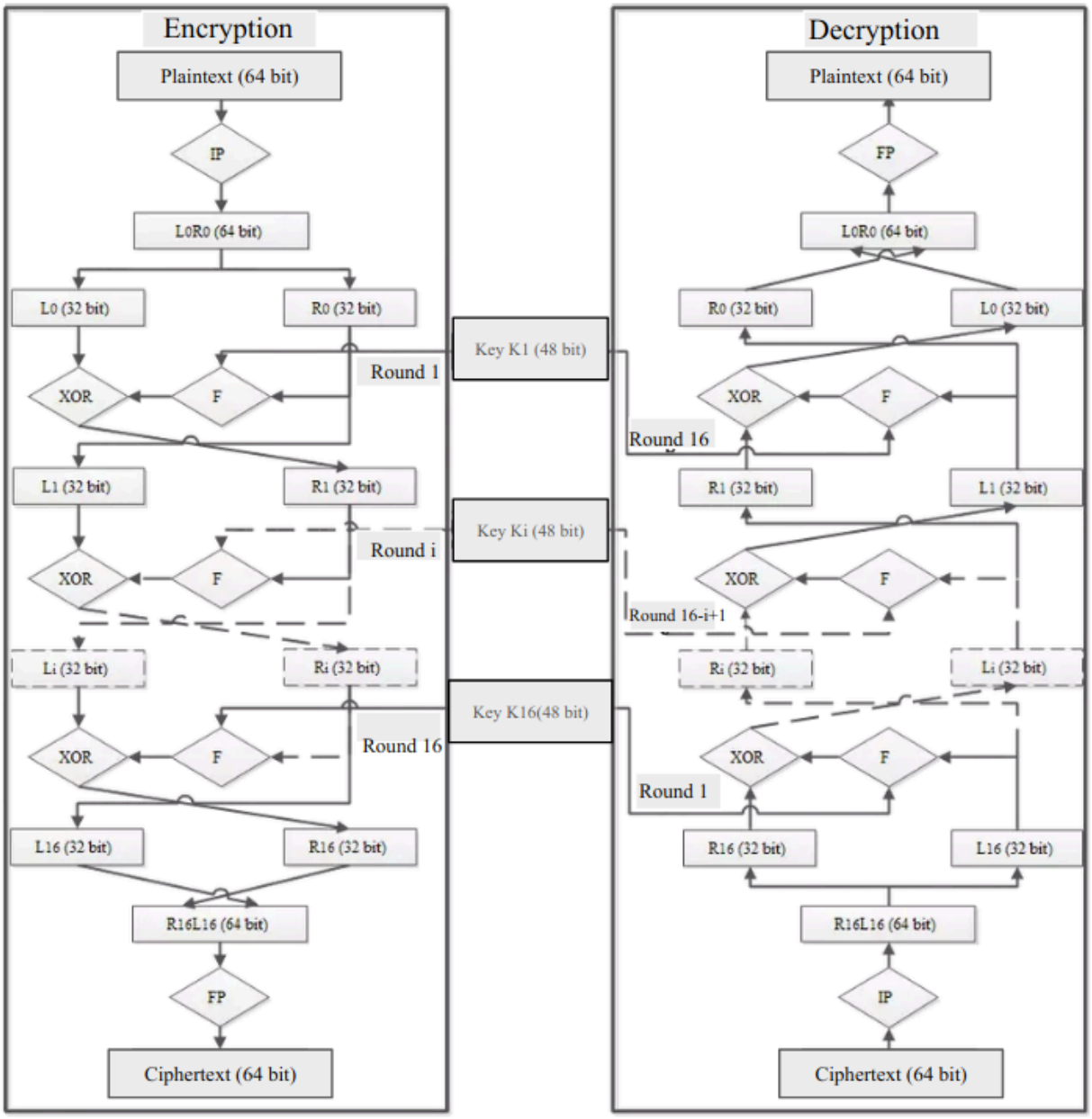


Рисунок 8 - процесс шифрования и дешифрования блока

6. Функция F

На вход функции f подаются 2 переменные:

- Первая переменная: Ri−1 — это битовая строка длиной 32 бита.
- Вторая переменная: Ki — это битовая строка длиной 48 бит.

