

## Криптосистема Хилла.

Создадим файл hill.py и реализуем следующие функции:

1. Умножение матрицы 2x2 на вектор 2x1 по модулю.

```
def mult_AP_mod(A, P, mod=256):  
    return [(A[0][0]*P[0] + A[0][1]*P[1]) % mod, (A[1][0]*P[0] + A[1][1]*P[1]) % mod]
```

2. Вычисление обратной матрицы 2x2 по модулю m.

Рассмотрим пример для матрицы  $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ .

Вычислим определитель матрицы A:  $\Delta = \det(A) = a * d - b * c$ .

Если  $\gcd(|\Delta|, \text{mod}) = 1$ , т.е. абсолютное значение определителя является взаимно простым со значением модуля, то обратную матрицу можно найти по следующей формуле:

$$A^{-1} = \Delta^{-1} * \begin{bmatrix} d \bmod m & -b \bmod m \\ -c \bmod m & a \bmod m \end{bmatrix}, \text{ где}$$

$\Delta^{-1}$  — обратное значение по умножению к  $\Delta$  по модулю m (вычисляется по расширенному алгоритму Евклида).

Расширенный алгоритм Евклида:

```
def get_inverse_mod(a, mod):  
    # Returns the modular inverse of a % mod, which is  
    # the number x such that a*x % mod = 1  
  
    if gcd(a, mod) != 1:  
        return None  
  
    # Calculate using the Extended Euclidean Algorithm:  
    u1, u2, u3 = 1, 0, a  
    v1, v2, v3 = 0, 1, mod  
    while v3 != 0:  
        q = u3 // v3  
        v1, v2, v3, u1, u2, u3 = (u1 - q * v1), (u2 - q * v2), (u3 - q * v3), v1, v2, v3  
    return u1 % mod
```

Реализуем функцию для вычисления обратной матрицы по модулю m:

```
def get_inverse_matrix_mod(A, mod=256):  
    d = A[0][0]*A[1][1] - A[0][1]*A[1][0]  
    inv_d = affine.get_inverse_mod(abs(d), mod)  
    return [[(A[1][1]*inv_d) % mod, (-A[0][1]*inv_d) % mod],  
            [(-A[1][0]*inv_d) % mod, (A[0][0]*inv_d) % mod]]
```

3. Реализуем функции encrypt\_data и decrypt\_data.

```
def ed(data, A, mod=256):  
    res = []  
    for i in range(0, len(data), 2):  
        x = mult_AP_mod(A, [data[i], data[i+1]], mod)  
        res.append(x[0])  
        res.append(x[1])  
    return res  
  
def dd(data, A, mod=256):  
    res = []  
    inv_A = get_inverse_matrix_mod(A, mod)  
    for i in range(0, len(data), 2):  
        x = mult_AP_mod(inv_A, [data[i], data[i+1]], mod)  
        res.append(x[0])  
        res.append(x[1])  
    return res
```

1. Расшифровать файл im3\_hill\_c\_all.bmp. Ключ  $K = [[189, 58], [21, 151]]$

```
key = [[189, 58], [21, 151]]
encrypt_data = read_write_file.read_data_1byte('encrypt_data/im3_hill_c_all.bmp')
decrypt = hill.dd(encrypt_data, key)
read_write_file.write_data_1byte('decrypt_data/im3_hill_c_all_decrypt.bmp', decrypt)
```

Результат:



2. Дешифровать b4\_hill\_c\_all.png.

Известно, что первые четыре байта в любом png-файле: 137, 80, 78, 71.

Посмотрим первые четыре байта у зашифрованной картинке: 23, 3, 239, 52.

Чтобы дешифровать картинку нужен ключ. Пусть  $K = \begin{bmatrix} K_1 & K_2 \\ K_3 & K_4 \end{bmatrix}$  – искомый ключ.

Тогда, картинка шифровалась следующим образом:

$$\begin{bmatrix} K_1 & K_2 \\ K_3 & K_4 \end{bmatrix} * \begin{bmatrix} 137 \\ 80 \end{bmatrix} = \begin{bmatrix} 23 \\ 3 \end{bmatrix} \text{ и } \begin{bmatrix} K_1 & K_2 \\ K_3 & K_4 \end{bmatrix} * \begin{bmatrix} 78 \\ 71 \end{bmatrix} = \begin{bmatrix} 239 \\ 52 \end{bmatrix}$$

Распишем в линейные уравнения:

$$\begin{cases} K_1 * 137 + K_2 * 80 = 23 \\ K_3 * 137 + K_4 * 80 = 3 \end{cases} \text{ и } \begin{cases} K_1 * 78 + K_2 * 71 = 239 \\ K_3 * 78 + K_4 * 71 = 52 \end{cases}$$

