

Алгоритм S-DES

Алгоритм S-DES (упрощенный DES, разработан Edward Schaefer [1, 2]) имеет такую же структуру, как и алгоритм DES. Отличия только в значениях параметров алгоритма. В S-DES они имеют существенно меньшую размерность.

Алгоритм построен на основе сети Фейстеля (рис.1). Способ организации шифрования, предложенный Хорстом Фейстелем, позволяет посредством многократного применения относительно простых преобразований (замен и перестановок) добиться построения стойкого шифра, обладающего свойствами конфузии и диффузии.

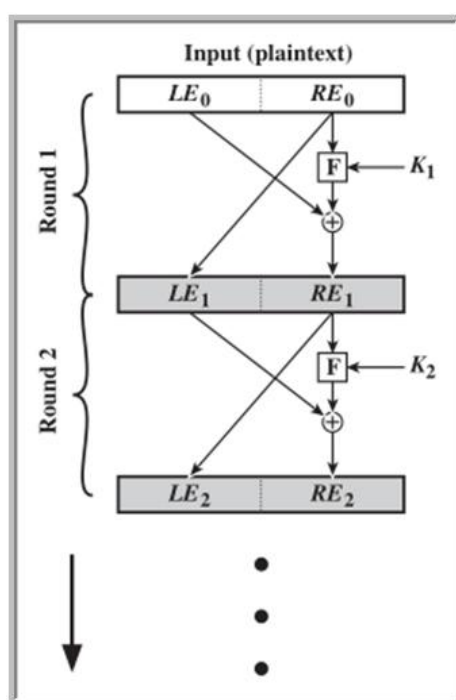


Рисунок 1 – сеть Фейстеля

Диффузия предполагает распространение влияния одного бита открытого блока на значительное количество бит зашифрованного блока. Наличие у шифра этого свойства позволяет скрыть статистическую

зависимость между битами открытого текста, а также не позволяет восстанавливать неизвестный ключ по частям.

Цель конфузии – сделать как можно более сложной зависимость между ключом и шифротекстом. Криптоаналитик на основе статистического анализа зашифрованного сообщения не должен получить сколько-нибудь значительного количества информации об использованном ключе.

Применение диффузии и конфузии по отдельности не обеспечивает необходимую стойкость, надёжная криптосистема получается только в результате их совместного использования.

На рис.2 приведена общая схема алгоритма S-DES, на которой показаны основные преобразования для шифрования и расшифрования 8-ми битового блока данных, а также представлен алгоритм формирования раундовых подключей.

