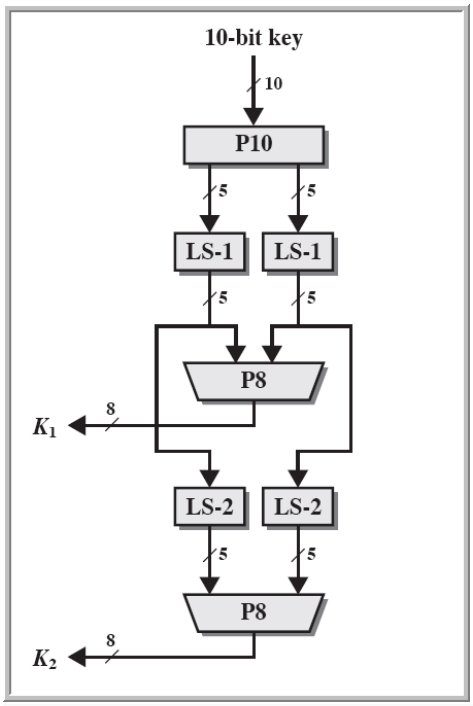
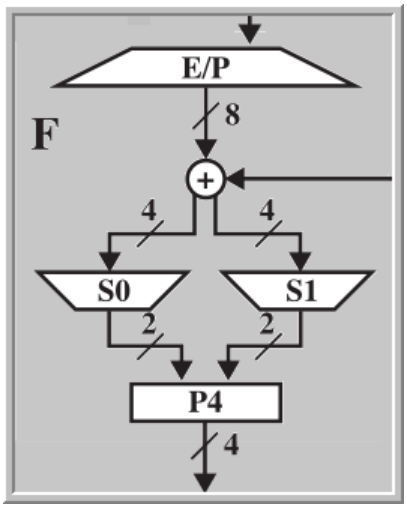
1. Написать функцию key_schedule(self, key), которая на основании 10-битового ключа key формирует два раундовых подключа.



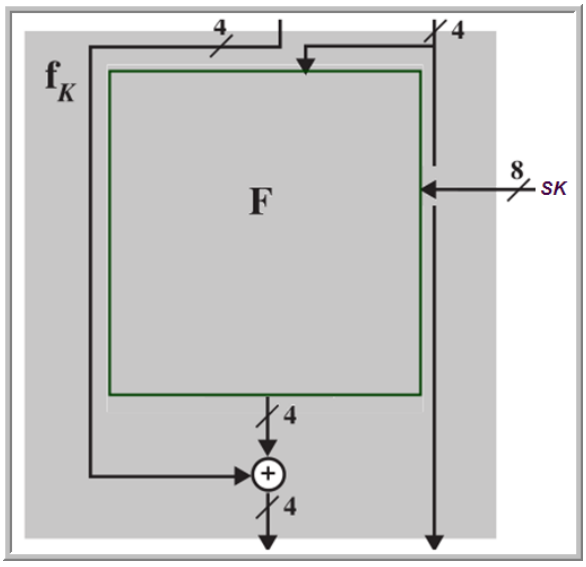

```
def key_schedule(self, key):
    """
    Алгоритм расширения ключа. Функция формирует из ключа шифрования key два
    раундовых ключа self.k1, self.k2
    """
    p_10  = self.p10(key)
    ls_1_left = self.ls1(self.divide_into_two(p_10, 10)[0])
    ls_1_right = self.ls1(self.divide_into_two(p_10, 10)[1])
    self.k1 = self.p8(self.mux(ls_1_left, ls_1_right, 5))

    ls_2_left = self.ls2(ls_1_left)
    ls_2_right = self.ls2(ls_1_right)
    self.k2 = self.p8(self.mux(ls_2_left, ls_2_right, 5))
```

2. Написать функцию F(self, block, k), которая выполняет обработку 4-х битового блока данных block с использованием раундового подключа k.