Теперь сгруппируем:

$$\begin{cases} K_1 * 137 + K_2 * 80 = 23 \\ K_1 * 78 + K_2 * 71 = 239 \end{cases} \text{ и } \begin{cases} K_3 * 137 + K_4 * 80 = 3 \\ K_3 * 78 + K_4 * 71 = 52 \end{cases}$$

То есть получили:

$$\begin{bmatrix} 137 & 80 \\ 78 & 71 \end{bmatrix} * \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} = \begin{bmatrix} 23 \\ 239 \end{bmatrix} \text{ и } \begin{bmatrix} 137 & 80 \\ 78 & 71 \end{bmatrix} * \begin{bmatrix} K_3 \\ K_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 52 \end{bmatrix}$$

Выражаем неизвестные, домножив обе части равенства на матрицу обратную к  $\begin{bmatrix} 137 & 80 \\ 78 & 71 \end{bmatrix}$  по

модулю 256. Обратная матрица =  $\begin{bmatrix} 89 & 80 \\ 175 & 251 \end{bmatrix}$ .

$$\begin{bmatrix} K_1 \\ K_2 \end{bmatrix} = \begin{bmatrix} 89 & 80 \\ 175 & 251 \end{bmatrix} * \begin{bmatrix} 23 \\ 239 \end{bmatrix} = \begin{bmatrix} 175 \\ 251 \end{bmatrix} \text{ и } \begin{bmatrix} K_3 \\ K_4 \end{bmatrix} = \begin{bmatrix} 89 & 80 \\ 175 & 251 \end{bmatrix} * \begin{bmatrix} 3 \\ 52 \end{bmatrix} = \begin{bmatrix} 75 \\ 214 \end{bmatrix}$$

В итоге искомый ключ выглядит так:

$$K = \begin{bmatrix} 175 & 251 \\ 75 & 214 \end{bmatrix}$$

Теперь напишем программу, которая расшифрует заданный файл:

```
encrypt_data = read_write_file.read_data_1byte('encrypt_data/b4_hill_c_all.png')
a = [[137, 80], [78, 71]]
b = encrypt_data[:4]
inv_a = hill.get_inverse_matrix_mod(a)
key = [hill.mult_AP_mod(inv_a, [b[0], b[2]]),
       hill.mult_AP_mod(inv_a, [b[1], b[3]])]
print(key)
decrypt_data = hill.dd(encrypt_data, key)
read_write_file.write_data_1byte('decrypt_data/b4_hill_c_all_decrypt.png', decrypt_data)
```

Результат:



3. Дешифровать файл text2\_hill\_c\_all.txt.

Известно, что текст в файле начинается со слова Whose.

Рассуждения аналогичны пункту 2. Возьмем первые четыре буквы: W, h, o, s; и возьмем значения этих букв из таблицы символов ASCII, тем самым получим числовые значения и, повторив расчеты, получим ключ  $K = \begin{bmatrix} 167 & 94 \\ 233 & 121 \end{bmatrix}$ .

Реализация:

```
encrypt_data = read_write_file.read_data_1byte('encrypt_data/text2_hill_c_all.txt')
a = [[ord('W'), ord('h')], [ord('o'), ord('s')]]
b = encrypt_data[:4]
inv_a = hill.get_inverse_matrix_mod(a)
key = [hill.mult_AP_mod(inv_a, [b[0], b[2]]),
       hill.mult_AP_mod(inv_a, [b[1], b[3]])]
print(key)
decrypt_data = hill.dd(encrypt_data, key)
read_write_file.write_data_1byte('decrypt_data/text2_hill_c_all_decrypt.txt', decrypt_data)
```

Результат:

```
1  Whose woods these are I think I know.
2  His house is in the village, though;
3  He will not see me stopping here
4  To watch his woods fill up with snow.
5  My little horse must think it queer
6  To stop without a farmhouse near
7  Between the woods and frozen lake
8  The darkest evening of the year.
9
10 He gives his harness bells a shake
11 To ask if there is some mistake.
12 The only other sound's the sweep
13 Of easy wind and downy flake.
14 The woods are lovely, dark and deep,
15 But I have promises to keep,
16 And miles to go before I sleep,
17 And miles to go before I sleep.
```

4. Используя расширенный алгоритм Евклида, найти обратное значение по умножению для 1234 по mod 4321.