Выходом функции f является строка длиной 32 бита.

- Процесс работы функции f включает следующие шаги:
- Переменная Ri−1 расширяется до строки длиной 48 бит с помощью расширяющей перестановки E (Expansion Permutation).

По сути, расширение E(Ri−1) — это перестановка с дублированием 16 бит Ri−1.

- Вычисляется: **E(Ri−1) XOR Ki**
- Результат операции делится на 8 строк по 6 бит: B1, B2, ..., B8.
- Каждая 6-битная строка Bi подаётся на вход одного из 8 блоков S1, S2, ..., S8 (так называемых S-блоков).

  Каждый S-блок представляет собой фиксированную таблицу 4×16 с номерами столбцов от 0 до 15 и строк от 0 до 3.

  Для каждого Bi = b1b2b3b4b5b6:
  - два бита b1 и b6 определяют номер строки r,
  - четыре бита b2b3b4b5 определяют номер столбца c.

    Тогда Si(Bi) = Si(r, c), и результат Si(Bi) — это 4-битное значение.

    Таким образом, из 8 блоков Bi ($1 \leq i \leq 8$) получаем 8 блоков Ci по 4 бита ($1 \leq i \leq 8$).
- Итоговая строка C = C1C2C3C4C5C6C7C8 имеет длину 32 бита и подвергается перестановке по таблице P (P-блок).

  Результат перестановки P(C) является выходом функции f(Ri−1, Ki).