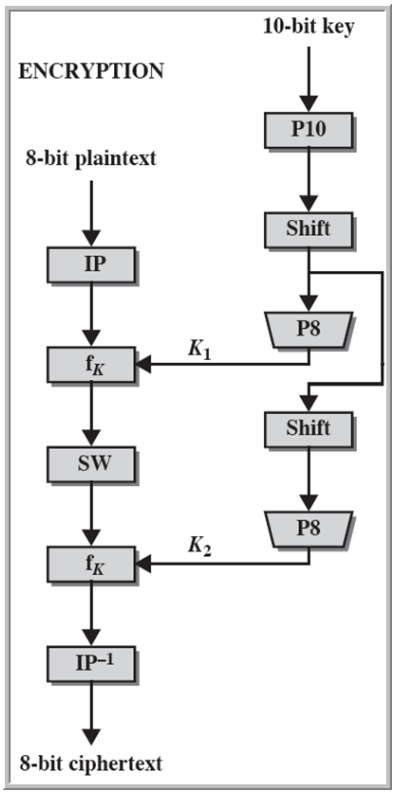

```
def F(self, block, k):
    """
    Функция выполняет обработку 4-х битового блока данных block
    с использованием раундового подключа k
    """
    e_p = self.ep(block)
    xor = e_p ^ k
    xor_left = self.divide_into_two(xor, 8)[0]
    xor_right = self.divide_into_two(xor, 8)[1]

    s_0 = self.s0(xor_left)
    s_1 = self.s1(xor_right)
    p_4 = self.p4(self.mux(s_0, s_1, 2))

    return p_4
```

3. Написать функцию f_k(self, block, SK), которая выполняет обработку 8-ми битового блока данных block с использованием раундового 8-ми битового подключа SK.




```
def f_k(self, block, SK):
    """
    Функция выполняет обработку 8-ми битового блока данных block
    с использованием раундового 8-ми битного подключа SK
    """
    block_left = self.divide_into_two(block, 8)[0]
    block_right = self.divide_into_two(block, 8)[1]
    f = self.F(block_right, SK)
    xor = block_left ^ f
    res = self.mux(xor, block_right, 4)

    return res
```

4. Написать функцию sdes(self, block, k1, k2), которая выполняет шифрование 8-ми битового блока данных block с раундовыми ключами k1, k2.




```
def sdes(self, block, k1, k2):
    """
    Выполняет шифрование 8-ми битового блока данных block
    с раундовыми ключами k1, k2
    """
    block_ip = self.ip(block)
    fk1 = self.f_k(block_ip, k1)
    block_sw = self.sw(fk1)
    fk2 = self.f_k(block_sw, k2)
    res = self.ipinv(fk2)

    return res
```

5. Написать функцию encrypt_block(self, plaintext_block), которая выполняет шифрование блока открытого сообщения (1 байт); decrypt_block(self, cipherext_block), которая выполняет расшифрование зашифрованного блока сообщения (1 байт); encrypt_data(); decrypt_data().

```
def encrypt_block(self, plaintext_block):
    return self.sdes(plaintext_block, self.k1, self.k2)

def decrypt_block(self, ciphertext_block):
    return self.sdes(ciphertext_block, self.k2, self.k1)

def encrypt_data(self, data):
    res = []
    for x in data:
        res.append(self.encrypt_block(x))
    return res

def decrypt_data(self, data):
    res = []
    for x in data:
        res.append(self.decrypt_block(x))
    return res
```

6. Key = 0111111101, data = [234, 54, 135, 98, 47].
   Результат:

```
[162, 222, 0, 10, 83]
```

7. Лавинный эффект. Key = 0111111101.

Открытый блок	P1 = 0000 0000	P2 = 1000 0000	Отличаются на _ бит
После первого раунда	P11 = 0000 1110	P21 = 0000 1111	1
После второго раунда	P12 = 0011 1001	P22 = 0101 1011	3

8. Лавинный эффект. Data = 10100100.

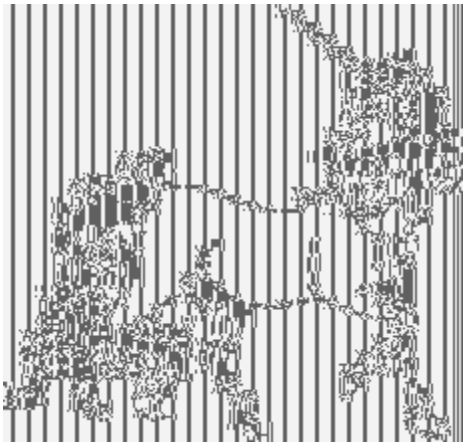
Ключ	K1 = 0111111101	K2 = 0011111101	Отличаются на _ бит
После первого раунда	P11 = 0000 1001	P21 = 0000 1001	0
После второго раунда	P12 = 1011 0110	P22 = 1001 0110	1