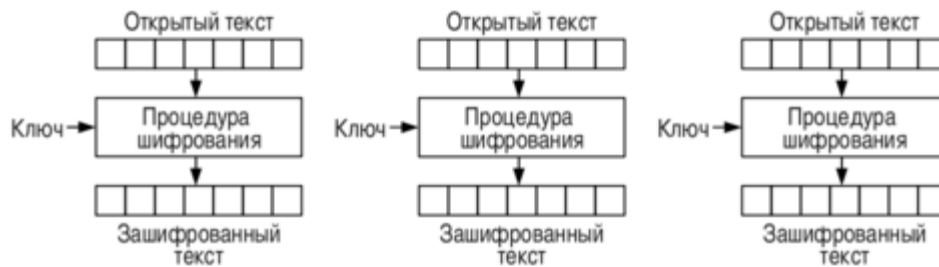
q	u1	u2	u3	v1	v2	v3
q=u3 // v3	v1(prev)	v2(prev)	v3(prev)	u1 - q * v1	u2 - q * v2	u3 - q * v3
	1	0	1234	0	1	4321
	0	1	4321	1	0	1234
3	1	0	1234	-3	1	619
1	-3	1	619	4	-1	615
1	4	-1	615	-7	2	4
153	-7	2	4	1075	-307	3
1	1075	-307	3	-1082	309	1
3	-1082	309	1	4321	-1234	0

-1082 mod 4321 = 3239.

## Режимы шифрования.

Реализуем файл encryption\_modes.py.

### 1. Режим ECB.



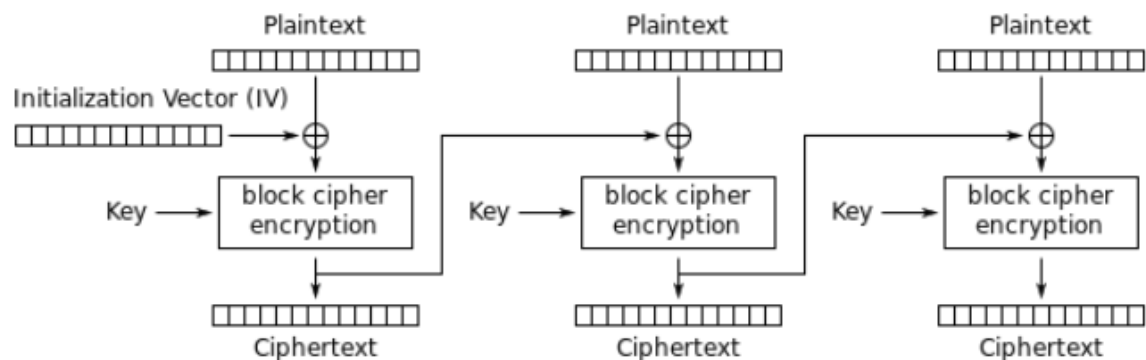
Реализация функций шифрования и расшифрования:

```
def ecb_e(data, key, mod=256, crypto_mode='caesar'):
    if crypto_mode == 'caesar': return caesar.ed(data, key, mod)

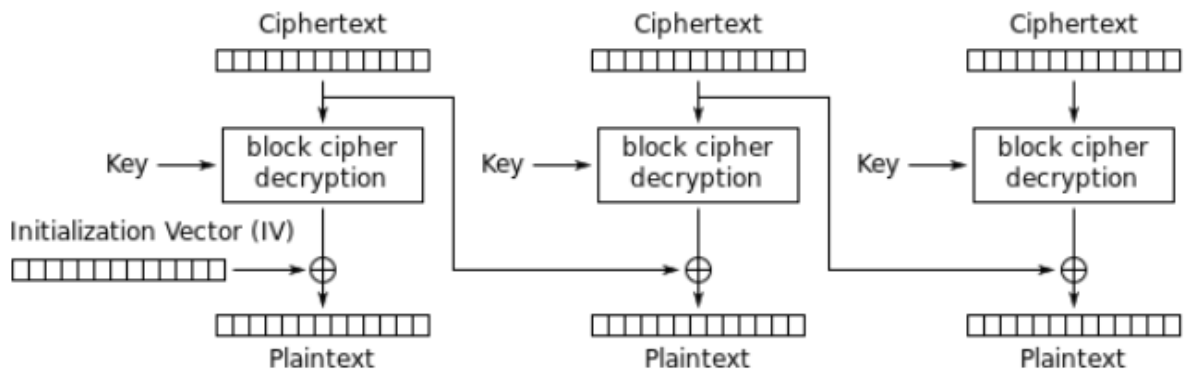
def ecb_d(data, key, mod=256, crypto_mode='caesar'):
    if crypto_mode == 'caesar': return caesar.dd(data, key, mod)
```

### 2. Режим CBC.

Шифрование:



Расшифровывание:



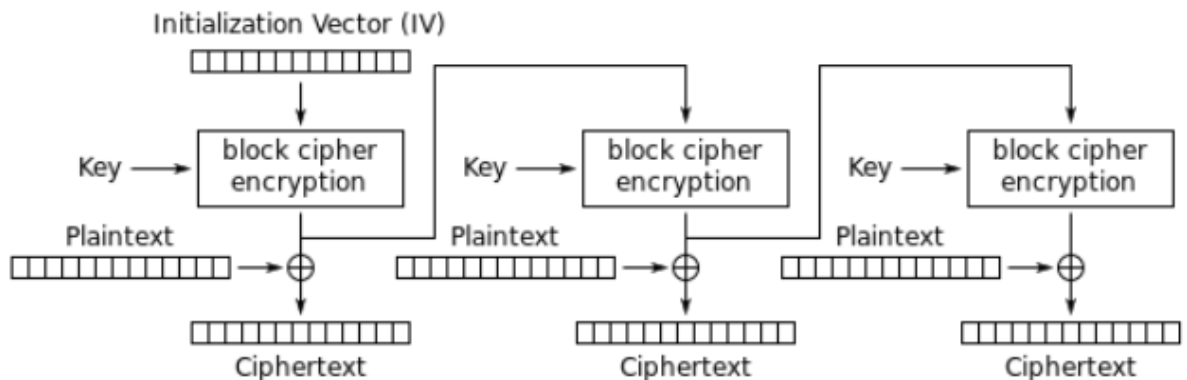
Реализация:

```
def cbc_e(data, key, iv, mod=256, crypto_mode='caesar'):
    res = []
    a = data[0] ^ iv
    if crypto_mode == 'caesar': a = caesar.e(a, key, mod)
    elif crypto_mode == 'vigenere': a = vigenere.e(a, ord(key[0]), mod)
    elif crypto_mode == 'affine': a = affine.e(a, key[0], key[1], mod)
    res.append(a)
    for i in range(1, len(data)):
        a = data[i] ^ res[i-1]
        if crypto_mode == 'caesar': a = caesar.e(a, key, mod)
        elif crypto_mode == 'vigenere': a = vigenere.e(a, ord(key[i % len(key)]), mod)
        elif crypto_mode == 'affine': a = affine.e(a, key[0], key[1], mod)
        res.append(a)
    return res

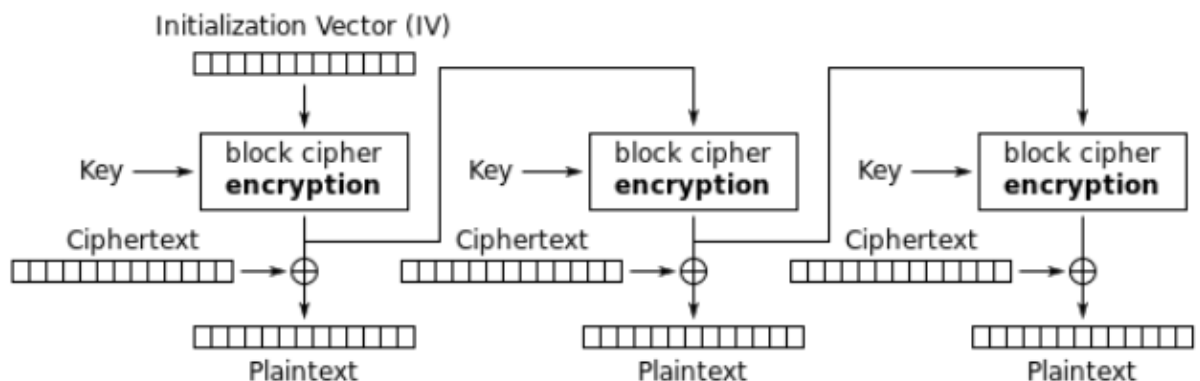
def cbc_d(data, key, iv, mod=256, crypto_mode='caesar'):
    res = []
    if crypto_mode == 'caesar': a = caesar.d(data[0], key, mod)
    elif crypto_mode == 'vigenere': a = vigenere.d(data[0], ord(key[0]), mod)
    elif crypto_mode == 'affine': a = affine.d(data[0], key[0], key[1], mod)
    a ^= iv
    res.append(a)
    for i in range(1, len(data)):
        if crypto_mode == 'caesar': a = caesar.d(data[i], key, mod)
        elif crypto_mode == 'vigenere': a = vigenere.d(data[i], ord(key[i % len(key)]), mod)
        elif crypto_mode == 'affine': a = affine.d(data[i], key[0], key[1], mod)
        a ^= data[i-1]
        res.append(a)
    return res
```

### 3. Режим OFB.

Шифрование:



Расшифровывание:



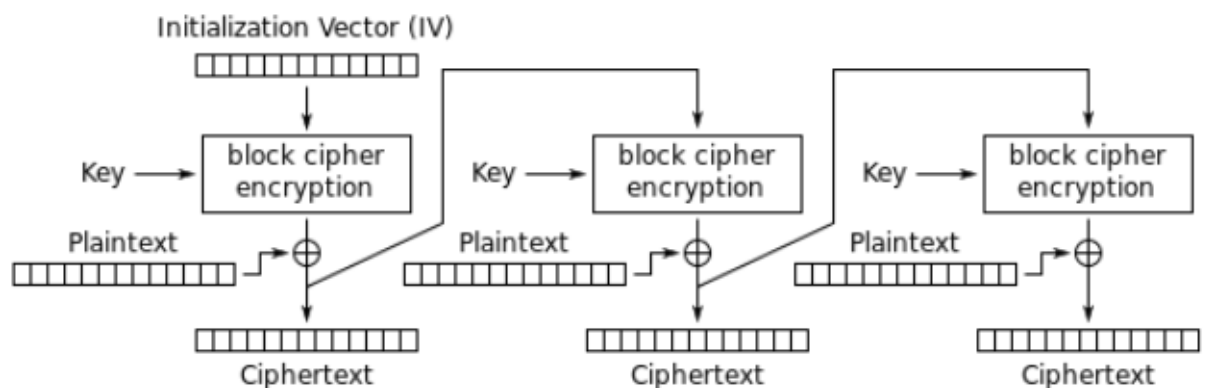
Реализация:

```
def ofb_e(data, key, iv, mod=256, crypto_mode='caesar'):
    res = []
    if crypto_mode == 'caesar': a = caesar.e(iv, key, mod)
    elif crypto_mode == 'vigenere': a = vigenere.e(iv, ord(key[0]), mod)
    elif crypto_mode == 'affine': a = affine.e(iv, key[0], key[1], mod)
    res.append(a ^ data[0])
    for i in range(1, len(data)):
        if crypto_mode == 'caesar': a = caesar.e(a, key, mod)
        elif crypto_mode == 'vigenere': a = vigenere.e(a, ord(key[i % len(key)]), mod)
        elif crypto_mode == 'affine': a = affine.e(a, key[0], key[1], mod)
        res.append(a ^ data[i])
    return res

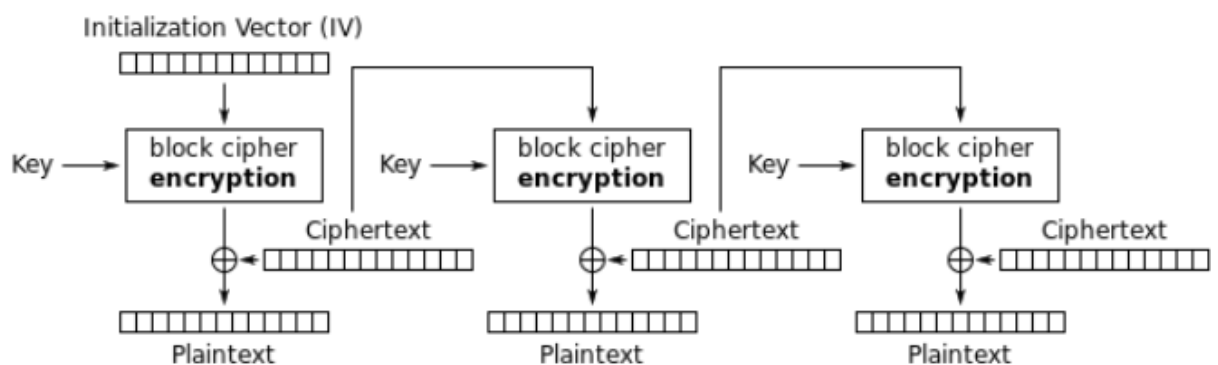
def ofb_d(data, key, iv, mod=256, crypto_mode='caesar'):
    return ofb_e(data, key, iv, mod=mod, crypto_mode=crypto_mode)
```

#### 4. Режим CFB.

Шифрование:



Расшифровывание:



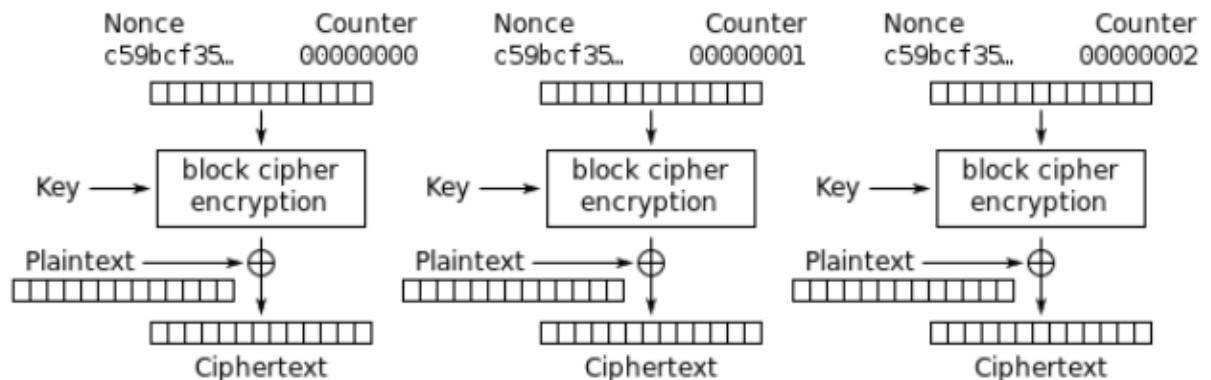
Реализация:

```
def cfb_e(data, key, iv, mod=256, crypto_mode='caesar'):
    res = []
    if crypto_mode == 'caesar': a = caesar.e(iv, key, mod)
    elif crypto_mode == 'vigenere': a = vigenere.e(iv, ord(key[0]), mod)
    elif crypto_mode == 'affine': a = affine.e(iv, key[0], key[1], mod)
    a ^= data[0]
    res.append(a)
    for i in range(1, len(data)):
        if crypto_mode == 'caesar': a = caesar.e(a, key, mod)
        elif crypto_mode == 'vigenere': a = vigenere.e(a, ord(key[i % len(key)]), mod)
        elif crypto_mode == 'affine': a = affine.e(a, key[0], key[1], mod)
        a ^= data[i]
        res.append(a)
    return res

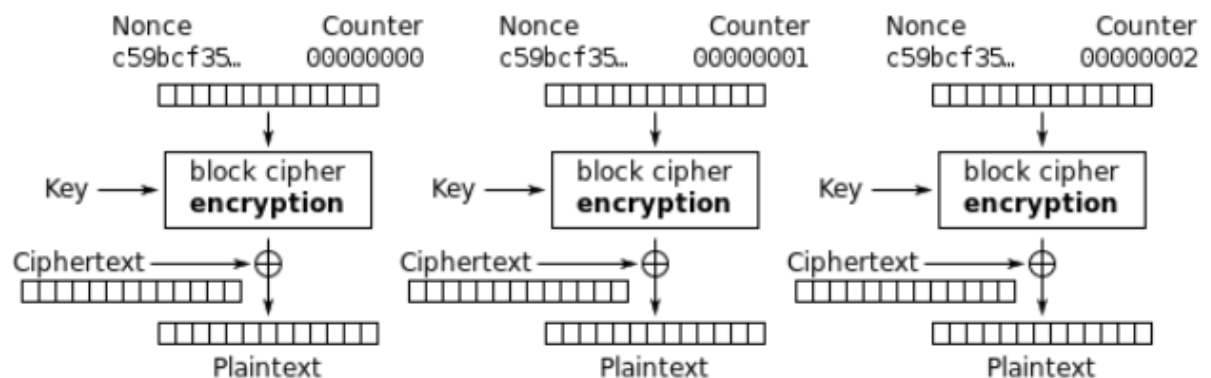
def cfb_d(data, key, iv, mod=256, crypto_mode='caesar'):
    res = []
    if crypto_mode == 'caesar': a = caesar.e(iv, key, mod) # encrypt!
    elif crypto_mode == 'vigenere': a = vigenere.e(iv, ord(key[0]), mod)
    elif crypto_mode == 'affine': a = affine.e(iv, key[0], key[1], mod)
    res.append(a ^ data[0])
    for i in range(1, len(data)):
        if crypto_mode == 'caesar': a = caesar.e(data[i-1], key, mod)
        elif crypto_mode == 'vigenere': a = vigenere.e(data[i-1], ord(key[i % len(key)]), mod)
        elif crypto_mode == 'affine': a = affine.e(data[i-1], key[0], key[1], mod)
        res.append(a ^ data[i])
    return res
```

## 5. Режим CTR.

Шифрование:



Расшифрование:





Реализация:

```
def ctr_e(data, key, counter=0, mod=256, crypto_mode='caesar'):
    res = []
    for i, x in enumerate(data):
        if crypto_mode == 'caesar': a = caesar.e(counter, key, mod)
        elif crypto_mode == 'vigenere': a = vigenere.e(counter, ord(key[i % len(key)]), mod)
        elif crypto_mode == 'affine': a = affine.e(counter, key[0], key[1], mod)
        res.append(a ^ x)
        counter += 1
    return res

def ctr_d(data, key, counter=0, mod=256, crypto_mode='caesar'):
    return ctr_e(data, key, counter=counter, mod=mod, crypto_mode=crypto_mode)
```

1. Расшифровать файл im7\_caesar\_cbc\_c\_all.bmp не удалось.

