

Оглавление

Алгоритм S-DES	2
Алгоритм шифрования	4
Реализация	6
Перестановки	8
Замена	10
Сдвиг	11
Задание 1	12
Задание 2	13
Задание 3	14
Задание 4	15
Задание 5	17
Задание 6	17
Задание 7	18
Задание 8	18
Задание 9	19
Задание 10	19
Литература	19

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.

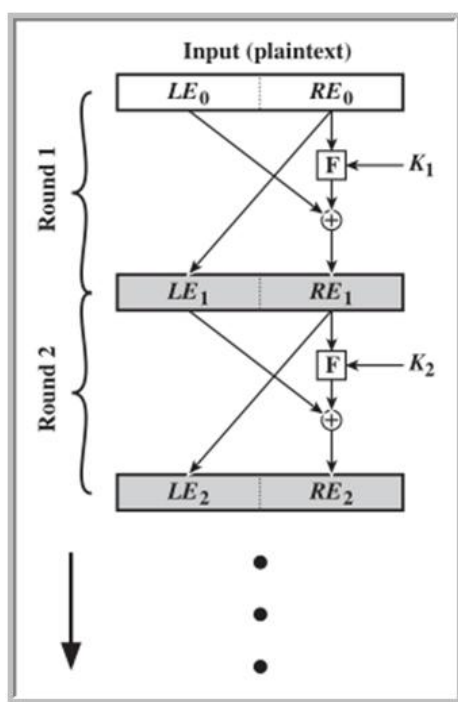


Рисунок 1 – сеть Фейстеля

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 двух раундовых 8-ми битовых ключей из 10-ти битового ключа.

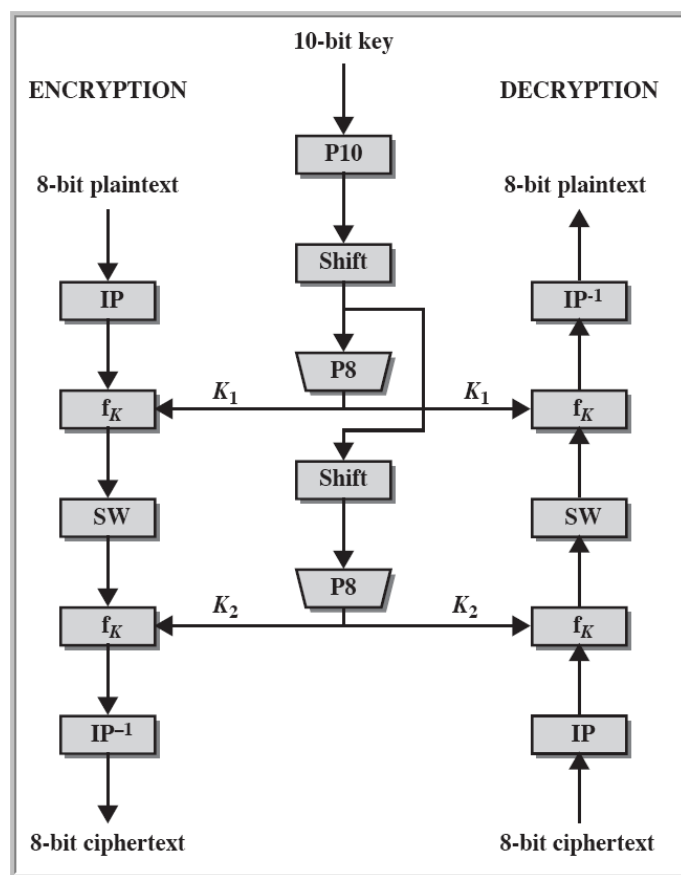


Рисунок 2

Алгоритм шифрования

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.