9. 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.

```
key = 645
data = read_write_file.read_data_1byte('encrypt_data/aa1_sdes_c_all.bmp')

sdes_d = em.ecb_d(data, key, crypto_mode='sdes')
read_write_file.write_data_1byte('decrypt_data/aa1_sdes_c_all_decrypt.bmp', sdes_d)

data = read_write_file.read_data_1byte('decrypt_data/aa1_sdes_c_all_decrypt.bmp')
sdes_e50 = em.ecb_e(data[50:], key, crypto_mode='sdes')
read_write_file.write_data_1byte('encrypt_data/aa1_sdes_c_all_50.bmp', data[:50] + sdes_e50)
```

Результат:

Оригинал

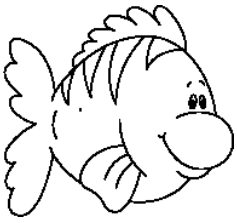


Режим ECB

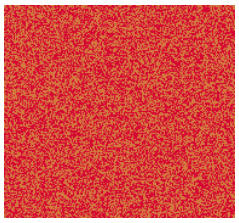
10. Расшифровать файл aa2_sdes_c_cbc_all.bmp – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования CBC. Ключ равен 845. Вектор инициализации равен 56. Зашифровать в режиме ECB и в режиме CBC, оставив первые 50 байт без изменения. Сравнить полученные изображения.

```
key = 845
iv = 56
data = read_write_file.read_data_1byte('encrypt_data/aa2_sdes_c_cbc_all.bmp')

sdes_d = em.cbc_d(data, key, iv, crypto_mode='sdes')
read_write_file.write_data_1byte('decrypt_data/aa2_sdes_c_cbc_all_decrypt.bmp', sdes_d)

data = read_write_file.read_data_1byte('decrypt_data/aa2_sdes_c_cbc_all_decrypt.bmp')
sdes_e50 = em.cbc_e(data[50:], key, iv, crypto_mode='sdes')
read_write_file.write_data_1byte('encrypt_data/aa2_sdes_c_cbc_all_50.bmp', data[:50] + sdes_e50)
```

Результат:



Оригинал



Режим CBC

11. 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.

```
iv = 202
data = read_write_file.read_data_1byte('encrypt_data/t15_sdes_c_cbc_all.txt')
k_last8 = int('11101001', 2)

for p in range(4):
    key = int(bin(p)[2:] + bin(k_last8)[2:], 2)
    sdes_d = em.cbc_d(data, key, iv, crypto_mode='sdes')

    txt = ''.join([chr(s) for s in sdes_d])
    is_english = detectEnglish.isEnglish(txt)
    if is_english:
        print('Detect english text with key:', key)
        read_write_file.write_data_1byte('decrypt_data/t15_sdes_c_cbc_all_decrypt.txt', sdes_d)
```

Key = 1001.

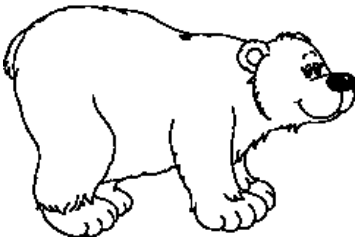
Результат:

```
His spirit inspired me with great respect. He seemed to have no strength,
and he never once hit me hard, and he was always knocked down; but, he
would be up again in a moment, sponging himself or drinking out of the
water-bottle, with the greatest satisfaction in seconding himself according
to form, and then came at me with an air and a show that made me believe he
really was going to do for me at last. He got heavily bruised, for I am
sorry to record that the more I hit him, the harder I hit him; but, he came
up again and again and again, until at last he got a bad fall with the back
of his head against the wall. Even after that crisis in our affairs, he got
up and turned round and round confusedly a few times, not knowing where I
was; but finally went on his knees to his sponge and threw it up: at the
same time panting out, "That means you have won."

He seemed so brave and innocent, that although I had not proposed the
contest I felt but a gloomy satisfaction in my victory. Indeed, I go so far
as to hope that I regarded myself while dressing, as a species of savage
young wolf, or other wild beast. However, I got dressed, darkly wiping my
sanguinary face at intervals, and I said, "Can I help you?" and he said "No
thankee," and I said "Good afternoon," and he said "Same to you."

When I got into the court-yard, I found Estella waiting with the keys. But,
she neither asked me where I had been, nor why I had kept her waiting; and
there was a bright flush upon her face, as though something had happened to
delight her. Instead of going straight to the gate, too, she stepped back
into the passage, and beckoned me.

"Come here! You may kiss me, if you like."

I kissed her cheek as she turned it to me. I think I would have gone
through a great deal to kiss her cheek. But, I felt that the kiss was given
to the coarse common boy as a piece of money might have been, and that it
was worth nothing.
```