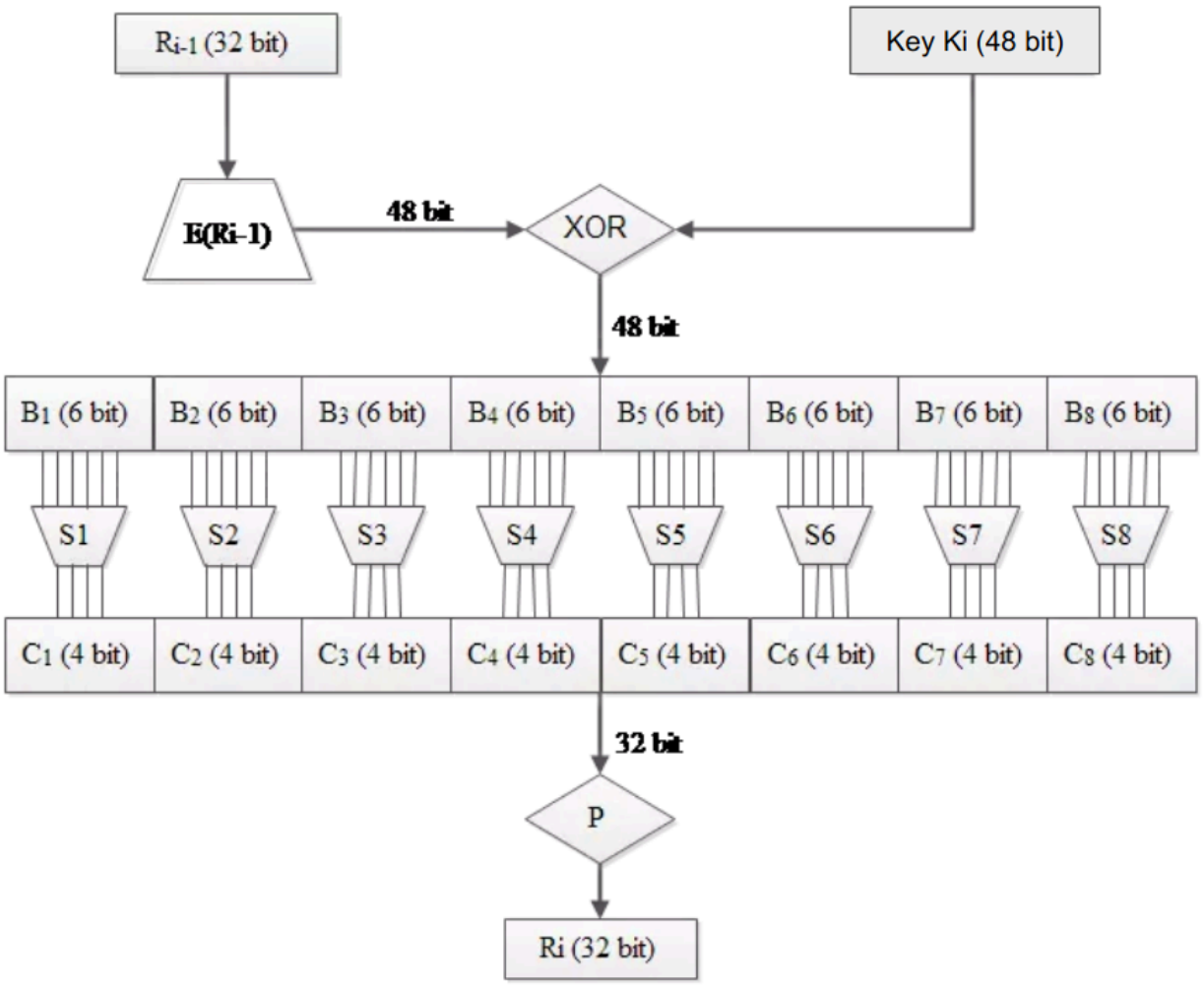


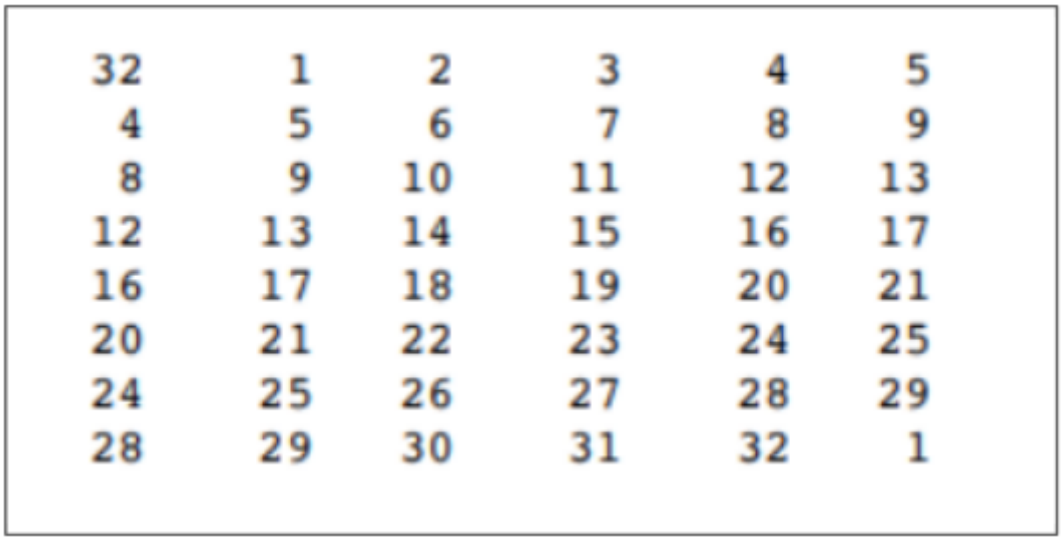
Рисунок 9 - Функция F

6.1. Функция расширения E

Функция расширения E увеличивает длину Ri−1 с 32 бит до 48 бит путём изменения порядка битов,
а также повторения некоторых битов. Эта операция преследует две цели:

- Приведение длины Ri−1 к длине ключа K для выполнения операции XOR по модулю 2.
- Увеличение длины результата, чтобы он мог быть сжат в процессе подстановки.
  Однако обе цели направлены на одну главную задачу — обеспечение безопасности данных.
  Позволяя одному биту появляться в двух позициях после расширения, обеспечивается зависимость выходных битов от входных.