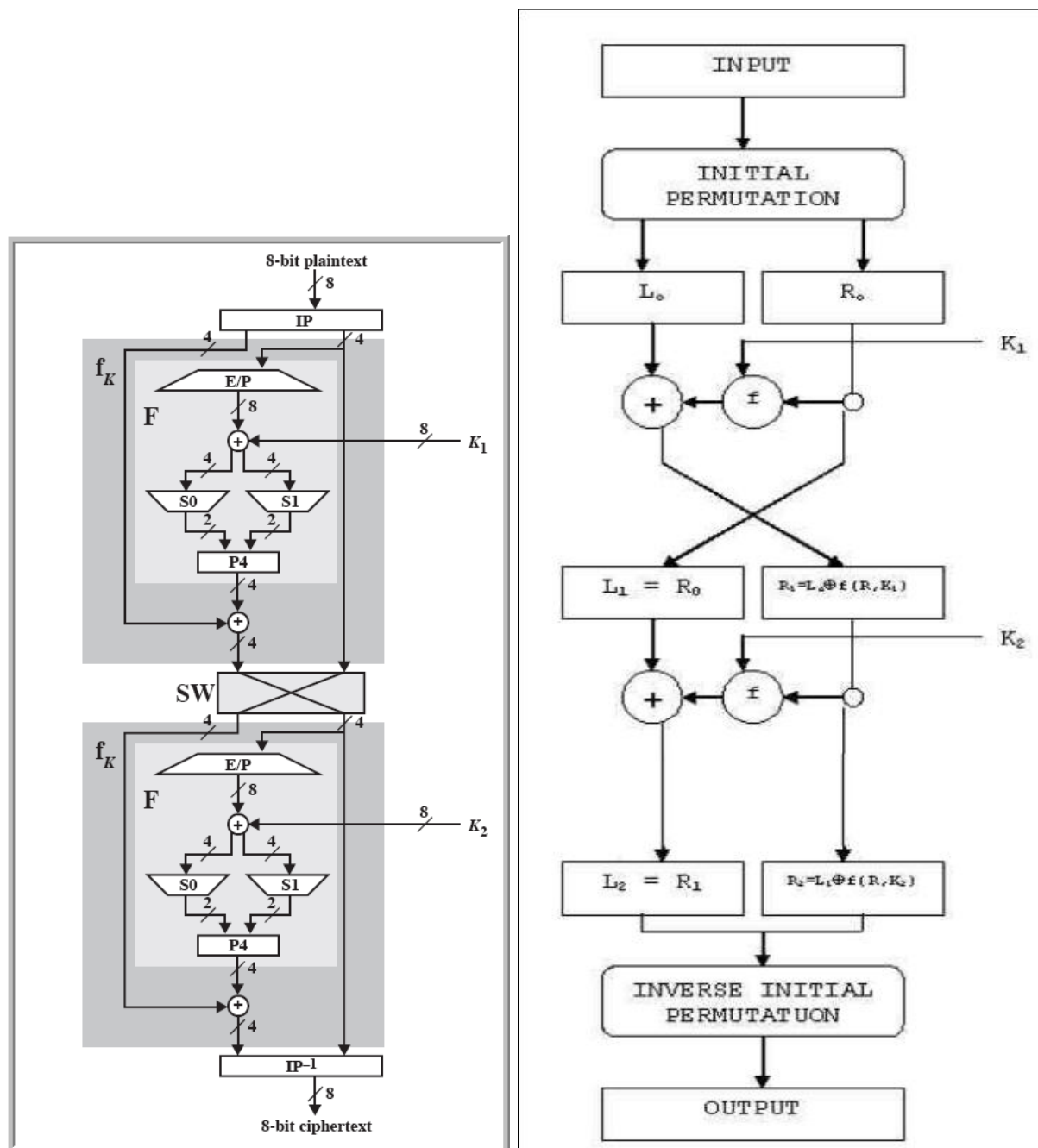


Рисунок 3

$$L_1 = R_0$$

$$R_1 = L_0 \oplus f(R, K_1)$$

после 1-раунда

$$L_2 = R_1$$

$$R_2 = L_1 \oplus f(R, K_2)$$

после 2-раунда

Как и в случае любой схемы шифрования, здесь на вход функции шифрования подаётся два типа данных – открытый текст, который требуется зашифровать, и ключ. В данном случае длина открытого текста предполагается равной 8 битам, а длина ключа – 10 битам.

В шифре реализовано два раунда однотипных преобразований, состоящих из последовательного применения перестановок и замен.

Процесс преобразования открытого текста состоит из трёх этапов.

I. Сначала 8-битный блок открытого текста поступает для обработки на вход начальной перестановки (IP), в результате чего, получаются переставленные входные данные.

II. Затем следует этап, состоящий из 2 раундов применения одной и той же функции, в которой используются операции перестановки и замены. На выходе 2 раунда получается 8-битная последовательность, являющаяся некоторой функцией открытого текста и ключа.

III. Полученная последовательность проходит через перестановку, обратную начальной (IP^{-1}), в результате чего получается 8-битный блок шифрованного текста.

В каждом раунде выполняется один шаг перемешивания (с использованием соответствующего раундового ключа и S-блоков замены), после которого следует шаг рассеивания, не зависящий от ключа.