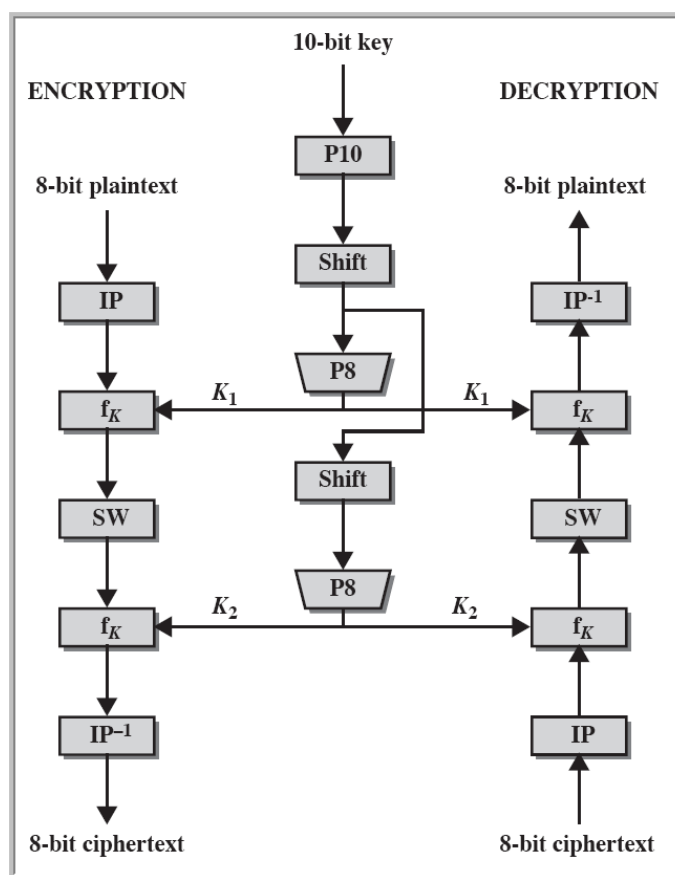


Рисунок 2

Особенностью шифров, основанных на сети Фейстеля, является использование одного и того же алгоритма как для шифрования так и для расшифрования. Отличие заключается только в порядке использования раундовых подключей.

На рис.3 показано, как используется сеть Фейстеля для построения шифра S-DES.

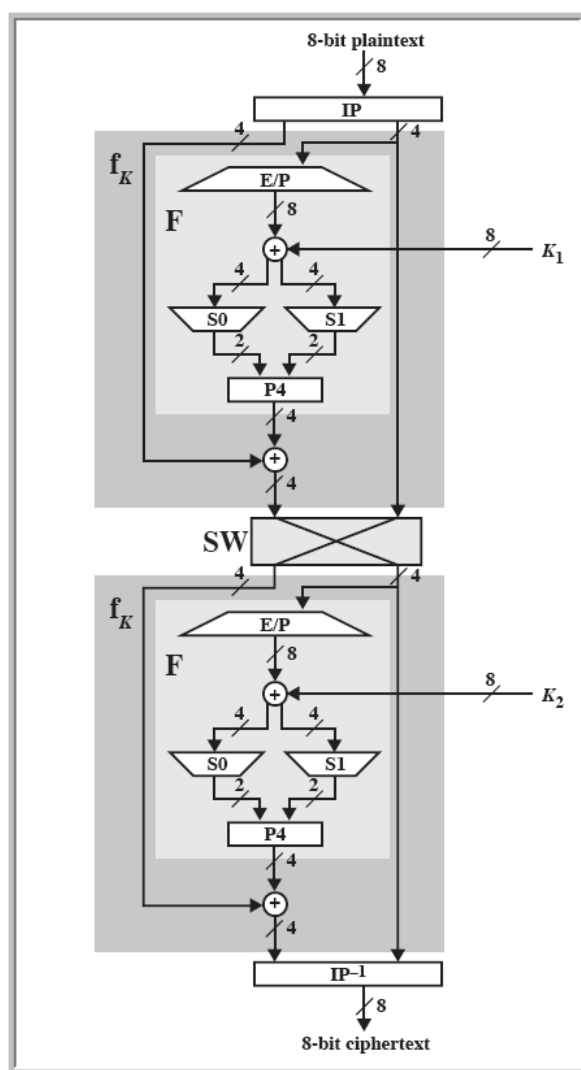


Рисунок 3

В шифре реализовано два раунда однотипных преобразований, состоящих из последовательного применения перестановок и замен. Преобразования применяются к 8-ми битовому блоку данных. Ниже