Поэтому используем другой файл с тем же ключом и вектором инициализации.

Key = 123, iv = 5. Зашифровать в режиме ECB и CBC, оставив первые 50 байт без изменения.

Шифр Цезаря.

Реализация:

```
key = 123
iv = 5
data = read_write_file.read_data_1byte('decrypt_data/f3_cbc_d.bmp')
ecb_e = em.ecb_e(data[50:], key, crypto_mode='caesar')
cbc_e = em.cbc_e(data[50:], key, iv, crypto_mode='caesar')

read_write_file.write_data_1byte('encrypt_data/f3_ecb_e_c_50.bmp', data[:50] + ecb_e)
read_write_file.write_data_1byte('encrypt_data/f3_cbc_e_c_50.bmp', data[:50] + cbc_e)
```

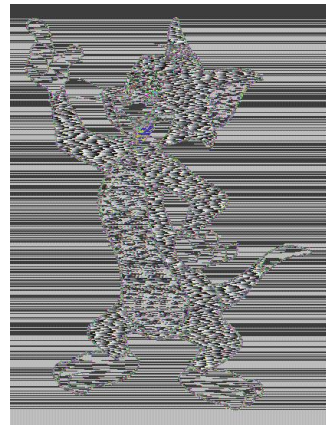
Результат:



Оригинал



Режим ECB



Режим CBC



2. Расшифровать файл `im8_caesar_ofb_c_all.bmp` не удалось.  
Поэтому используем другой файл с тем же ключом и вектором инициализации.  
`Key = 56`, `iv = 9`. Зашифровать в режиме ECB и OFB, оставив первые 50 байт без изменения.  
Шифр Цезаря.

Реализация:

```
key = 56
iv = 9
data = read_write_file.read_data_1byte('data/f4.bmp')
ecb_e = em.ecb_e(data[50:], key, crypto_mode='caesar')
ofb_e = em.ofb_e(data[50:], key, iv, crypto_mode='caesar')
read_write_file.write_data_1byte('encrypt_data/f4_ecb_c_e_50.bmp', data[:50] + ecb_e)
read_write_file.write_data_1byte('encrypt_data/f4_ofb_c_e_50.bmp', data[:50] + ofb_e)
```

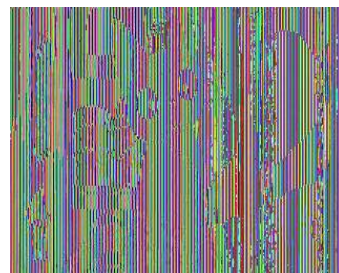
Результат:



Оригинал



Режим ECB



Режим OFB

3. Расшифровать файл `im9_caesar_cfb_c.bmp` не удалось.  
Поэтому используем другой файл с тем же ключом и вектором инициализации.  
`Key = 174`, `iv = 9`. Зашифровать в режиме ECB и CFB, оставив первые 50 байт без изменения.  
Шифр Цезаря.

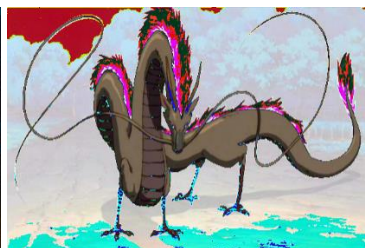
Реализация:

```
key = 174
iv = 9
data = read_write_file.read_data_1byte('data/f5.bmp')
ecb_e = em.ecb_e(data[50:], key, crypto_mode='caesar')
cfb_e = em.cfb_e(data[50:], key, iv, crypto_mode='caesar')
read_write_file.write_data_1byte('encrypt_data/f5_ecb_c_e_50.bmp', data[:50] + ecb_e)
read_write_file.write_data_1byte('encrypt_data/f5_cfb_c_e_50.bmp', data[:50] + cfb_e)
```

Результат:



Оригинал



Режим ECB



Режим CFB

4. Расшифровать файл im10\_caesar\_ctr\_c\_all.bmp не удалось.

Поэтому используем другой файл с тем же ключом и вектором инициализации.

Key = 223, iv = 78. Зашифровать в режиме ECB и CTR, оставив первые 50 байт без изменения.

Шифр Цезаря.