Реализация

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)

```

Перестановки

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

Функции $p10$, $p8$, $p4$, ip , $ipinv$, 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

Например, для блока $k = 1001110000$ результатом применения перестановки P10 будет блок $l = P10(k) = 0100101001$ (рис.5).

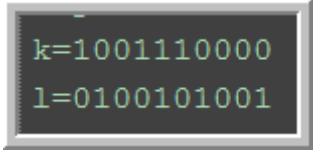
вход k	1	2	3	4	5	6	7	8	9	10
	1			1	1	1				
выход $P10(z)$	3	5	2	7	4	10	1	9	8	6
		1			1		1			1

Рисунок 5

На рис.5 пустая ячейка в таблице означает нуль. Индексация бит в блоке начинается с 1 и формируется слева направо. Перестановка задает, какой бит будет под индексом 1, 2,

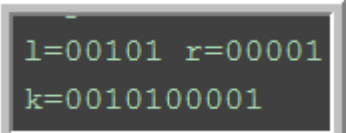
Закрашенные одинаковым цветом ячейки таблицы показывают перемещение соответствующих единиц в результате перестановки.

Пример

<pre>import s_des sdes = s_des.SDes() k = int('1001110000', 2) l = sdes.p10(k) print('k={}'.format(bin(k)[2:].zfill(10))) print('l={}'.format(bin(l)[2:].zfill(10)))</pre>	
---	--

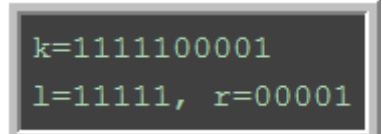
Функция **mux**(l, r, n) формирует из двух n-битовых чисел l и r одно 2n-битовое число.

Пример.

<pre>l = int('00101', 2) r = int('00001', 2) k = sdes.mux(l, r, 5)</pre>	
--	--

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.

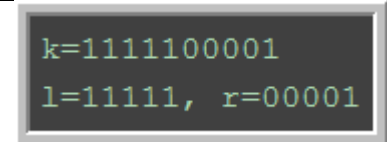
Пример:

<pre>k = int('1111100001', 2) l, r = sdes.divide_into_two(k, 10) print('k={}'.format(bin(k)[2:].zfill(10))) print('l={}, r={}'.format(bin(l)[2:].zfill(5), bin(r)[2:].zfill(5)))</pre>	
--	--

Если написать функция b2:

<pre>def b2(x, k): return bin(x)[2:].zfill(k)</pre>

то рассмотренный пример будет более компактным:

<pre> k = int('1111100001', 2) l, r = sdes.divide_into_two(k, 10) print('k={}'.format(b2(k, 10))) print('l={}, r={}'.format(b2(l, 5), b2(r, 5))) </pre>		
---	--	--