представлен код, в котором реализованы все замены и перестановки, которые применяются в этом шифре.

```
class SDes():
    P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
    P8 = [6, 3, 7, 4, 8, 5, 10, 9]
    LS1 = [2, 3, 4, 5, 1]
    LS2 = [3, 4, 5, 1, 2]
    IP = [2, 6, 3, 1, 4, 8, 5, 7]
    IPinv = [4, 1, 3, 5, 7, 2, 8, 6]
    EP = [4, 1, 2, 3, 2, 3, 4, 1]
    P4 = [2, 4, 3, 1]
    SW = [5, 6, 7, 8, 1, 2, 3, 4]

    # таблицы замен
    S0 = [[1, 0, 3, 2],
           [3, 2, 1, 0],
           [0, 2, 1, 3],
           [3, 1, 3, 2]]

    S1 = [[0, 1, 2, 3],
           [2, 0, 1, 3],
           [3, 0, 1, 0],
           [2, 1, 0, 3]]

    def __init__(self):
        """
        раундовые ключи. рассчитываются в функции key_schedule
        """
        self.k1 = 0
        self.k2 = 0

    @staticmethod
    def pbox(x, p, nx):
        # перестановка бит в nx-битовом числе x по таблице перестановок p
        y = 0
        np = len(p)
        for i in reversed(range(np)):
            if (x & (1 << (nx - 0 - p[i]))) != 0:
                y ^= (1 << (np - 1 - i))
        return y

    def p10(self, x):
        return self.pbox(x, self.P10, 10)

    def p8(self, x):
        return self.pbox(x, self.P8, 10)

    def p4(self, x):
        return self.pbox(x, self.P4, 4)

    def ip(self, x):
        return self.pbox(x, self.IP, 8)

    def ipinv(self, x):
        return self.pbox(x, self.IPinv, 8)

    def ep(self, x):
        return self.pbox(x, self.EP, 4)
```

```

def sw(self, x):
    return self.pbox(x, self.SW, 8)

def ls1(self, x):
    return self.pbox(x, self.LS1, 5)

def ls2(self, x):
    return self.pbox(x, self.LS2, 5)

@staticmethod
def divide_into_two(k, n):
    """
    функция разделяет n-битовое число k на два (n/2)-битовых числа
    """

    n2 = n//2
    mask = 2**n2 - 1
    l1 = (k >> n2) & mask
    l2 = k & mask
    return l1, l2

@staticmethod
def mux(l, r, n):
    """
    # l, r - n-битовые числа
    # возвращает число (2n-битовое), являющееся конкатенацией бит этих чисел
    """

    y = 0
    y ^= r
    y ^= l << n
    return y

@staticmethod
def apply_subst(x, s):
    """
    замена по таблице s
    """

    r = 2*(x >> 3) + (x & 1)
    c = 2*((x >> 2) & 1) + ((x >> 1) & 1)
    return s[r][c]

def s0(self, x):
    """
    замена по таблице s0
    """

    return self.apply_subst(x, self.S0)

def s1(self, x):
    """
    замена по таблице s1
    """

    return self.apply_subst(x, self.S1)

```

Функция **pbox**(x,p,nx) выполняет перестановку бит p в nx-битовом числе x.

Функции p10, p8, p4, ip, ip⁻¹, ep, sw выполняют конкретные перестановки бит в числе, являющимся аргументом функции. Перестановки заданы в виде следующих таблиц (рис.4).

P10									
3	5	2	7	4	10	1	9	8	6

P8							
6	3	7	4	8	5	10	9

IP							
2	6	3	1	4	8	5	7

IP ⁻¹							
4	1	3	5	7	2	8	6

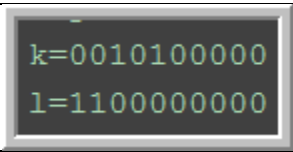
E/P							
4	1	2	3	2	3	4	1

P4			
2	4	3	1

Рисунок 4 – таблицы перестановок

Пример.

```
k = int('0010100000', 2)
l = sdes.p10(k)
print('k={}'.format(binary(k, 10)))
print('l={}'.format(binary(l, 10)))
```



```
k=0010100000
l=1100000000
```

В этом примере

```
# Prints out its argument in binary
def binary(x, k):
    return bin(x)[2:].zfill(k)
```

Как показано на рис.4 перестановка P10 задана числами: 3, 5, 2, 7, 4, 10, 1, 9, 8, 6. Рассмотрим, что означают эти значения:

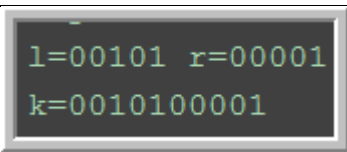
Индексы бит	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Значения бит в блоке	0, 0, 1, 0, 1, 0, 0, 0, 0, 0
Перестановка	3, 5, 2, 7, 4, 10, 1, 9, 8, 6
Результат перестановки	1, 1, 0, 0, 0, 0, 0, 0, 0, 0

Индексация бит в блоке начинается с 1 и формируется слева направо. Перестановка задает, какой бит будет под индексом 1, 2,

Функция **mux**(l, r, n) формирует из двух n-битовых чисел l и r одно 2n-битовое число.