Схема ниже показывает, как начальные 32 бита расширяются до 48 бит с помощью перестановки:



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

Рисунок 10 - таблица E-expansion

6.2. S-блоки

После выполнения операции XOR между E(Ri−1) и Ki,
результирующая 48-битная строка делится на 8 блоков и подаётся на вход 8 S-блокам.
Каждый S-блок принимает на вход 6 бит и выдаёт 4 бита на выходе.

Полученный результат представляет собой строку длиной 32 бита,
которая затем поступает в P-блок.

- Каждая строка в S-блоке представляет собой перестановку целых чисел от 0 до 15.
- S-блоки являются нелинейными, то есть их выход не должен быть линейной функцией от входа.
- Изменение одного или нескольких битов на входе приводит к изменению на выходе.
- Если входные данные двух разных значений S-блока отличаются хотя бы на 2 бита (например, в битах 3 и 4),
  то соответствующие выходные значения должны отличаться минимум на 2 бита.
  Иными словами, **S(x)** и **S(x XOR 001100)** должны различаться как минимум по 2 битам.

S1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Рисунок 11 - таблица SBox1

S2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Рисунок 12 - таблица SBox2

S3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Рисунок 13 - таблица SBox3

S4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Рисунок 14 - таблица SBox4

S5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	0	14	2	13	6	15	0	9	10	4	5	3

Рисунок 15 - таблица SBox5

S6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Рисунок 16 - таблица SBox6

S7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Рисунок 17 - таблица SBox7

S8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Рисунок 18 - таблица SBox8

6.3. P-блок

Каждые 4 бита, полученные на выходе из S-блоков, объединяются в порядке следования блоков и подаются на вход P-блоку.

P-блок представляет собой простую перестановку битов между собой.

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

Рисунок 19 - таблица P

7. Пример

Предположим, у нас есть сообщение **M = 0123456789ABCDEF** и ключ **K = 13345799BBCDDF1**.

Ключ K задан в шестнадцатеричном виде. Мы будем выполнять шифрование и расшифровку по алгоритму DES в соответствии со следующими шагами.

Представим M и K в двоичной форме. Тогда мы получим:

64-битный блок для открытого текста M:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

Разделим 64-битный блок на две половины, левую и правую, каждая по 32 бита:

- Левая часть **L = 0000 0001 0010 0011 0100 0101 0110 0111**
- Правая часть **R = 1000 1001 1010 1011 1100 1101 1110 1111**

DES работает с ключами длиной 56 бит.

Хотя ключ действительно хранится как 64 бита, каждый восьмой бит используется для контроля чётности (то есть биты с номерами 8, 16, 24, 32, 40, 48, 56 и 64 не участвуют в шифровании).

Мы будем использовать только 56 бит, пронумерованных от 1 до 64, двигаясь слева направо, и исключим контрольные биты перед формированием ключей.

Переведя ключ K в двоичный вид, получим:

K = 0001001100110100010101110111100110011011101111001101111111110001

Алгоритм DES выполняется следующим образом:

Шаг 1: Генерация 16 подключей, каждый длиной 48 бит.

Исходный 64-битный ключ подвергается перестановке PC-1, в результате чего используется только 56 бит из исходного ключа.

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

Из 64-битного исходного ключа:

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

Согласно таблице перестановки выше, например:
бит 57 ключа K — это '1', бит 49 — '1', биты 41, 33, 25, 17, 9 — это соответственно 1, 1, 0, 0, 0...

Таким образом, после применения перестановки PC-1 мы получаем 56-битный ключ:

K+ = 11110000 01100111 01010101 01010101 10101010 10001111

Затем ключ делится на две части: левую и правую — **C0** и **D0**, каждая по 28 бит:

- **C0 = 11110000 01100111 01010101**
- **D0 = 01010101 10101010 10001111**

Имея C0 и D0, мы начинаем процесс генерации 16 пар CnDn, где $1 \leq n \leq 16$. Каждая пара CnDn формируется на основе предыдущей пары C(n−1)D(n−1), следуя определённому правилу сдвига.