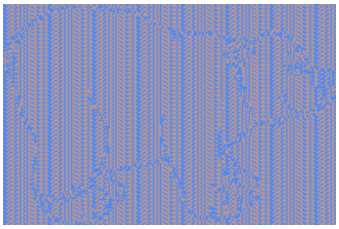
12. Расшифровать файл aa3_sdes_c_ofb_all.bmp – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования OFB. Ключ равен 932. Вектор инициализации равен 234. Зашифровать в режиме ECB и в режиме OFB, оставив первые 50 байт без изменения. Сравнить полученные изображения.

```
key = 932
iv = 234
data = read_write_file.read_data_1byte('encrypt_data/aa3_sdes_c_ofb_all.bmp')

sdes_d = em.ofb_d(data, key, iv, crypto_mode='sdes')
read_write_file.write_data_1byte('decrypt_data/aa3_sdes_c_ofb_all_decrypt.bmp', sdes_d)

data = read_write_file.read_data_1byte('decrypt_data/aa3_sdes_c_ofb_all_decrypt.bmp')
sdes_ecb50 = em.ecb_e(data[50:], key, crypto_mode='sdes')
sdes_ofb50 = em.ofb_e(data[50:], key, iv, crypto_mode='sdes')
read_write_file.write_data_1byte('encrypt_data/aa3_sdes_c_ecb_all_50.bmp', data[:50] + sdes_ecb50)
read_write_file.write_data_1byte('encrypt_data/aa3_sdes_c_ofb_all_50.bmp', data[:50] + sdes_ofb50)
```

Результат:


Оригинал


Режим ECB

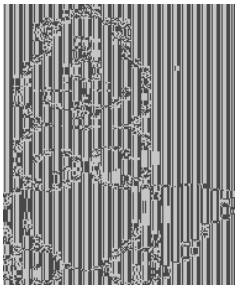

Режим OFB

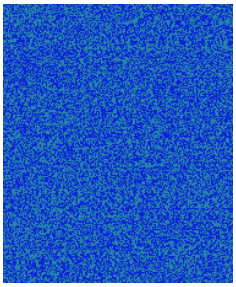
13. Расшифровать файл aa4_sdes_c_cfb_all.bmp – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования CFB. Ключ равен 455. Вектор инициализации равен 162. Зашифровать в режиме ECB и в режиме CFB, оставив первые 50 байт без изменения. Сравнить полученные изображения.

```
key = 455
iv = 162
data = read_write_file.read_data_1byte('encrypt_data/aa4_sdes_c_cfb_all.bmp')

sdes_d = em.cfb_d(data, key, iv, crypto_mode='sdes')
read_write_file.write_data_1byte('decrypt_data/aa4_sdes_c_cfb_all_decrypt.bmp', sdes_d)

data = read_write_file.read_data_1byte('decrypt_data/aa4_sdes_c_cfb_all_decrypt.bmp')
sdes_ecb50 = em.ecb_e(data[50:], key, crypto_mode='sdes')
sdes_cfb50 = em.cfb_e(data[50:], key, iv, crypto_mode='sdes')
read_write_file.write_data_1byte('encrypt_data/aa4_sdes_c_ecb_all_50.bmp', data[:50] + sdes_ecb50)
read_write_file.write_data_1byte('encrypt_data/aa4_sdes_c_cfb_all_50.bmp', data[:50] + sdes_cfb50)
```

Результат:


Оригинал


Режим ECB


Режим CFB

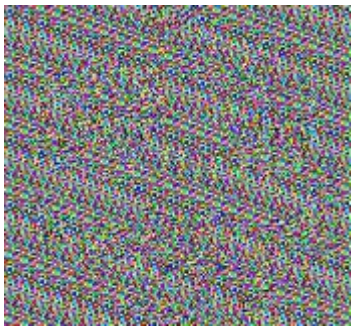
14. Расшифровать файл im38_sdes_c_ctr_all.bmp – зашифрованное шифром S_DES изображение в формате bmp. Режим шифрования CTR. Ключ равен 572. Вектор инициализации равен 157. Зашифровать в режиме ECB и в режиме CTR, оставив первые 50 байт без изменения. Сравнить полученные изображения.

```
key = 572
iv = 157
data = read_write_file.read_data_1byte('encrypt_data/im38_sdes_c_ctr_all.bmp')

sdes_d = em.ctr_d(data, key, iv, crypto_mode='sdes')
read_write_file.write_data_1byte('decrypt_data/im38_sdes_c_ctr_all_decrypt.bmp', sdes_d)

data = read_write_file.read_data_1byte('decrypt_data/im38_sdes_c_ctr_all_decrypt.bmp')
sdes_ecb50 = em.ecb_e(data[50:], key, crypto_mode='sdes')
sdes_ctr50 = em.ctr_e(data[50:], key, iv, crypto_mode='sdes')
read_write_file.write_data_1byte('encrypt_data/im38_sdes_c_ecb_all_50.bmp', data[:50] + sdes_ecb50)
read_write_file.write_data_1byte('encrypt_data/im38_sdes_c_ctr_all_50.bmp', data[:50] + sdes_ctr50)
```

Результат:

Оригинал


Режим ECB


Режим CTR

15. Добавить третий раунд. Третий подключ сформировать аналогично первым двум, сделав сдвиг на три бита.

```
LS3 = [4, 5, 1, 2, 3]
def ls3(self, x):
    return self.pbox(x, self.LS3, 5)
```

```
def key_schedule3(self, key):
    """
    Алгоритм расширения ключа. Функция формирует из ключа шифрования key два
    раундовых ключа self.k1, self.k2
    """
    p_10  = self.p10(key)
    ls_1_left = self.ls1(self.divide_into_two(p_10, 10)[0])
    ls_1_right = self.ls1(self.divide_into_two(p_10, 10)[1])
    self.k1 = self.p8(self.mux(ls_1_left, ls_1_right, 5))

    ls_2_left = self.ls2(ls_1_left)
    ls_2_right = self.ls2(ls_1_right)
    self.k2 = self.p8(self.mux(ls_2_left, ls_2_right, 5))

    ls_3_left = self.ls3(ls_2_left)
    ls_3_right = self.ls3(ls_2_right)
    self.k3 = self.p8(self.mux(ls_3_left, ls_3_right, 5))
```

```
def sdes3(self, block, k1, k2, k3):
    """
    Выполняет шифрование 8-ми битового блока данных block
    с раундовыми ключами k1, k2
    """
    block_ip = self.ip(block)
    fk1 = self.f_k(block_ip, k1)
    block_sw = self.sw(fk1)

    fk2 = self.f_k(block_sw, k2)
    block_sw = self.sw(fk2)

    fk3 = self.f_k(block_sw, k3)
    res = self.ipinv(fk3)

    return res
```

16. Выполнить пп. 7, 8 для трехраундового варианта. Сравнить полученные результаты.

Лавинный эффект. Key = 0111111101.

Открытый блок	P1 = 0000 0000	P2 = 1000 0000	Отличаются на _ бит
После первого раунда	P11 = 0000 1110	P21 = 0000 1111	1
После второго раунда	P12 = 1110 0010	P22 = 1111 1000	3
После третьего раунда	P13 = 1010 1000	P23 = 0011 0100	4

Лавинный эффект. Data = 10100100.

Ключ	K1 = 0111111101	K2 = 0011111101	Отличаются на _ бит
После первого раунда	P11 = 0000 1001	P21 = 0000 1001	0
После второго раунда	P12 = 1001 0111	P22 = 1001 0101	1
После третьего раунда	P13 = 0010 1011	P23 = 0100 0111	4