Пример.

```
l = int('00101', 2)
r = int('00001', 2)
k = sdes.mux(l, r, 5)
```

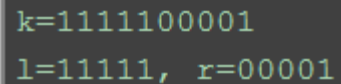


```
l=00101 r=00001
k=0010100001
```

Функция **divide_into_two**(k, n) формирует из n-битового числа k два числа размерности n/2. Первое число – значение, сформированное старшими битами числа k, второе число – значение, сформированное младшими битами числа k.

Пример:

```
k = int('1111100001', 2)
l, r = sdes.divide_into_two(k, 10)
print('k={}'.format(binary(k, 10)))
print('l={}, r={}'.format(binary(l, 5), binary(r, 5)))
```

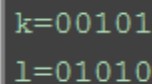


```
k=1111100001
l=11111, r=00001
```

Функция **ls1(x)** выполняет в 5-битовом числе x циклический сдвиг на 1 бит влево. Сдвиг реализован посредством применения соответствующей перестановки бит.

Пример:

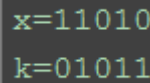
```
k = int('00101', 2)
l = sdes.ls1(k)
print('k={}'.format(binary(k, 5)))
print('l={}'.format(binary(l, 5)))
```



```
k=00101
l=01010
```

Функция **ls2(x)** выполняет в 5-битовом числе x циклический сдвиг на 2 бита влево. Сдвиг реализован посредством применения соответствующей перестановки бит.

```
x = int('11010', 2)
k = sdes.ls2(k)
print('x={}'.format(binary(x, 5)))
print('k={}'.format(binary(k, 5)))
```



```
x=11010
k=01011
```

Функция **apply_subst(x, s)** выполняет операцию замены числа x на число из таблицы замен s .

Функция **s0(self, x)** выполняет операцию замены по таблице замен $S0$. Четырехбитовое значение числа $x=(x_1, x_2, x_3, x_4)$ заменяется на двухбитовое значение, как показано на рис.5. Т.е. значения (x_1, x_4) формируют индекс строки, значения (x_2, x_3) формируют индекс столбца.

S_0		x_2	0	0	1	1
		x_3	0	1	0	1
x_1	x_4					
0	0		01	00	11	10
0	1		11	10	01	00
1	0		00	10	01	11
1	1		11	01	11	10

Рисунок 5

Пример.

```
x = int('0101', 2)
k = sdes.s0(x)
print('x={}'.format(binary(x, 4)))
print('k={}'.format(binary(k, 2)))
```

```
x=0101
k=01
```

Функция **s1**(self, x) выполняет операцию замены по таблице замен **S1**. Четырехбитовое значение числа $x=(x_1, x_2, x_3, x_4)$ заменяется на двухбитовое значение, как показано на рис.6.

S_1		x_2	0	0	1	1
		x_3	0	1	0	1
x_1	x_4					
0	0		00	01	10	11
0	1		10	00	01	11
1	0		11	00	01	00
1	1		10	01	00	11

Рисунок 6

Пример.

```
x = int('1110', 2)
k = sdes.s1(x)
print('x={}'.format(binary(x, 4)))
print('k={}'.format(binary(k, 2)))
```

```
x=1110
k=00
```

1. Написать функцию **key_schedule**(self, key), которая на основании 10-битового ключа key формирует два раундовых подключа в соответствии с алгоритмом расширения ключа, представленным в виде схемы на рис.7.