Замена

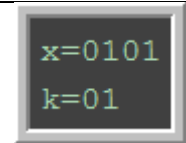
Функция **apply_subst**(x, s) выполняет операцию замены числа x на число из таблицы замен s.

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.6. Т.е. значения (x_1, x_4) формируют индекс строки, значения (x_2, x_3) формируют индекс столбца.

(x_1, x_2, x_3, x_4)									
S ₀									
				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 6

Пример:

<pre> x = int('0101', 2) k = sdes.s0(x) print('x={}'.format(b2(x, 4))) print('k={}'.format(b2(k, 2))) </pre>		
--	--	--

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.7.

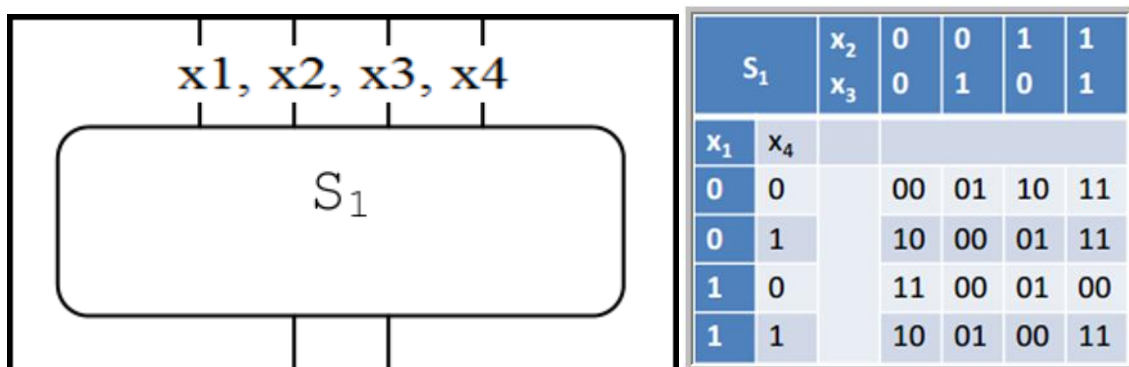


Рисунок 7

Пример:

<pre>x = int('1110', 2) k = sdes.s1(x) print('x={}'.format(b2(x, 4))) print('k={}'.format(b2(k, 2)))</pre>	<pre>x=1110 k=00</pre>
--	------------------------

Сдвиг

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

Пример:

<pre>k = int('00101', 2) l = sdes.ls1(k) print('k={}'.format(b2(k, 5))) print('l={}'.format(b2(l, 5)))</pre>	<pre>k=00101 l=01010</pre>
--	----------------------------

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

<pre>x = int('11010', 2) k = sdes.ls2(k) print('x={}'.format(b2(x, 5))) print('k={}'.format(b2(k, 5)))</pre>	<pre>x=11010 k=01011</pre>
--	----------------------------

Задание 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.8.

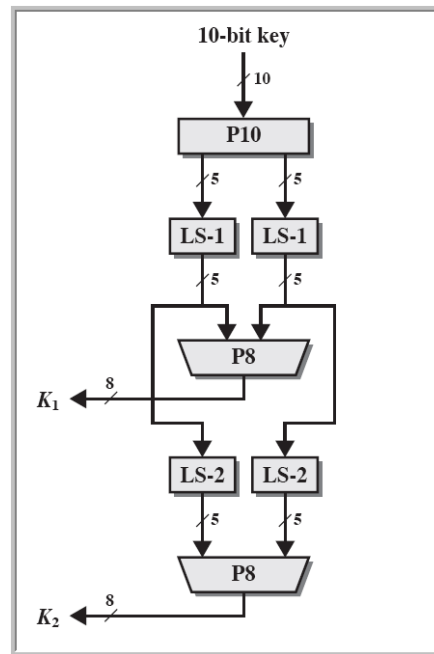


Рисунок 8 – Алгоритм генерации подключей

На рис.8 приведена последовательность преобразований, которые можно описать в виде следующих формул:

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

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

где P8, P10 – перестановки, Shift – циклический сдвиг влево на 1 бит.

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 `key = 0111111101` результат обработки по алгоритму на рис.8 показан на рис.9.