Раунд	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Кол-во циклических сдвигов влево	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Это означает, что, например, пары **C3**, **D3** получаются путём **двойного циклического сдвига влево** от **C2**, **D2**,
а пары **C2**, **D2** — путём **одного сдвига** от **C1**, **D1**.
Исходные значения:

C0 = 11110000 01100111 01010101
D0 = 01010101 10101010 10001111

По вышеуказанным правилам сдвига получаем:

C1 = 11100000 11001110 10101010
D1 = 10101011 01010101 00011110

Аналогично, применяя таблицу количества сдвигов, получаем следующие пары CnDn ($1 \leq n \leq 16$):

C2 = 11000001 10011101 01010100 D2 = 01010110 10101010 00111101
C3 = 00000110 01110101 01010001 D3 = 01011010 10101000 11110101
C4 = 00110011 10010101 01000101 D4 = 01101010 10100011 11010101
C5 = 11001111 00101010 10010100 D5 = 10101010 10001111 01010101
C6 = 00111100 10101001 01001001 D6 = 10101001 00011110 10101011
C7 = 11110010 10100101 00100110 D7 = 10100100 01111010 10101110
C8 = 11001010 10010100 10011011 D8 = 10010001 11101010 10111010
C9 = 00101010 01010010 01101111 D9 = 01000111 10101010 11101010
C10 = 10101001 01001001 10111100 D10 = 00011110 10101011 10101010
C11 = 01010010 10010011 01111001 D11 = 01111010 10101110 10101001

C12 = 01001001 00100110 11110010 D12 = 11110101 01011101 01010011
C13 = 00100100 10011011 11001001 D13 = 11101010 10111010 10100111
C14 = 11100110 11110010 01101001 D14 = 10101011 10101010 01111011
C15 = 11001101 11100100 11010011 D15 = 01010111 01010010 11110111
C16 = 10011011 11001001 10100111 D16 = 01011101 01001011 11011110

Теперь мы строим **подключи Kn** (где $1 \leq k \leq 16$),
применяя **перестановку PC-2** к каждой паре CnDn.
Каждая пара имеет 56 бит, но таблица **PC-2 использует только 48 бит** из них.

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

Из **C1D1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110**
и используя таблицу перестановки PC-2, определим первый подключ **K1** следующим образом:

- 14-й бит C1D1 — это первый бит K1 → '0'

- 17-й бит — второй бит K1 → '0'
- 32-й бит — последний бит K1 → тоже '0'

Таким образом:
K1 = 000010 110000 001011 101111 111001 000001 110010

Аналогично, другие подключи (K2 до K16) будут:

K2 = 011110 001010 110111 011001 110100 100111 100101
K3 = 010101 011111 110010 000100 010100 101101 111100
K4 = 011100 101010 010011 110100 011101 101111 001101
K5 = 011111 000110 110100 000111 111001 101100 111001
K6 = 011000 110101 101000 101111 111010 110000 000111
K7 = 111011 001011 001101 111011 100101 111010 100100
K8 = 111101 111000 101000 111100 101111 010111 011001
K9 = 111001 001101 110110 001011 110101 110000 001001
K10 = 111100 001110 110111 011101 110000 110011 101101
K11 = 001011 001110 101101 010100 011111 001011 101100
K12 = 011101 001110 001001 100100 100111 111100 100011
K13 = 100101 111101 110000 111110 110111 001010 101101
K14 = 010111 110100 011011 111001 110101 101010 110010
K15 = 101111 111001 000011 001101 010011 101011 010100
K16 = 110001 110011 111100 011101 111100 000101 000011

Шаг 2: Шифрование 64-битного блока данных

В начале выполняется начальная перестановка **IP** над 64-битным блоком данных M. Биты будут упорядочены в соответствии с таблицей IP, приведённой ниже.