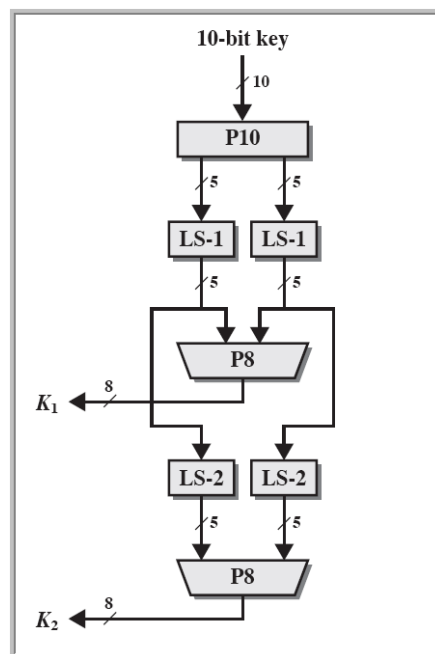


Рисунок 7 – Алгоритм генерации подключей

На рис.7 приведена последовательность преобразований, которые можно описать в виде следующих формул:

$$K_1 = P8(\text{Shift}(P10(\text{key})))$$

$$K_2 = P8(\text{Shift}(\text{Shift}(P10(\text{key}))))$$

Прототип функции:

```
def key_schedule(self, key):
    """
    Алгоритм расширения ключа. Функция формирует из ключа шифрования key два
    раундовых ключа self.k1, self.k2
    """
```

Для ключа $\text{key} = 0111111101$ результат обработки по алгоритму на рис.7 показан на рис.8.

```

After P10: 1111110011
After LS-1:11111 00111
After P8 (K1): 01011111
After LS-2:11111 11100
After P8 (K2): 11111100

```

Рисунок 8

2. Написать функцию **F**(self, block, k), которая выполняет обработку 4-х битового блока данных block с использованием раундового подключа k по схеме, приведенной на рис.9.

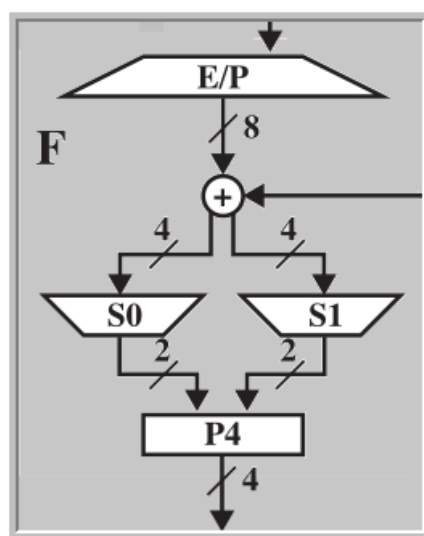


Рисунок 9

Прототип функции:

```

def F(self, block, k):
    # Inputs
    # block = 4 bits block data (int number)
    # k = 8 bits subkey (int number)
    # Outputs
    # Out=4 bits block data (int number)

```

Для block = 0011 и k = 01011111 результат обработки по алгоритму на рис.9 показан на рис.10.

```

After E/P: 10010110
After xor with subkey: 11001001
After S0: 01
After S1: 10
After P4: 1010

```

Рисунок 10

3. Написать функцию $f_k(\text{self}, \text{block}, \text{SK})$, которая выполняет обработку 8-ми битового блока данных block с использованием раундового 8-ми битового подключа SK . Вначале 8-ми битовый блок нужно разбить на две части – левую (L) и правую (R), затем выполнить обработку по формуле:

$$f_k(L, R) = (L \oplus F(R, SK), R)$$

где F – функция, реализованная в п.2.

На рис.11 данное уравнение представлено в виде схемы.