```

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

```

Рисунок 9

Задание 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.10.

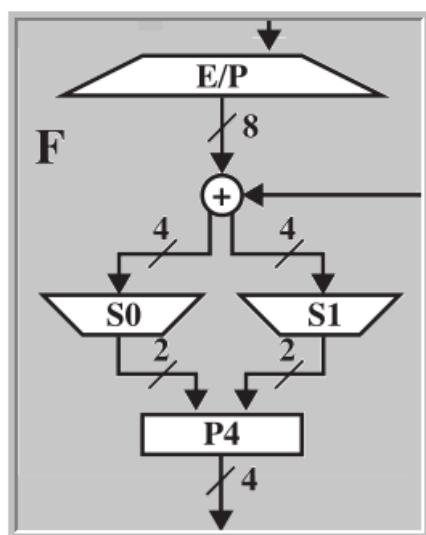


Рисунок 10 – Функция F в алгоритме на рис.3

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 значений block = 0011 и k = 01011111 результат обработки по алгоритму на рис.10 показан на рис.11.

```

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

```

Рисунок 11

Задание 3

Написать функцию $f_k(\text{self}, \text{block}, SK)$, 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 задании 2.

На рис.12 данное уравнение представлено в виде схемы.

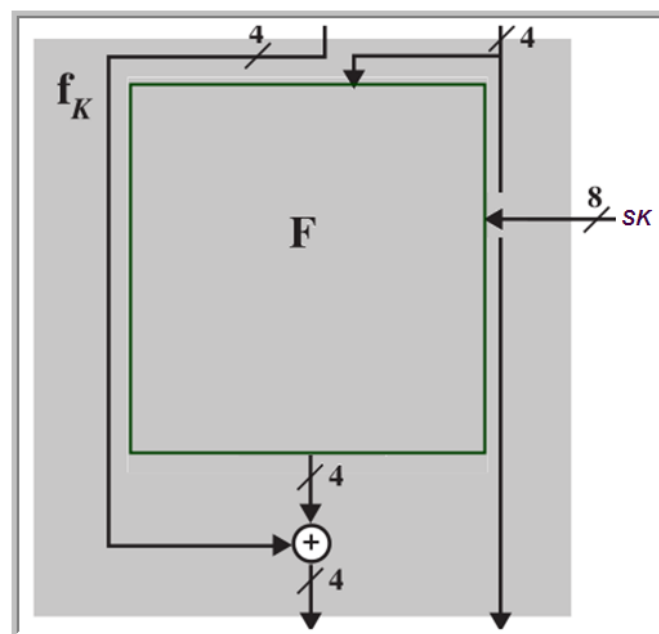
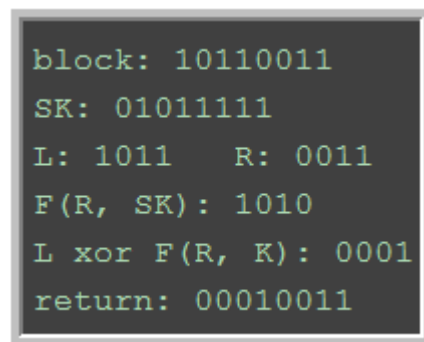


Рисунок 12

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 для значений block=10110011 и SK=01011111 результат обработки в функции f_k приведен на рис.13.



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

Рисунок 13

Задание 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.3, 14. 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 значений $\text{block}=11101010$, $k1=01011111$, $k2=11111100$ результат обработки в функции `sdes` приведен на рис.15.

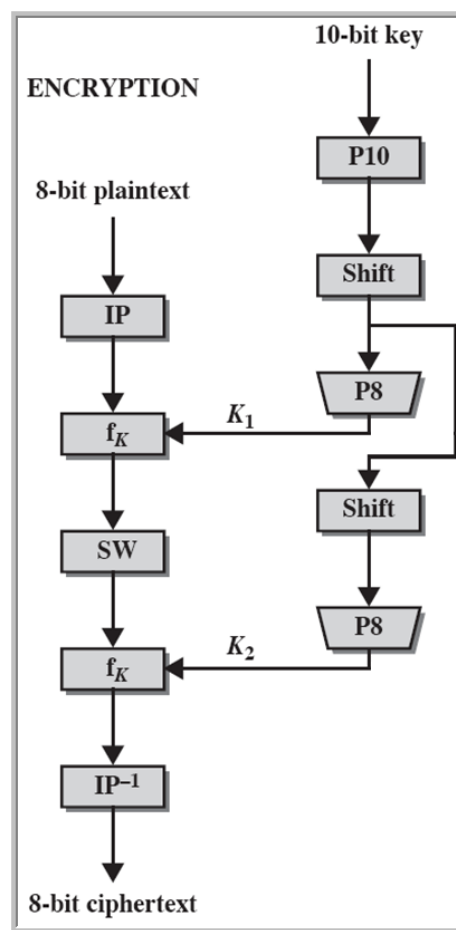


Рисунок 14


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

Рисунок 15

Задание 5

Написать функцию **encrypt** (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.14.

Задание 6

Написать функцию **decrypt** (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.16. В виде формулы последовательность преобразований для расшифрования блока выглядит следующим образом:

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

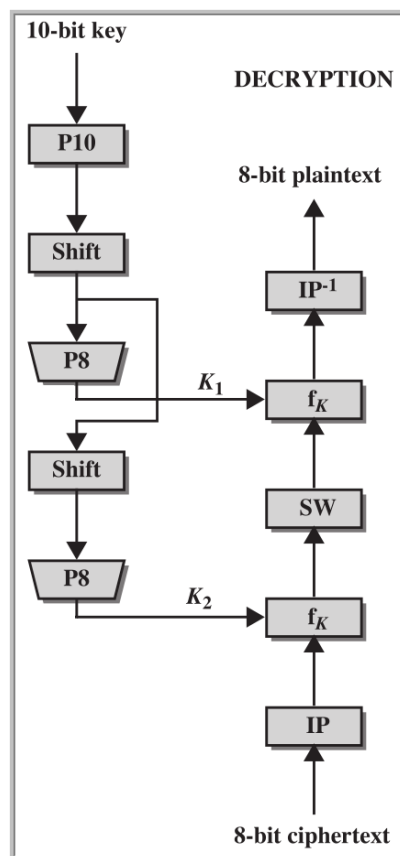


Рисунок 16

Задание 7

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

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

Задание 8

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

Задание 9

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.

Задание 10

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.

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.