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

Применяя начальную перестановку IP к исходному блоку открытого текста **M**, получаем:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
IP = 1100 1100 0000 0000 1111 1111 1010 1010 0000 0000 1100 1100 1111 1111 1010 1010

Затем блок IP делится на две части:
левая половина **L0** — 32 бита и правая половина **R0** — 32 бита.
После IP получаем:

- **L0 = 1100 1100 0000 0000 1111 1111 1010 1010**
- **R0 = 1111 0000 0011 1010 1010 1100 1010 1010**

Теперь мы выполняем 16 раундов, используя функцию f,
которая работает с двумя блоками: один 32-битный блок данных и один 48-битный подключ Kn,
чтобы создать новый 32-битный блок.

Правила:

- **Ln = Rn−1**
- **Rn = Ln−1 XOR f(Rn−1, Kn)**

Для n = 1, пусть:
K1 = 000010 110000 001011 101111 111001 000001 110010
L0 = R0 = 1111 0000 0011 1010 1010 1100 1010 1010
R1 = L0 XOR f(R0, K1)

Чтобы вычислить функцию f, нужно расширить 32-битный блок R(n−1) до 48 бит, используя таблицу выбора битов, в которой определённые биты из R(n−1) повторяются. Этот процесс называется **расширением E**.

Таким образом, **E(R(n−1))** из 32-битного блока создаёт 48-битный блок, который затем делится на 8 блоков по 6 бит.
Биты выбираются согласно **таблице выбора битов E (E Bit-Selection Table)**.

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

Мы вычисляем **E(R0)** из **R0** следующим образом:
R0 = 1111 0000 1010 1010 1111 0000 1010 1010
E(R0) = 011110 100001 010101 010101 011110 100001 010101 010101

Как видно, каждый 4-битный блок данных был расширен до 6 бит.
Затем, чтобы вычислить функцию **f**, мы выполняем XOR между E(R(n−1)) и ключом **Kn**:
E(R(n−1)) ⊕ **Kn**

Например, для K1:

K1 = 000010 110000 001011 101111 111001 000001 110010

E(R0) = 011110 100001 010101 010101 011110 100001 010101 010101

K1 ⊕ E(R0) = 011100 010001 011110 111010 100111 100000 100111 000111

Далее мы используем каждую из 6-битных групп как адреса для поиска в так называемых **S-блоках (S-boxes)**.

Каждая 6-битная группа определяет одно значение в своём S-блоке.

Адрес выбирается по краевым битам и средним четырём битам:

- 1-й и 6-й биты группы образуют **номер строки** (2 бита)
- Биты с 2 по 5 образуют **номер столбца** (4 бита)

S-блок преобразует 6-битный вход в 4-битный выход.

В итоге, 8 блоков по 6 бит (B1, B2, ..., B8) преобразуются в 8 блоков по 4 бита,

что даёт в результате **32-битную строку**.

Обозначим:

Kn ⊕ E(R(n−1)) = B1B2B3B4B5B6B7B8

Каждый **Bi** — 6-битный блок, как мы vừa вычислили выше.

Теперь нужно вычислить:

S1(B1), S2(B2), ..., S8(B8)

Каждый S-блок **S1, S2, ..., S8** принимает 6-битный вход и возвращает 4-битный выход.

Далее следует таблица определения **S1**.

S1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Рисунок 20 - S1

S1 — это функция, определяемая таблицей, и **B** — это 6-битный входной блок.
Значение **S1(B)** определяется следующим образом:

Определим первый и последний биты B:
они образуют двоичное число вида 00, 01, 10 или 11, то есть значение от 0 до 3 в десятичной системе.
Обозначим это значение как **i**.

Оставшиеся 4 бита посередине имеют значение от 0000 до 1111 (т.е. от 0 до 15 в десятичной системе).
Обозначим это значение как **j**.

Затем мы смотрим в таблицу S1 на строке **i** и столбце **j**.
Результатом будет число от 0 до 15, представленное в виде 4-битного двоичного числа.
Именно оно является выходным значением S1(B).