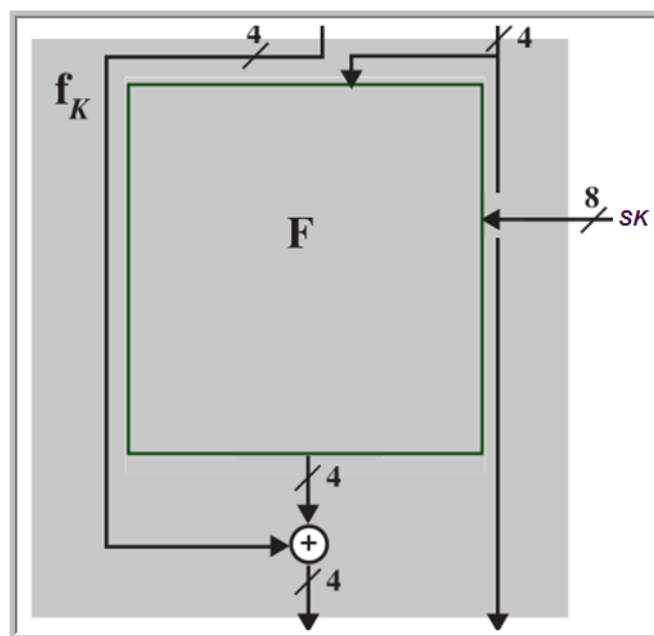
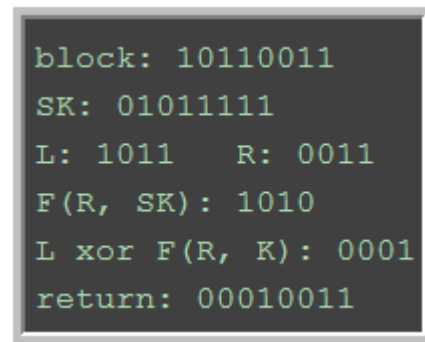


Рисунок 11

Прототип функции:

```
def f_k(self, block, SK):  
    # Inputs  
    # block = 8 bits block data (int number)  
    # SK = 8 bits subkey (int number)  
    # Outputs  
    # Out=8 bits block data (int number)
```

Для block=10110011 и SK=01011111 результат обработки в функции f_k приведен на рис.12.



```
block: 10110011  
SK: 01011111  
L: 1011    R: 0011  
F(R, SK): 1010  
L xor F(R, K): 0001  
return: 00010011
```

Рисунок 12

4. Написать функцию **sdes**(self, block, k1, k2), которая выполняет шифрование 8-ми битового блока данных block с раундовыми ключами k1, k2. Шифрование основано на алгоритме, который в виде схемы представлен на рис.13. Алгоритм состоит из последовательного применения 5 преобразований: начальная перестановка IP исходного 8-ми битового блока данных, преобразование f_k, перестановка SW (поменять местами левую и правую части блока), преобразование f_k (второй раунд), обратная перестановка к начальной IP^{-1} . В итоге получение зашифрованного блока данных можно представить в виде формулы:

$$\text{ciphertext} = IP^{-1} \left(f_{K_2} \left(SW \left(f_{K_1} \left(IP(\text{plaintext}) \right) \right) \right) \right)$$

Прототип функции:

```
def sdes(self, block, k1, k2):  
    # Inputs  
    # block = 8 bits block data (int number)  
    # K1 = 8 bits subkey (int number)  
    # K2 = 8 bits subkey (int number)  
    # Outputs  
    # Out=8 bits block data (int number)
```

Для block=11101010, k1=01011111, k2=11111100 результат обработки в функции sdes приведен на рис.14.

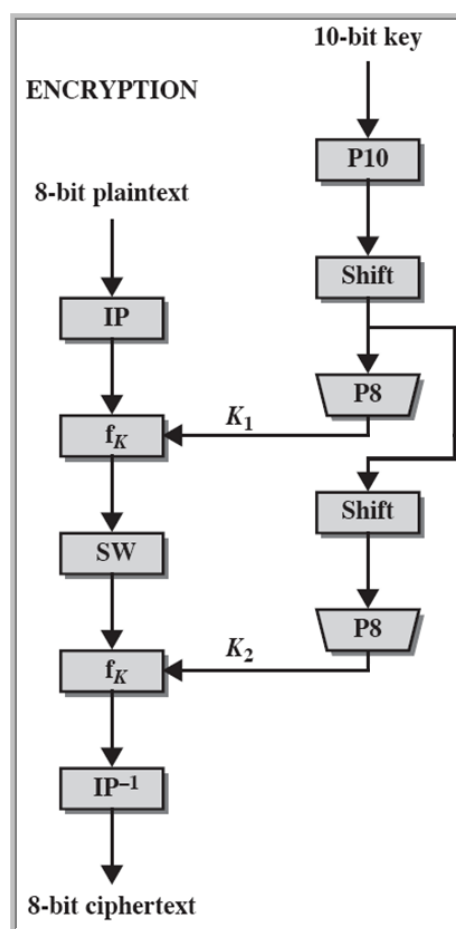


Рисунок 13

```

block: 11101010
K1: 01011111   K2: 11111100
After IP: 10110011
After f_k: 00010011
After SW: 00110001
After f_k: 00110001
After IPinv: 10100010

```

Рисунок 14

5. Написать функцию **encrypt_block**(self, plaintext_block), которая выполняет шифрование блока открытого сообщения (1 байт). Внутри функции надо вызвать функцию sdes с указанием раундовых ключей, которые участвуют в шифровании, как показано на рис.13.

6. Написать функцию **decrypt_block**(self, cipherext_block), которая выполняет расшифрование зашифрованного блока сообщения (1 байт). Внутри функции надо вызвать функцию sdes с указанием раундовых ключей, которые участвуют в расшифровании, как показано на рис.15. В виде формулы последовательность преобразований для расшифрования блока выглядит следующим образом:

$$\text{plaintext} = \text{IP}^{-1} \left(f_{K_1} \left(\text{SW} \left(f_{K_2} \left(\text{IP}(\text{cipherext}) \right) \right) \right) \right)$$

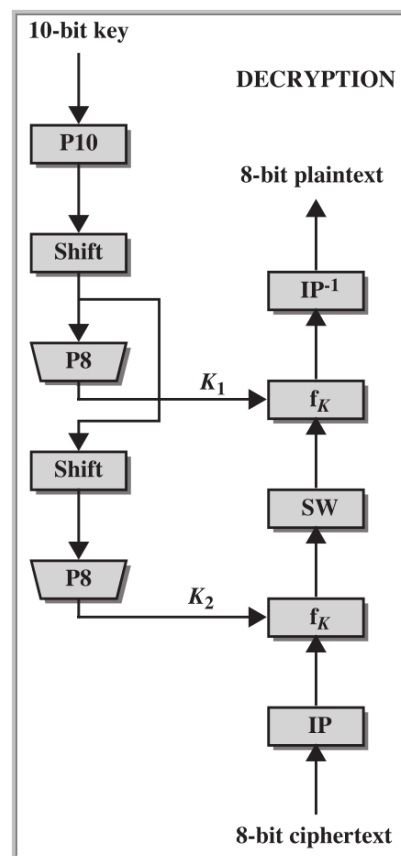


Рисунок 15

7. Написать функцию **encrypt_data()** и **decrypt_data()**, которые позволяют зашифровать и расшифровать массивы байт.

Например, для ключа $\text{key} = 0111111101$ результатом шифрования чисел из массива $[234, 54, 135, 98, 47]$ будет массив чисел $[162, 222, 0, 10, 83]$.

8. Лавинный эффект (avalanche) – это число бит, которое изменилось в зашифрованном блоке при изменении одного бита открытого блока или ключа. Чем больше лавинный эффект, тем выше надёжность шифра.

Заполнить таблицу для $\text{key} = 0111111101$:

Открытый блок	P1=0000 0000	P2=1000 0000
После первого раунда	P11 =	P21=
После второго раунда	P12=	P22=

Определить, насколько бит отличаются P11 и P21. Определить, насколько бит отличаются P22 и P12.

9. Заполнить таблицу для открытого блока=10100100:

ключ	K1=0111111101	K2=0011111101
После первого раунда	P11 =	P21=
После второго раунда	P12=	P22=

Определить, насколько бит отличаются P11 и P21. Определить, насколько бит отличаются P22 и P12.

10. Расшифровать файл aa1_sdes_c_all.bmp – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования ECB. Ключ равен 645. Зашифровать в режиме ECB, оставив первые 50 байт без изменения.

11. Расшифровать файл aa2_sdes_c_cbc_all.bmp – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования CBC. Ключ равен 845. Вектор инициализации равен 56. Зашифровать в режиме ECB и в режиме CBC, оставив первые 50 байт без изменения. Сравнить полученные изображения.