Ví dụ (например):
Пусть входной блок **B1 = 011000**

- Первый бит = "0", последний бит = "0" → **00** → i = 0
- Средние 4 бита = "1100" → j = 12

Итак, мы ищем в таблице S1 значение на **строке 0**, **в столбце 12** (позиция (0,12)).
Значение там — **5**, что в двоичном виде — **0101**.

Следовательно:
S1(B1) = 0101

Таблицы S2, S3, ..., S8 определяются аналогично.

S2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Рисунок 21 - S2

S3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Рисунок 22 - S3

S4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Рисунок 23 - S4

S5	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	0	14	2	13	6	15	0	9	10	4	5	3

Рисунок 24 - S5

S6	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Рисунок 25 - S6

S7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Рисунок 26 - S7

S8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Рисунок 27 - S8

Для первого раунда мы уже вычислили:

K1 ⊕ E(R0) = 011000 010001 011110 111010 100001 100110 010100 100111

Используя таблицы S-блоков и группы B1, B2, ..., B8,
мы получаем значения:
S1(B1) S2(B2) S3(B3) S4(B4) S5(B5) S6(B6) S7(B7) S8(B8) = 0101 1100 1000 0010 1011 0101 1001 0111

Последний шаг в вычислении функции **f** — это применение перестановки **P**
к выходам S-блоков, чтобы получить окончательное значение функции:
f(R0, K1) = P(S1(B1) S2(B2) S3(B3) S4(B4) S5(B5) S6(B6) S7(B7) S8(B8))

Перестановка **P** определяется в таблице ниже.

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

Из выходов 8 S-блоков:
S1(B1) S2(B2) S3(B3) S4(B4) S5(B5) S6(B6) S7(B7) S8(B8) = 0101 1100 1000 0010 1011 0101 1001 0111

Получаем:
f(R0, K1) = 0010 0011 0100 1010 1010 1001 1011 1011

R1 = L0 XOR f(R0, K1)

L0 = 1100 1100 0000 0000 1111 1111 1010 1010

XOR:

1100 1100 0000 0000 1111 1111 1010 1010

⊕ 0010 0011 0100 1010 1010 1001 1011 1011

= **1110 1111 0100 1010 0101 0101 0000 0100**

Следующий раунд:

L2 = R1, R2 = L1 XOR f(R1, K2)

И так далее — аналогично для всех 16 раундов.

В последнем раунде мы получаем блок **L16R16**.

Меняем порядок этих двух блоков (объединяем как R16L16),

затем применяем финальную перестановку **IP-1**,

которая определяется таблицей IP-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

Обработав все 16 раундов, как описано выше, на 16-м раунде получаем:

L16 = 0100 0011 0100 0010 0011 0010 0011 0100

R16 = 0000 1010 0100 1100 1101 1001 1001 0101

Меняем порядок этих двух блоков и объединяем:

R16L16 = 0000 1010 0100 1100 1101 1001 1001 0101 0100 0011 0100 0010 0011 0010 0011 0100

Затем применяем обратную перестановку **IP-1**:

IP-1 = 10000011 11101000 00000101 01010100 00000101 10101100 00000101

Преобразуем результат IP-1 в шестнадцатеричную форму:

85E813540F0AB405

Вывод: Таким образом, зашифрованная форма M = 0123456789ABCDEF
есть: **C = 85E813540F0AB405**

Дешифрование: Процесс дешифрования просто является обратным к шифрованию,
с теми же шагами, но ключи применяются в обратном порядке —
то есть от **K16 до K1**, и процедура полностью повторяет порядок шифрования.

8. Демонстрация работы

```
 ⚡ chu ❯❯ python3 main.py
Original message: Hello from DES Cipher!
Key (hex): 380294585078a9ec
Encrypted (hex): 2cb921bfe36cf83ec3d43639c267a796f453183c86cade33
Decrypted: Hello from DES Cipher!
 chu-latitude-5510  \Documents\SOC\Các phương pháp mật mã đảm bảo an ni
 ⚡ chu ❯❯ python3 main.py
Original message: Hello from DES Cipher!
Key (hex): c2c2004593dbc4bd
Encrypted (hex): 74b5119a970318240de782f5f4a503e437b6530deba450ab
Decrypted: Hello from DES Cipher!
```

Рисунок 28 - пример

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- Проанализирован алгоритм DES
- Реализован алгоритм DES
- Выполнен отчёт

Таким образом, все поставленные задачи решены, цель работы успешно достигнута.

ПРИЛОЖЕНИЕ А

Листинг А.1 – Код файла tables.py

```
# tables.py

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
]

FP = [
    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
]

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
]

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
]
```

```
PC1 = [
   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
]

PC2 = [
   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
]

SHIFT_SCHEDULE = [
   1, 1, 2, 2, 2, 2, 2, 2,
   1, 2, 2, 2, 2, 2, 2, 1
]

# S_BOX
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]]
]
```

Листинг А.2 – Код файла DES.py

```
# des.py

from tables import *

def permute(block, table, block_size):
    result = 0
    for i in range(len(table)):
        bit = (block >> (block_size - table[i])) & 1
        result = (result << 1) | bit
    return result

def left_rotate(val, n):
    return ((val << n) & 0xfffffff) | (val >> (28 - n))

def generate_keys(key):
    key = permute(key, PC1, 64)
    C, D = key >> 28, key & 0xfffffff
    keys = []
    for shift in SHIFT_SCHEDULE:
        C = left_rotate(C, shift)
        D = left_rotate(D, shift)
        keys.append(permute((C << 28) | D, PC2, 56))
```

```
    return keys

def sbox_substitute(block):
    output = 0
    for i in range(8):
        segment = (block >> (42 - 6 * i)) & 0x3f
        row = ((segment >> 5) << 1) | (segment & 1)
        col = (segment >> 1) & 0xf
        output = (output << 4) | S_BOX[i][row][col]
    return output

def des_round(L, R, key):
    expanded = permute(R, E, 32)
    x = expanded ^ key
    sbox_result = sbox_substitute(x)
    f_result = permute(sbox_result, P, 32)
    return R, L ^ f_result

def des_block(block, keys, decrypt=False):
    block = permute(block, IP, 64)
    L, R = block >> 32, block & 0xffffffff
    if decrypt:
        keys = keys[::-1]
    for k in keys:
        L, R = des_round(L, R, k)
    return permute((R << 32) | L, FP, 64)

def pad(data):
    pad_len = 8 - (len(data) % 8)
    return data + bytes([pad_len] * pad_len)

def unpad(data):
    return data[:-data[-1]]

def encrypt(data: bytes, key: bytes) -> bytes:
    key_int = int.from_bytes(key, 'big')
    subkeys = generate_keys(key_int)
    data = pad(data)
    return b''.join(
         des_block(int.from_bytes(data[i:i+8], 'big'), subkeys).to_bytes(8,
'big')
        for i in range(0, len(data), 8)
    )
```

```
def decrypt(data: bytes, key: bytes) -> bytes:
    key_int = int.from_bytes(key, 'big')
    subkeys = generate_keys(key_int)
    result = b''.join(
                    des_block(int.from_bytes(data[i:i+8],   'big'),   subkeys,
decrypt=True).to_bytes(8, 'big')
        for i in range(0, len(data), 8)
    )
    return unpad(result)
```

Листинг А.3 – Код файла main.py

```
# main.py

from DES import encrypt, decrypt
from secrets import token_bytes

def main():
    message = "Hello from DES Cipher!"
    key = token_bytes(8)

    print("Original message:", message)
    print("Key (hex):", key.hex())

    ciphertext = encrypt(message.encode(), key)
    print("Encrypted (hex):", ciphertext.hex())

    decrypted = decrypt(ciphertext, key)
    print("Decrypted:", decrypted.decode())

if __name__ == "__main__":
    main()
```