12. Дешифровать файл t15_sdes_c_cbc_all.txt – зашифрованное шифром S_DES сообщение в формате txt. Режим шифрования CBC. Известны младшие 8 бит ключа (11101001). Вектор инициализации равен 202.

13. Расшифровать файл aa3_sdes_c_ofb_all.bmp – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования OFB. Ключ равен 932. Вектор инициализации равен 234. Зашифровать в режиме ECB и в режиме OFB, оставив первые 50 байт без изменения. Сравнить полученные изображения.

14. Расшифровать файл `aa4_sdes_c_cfb_all.bmp` – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования CFB. Ключ равен 455. Вектор инициализации равен 162. Зашифровать в режиме ECB и в режиме CFB, оставив первые 50 байт без изменения. Сравнить полученные изображения.

15. Расшифровать файл `im38_sdes_c_ctr_all.bmp` – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования CTR. Ключ равен 572. Вектор инициализации равен 157. Зашифровать в режиме ECB и в режиме CTR, оставив первые 50 байт без изменения. Сравнить полученные изображения.

16. Добавить третий раунд, как показано на рис.16. Третий подключ сформировать аналогично первым двум, сделав сдвиг на три бита.

17. Выполнить пп. 8, 9 для трехраундового варианта. Сравнить полученные результаты.

18. Написать небольшой материал минимум по трем источникам по алгоритму DES. История создания, использования, взлома. Схемы, таблицы и т.п.

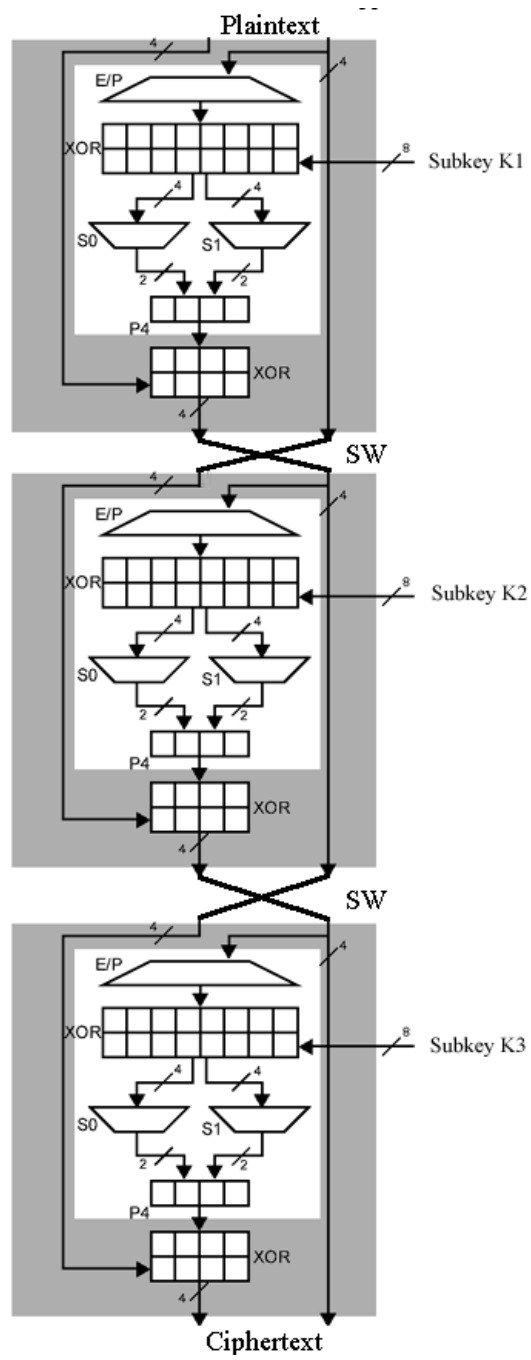


Рисунок 16

Литература

[1] Schaefer E, “A Simplified Data Encryption Standard Algorithm”, Cryptologia, Vol .20, No.1, pp. 77-84, 1996.

[2] Stallings W, “Cryptography And Network Security. Principles And Practice”, 5th Edition, 2011.