Реализация:

```
key = 223
iv = 78
data = read_write_file.read_data_1byte('data/f7.bmp')
ecb_e = em.ecb_e(data[50:], key, crypto_mode='caesar')
ctr_e = em.ctr_e(data[50:], key, crypto_mode='caesar')
read_write_file.write_data_1byte('encrypt_data/f7_ecb_c_e_50.bmp', data[:50] + ecb_e)
read_write_file.write_data_1byte('encrypt_data/f7_ctr_c_e_50.bmp', data[:50] + ctr_e)
```

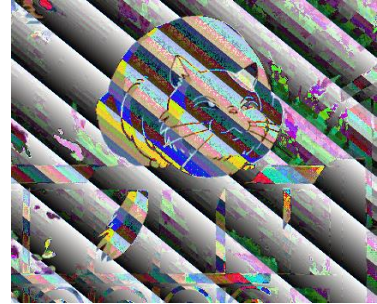
Результат:



Оригинал



Режим ECB



Режим CTR

5. Для одного из расшифрованных изображений выполнить следующее: на одном и том же ключе и векторе инициализации зашифровать во всех рассмотренных режимах, включая ECB, оставив первые 50 байт без изменения. Шифр Цезаря.

Реализация:

```
key = 331
iv = 53
data = read_write_file.read_data_1byte('data/f6.bmp')

ecb_e = em.ecb_e(data[50:], key, crypto_mode='caesar')
cbc_e = em.cbc_e(data[50:], key, iv, crypto_mode='caesar')
cfb_e = em.cfb_e(data[50:], key, iv, crypto_mode='caesar')
ofb_e = em.ofb_e(data[50:], key, iv, crypto_mode='caesar')
ctr_e = em.ctr_e(data[50:], key, crypto_mode='caesar')

read_write_file.write_data_1byte('encrypt_data/f6_ecb_c_e_50.bmp', data[:50] + ecb_e)
read_write_file.write_data_1byte('encrypt_data/f6_cbc_c_e_50.bmp', data[:50] + cbc_e)
read_write_file.write_data_1byte('encrypt_data/f6_cfb_c_e_50.bmp', data[:50] + cfb_e)
read_write_file.write_data_1byte('encrypt_data/f6_ofb_c_e_50.bmp', data[:50] + ofb_e)
read_write_file.write_data_1byte('encrypt_data/f6_ctr_c_e_50.bmp', data[:50] + ctr_e)
```



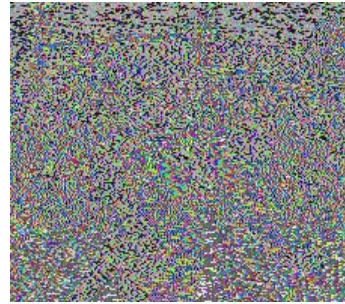
Результат:



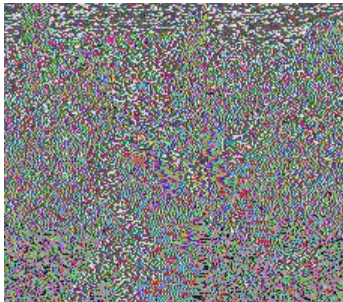
Оригинал



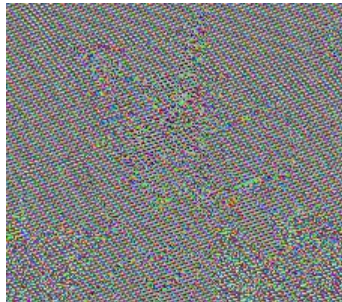
Режим ECB



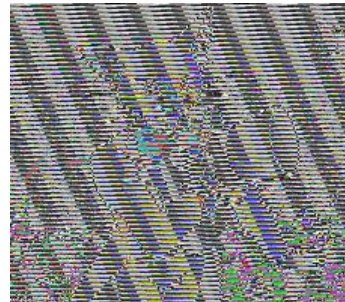
Режим CBC



Режим CFB



Режим OFB



Режим CTR

6. Расшифровать файл `im3_vigener_cbc_c_all.bmp`. Шифр Виженера. Режим CBC.  
Key = `MODELING`, iv = 67. Зашифровать, оставив первые 50 байт без изменения.

Реализация:

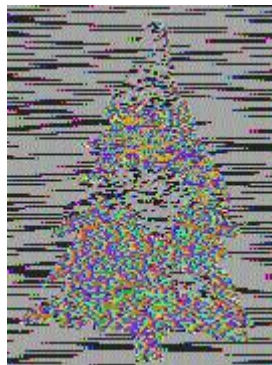
```
data = read_write_file.read_data_1byte('encrypt_data/im3_vigener_cbc_c_all.bmp')
key = 'MODELING'
iv = 67
cbc_d = em.cbc_d(data, key, iv, crypto_mode='vigenere')
read_write_file.write_data_1byte('decrypt_data/im3_vigener_cbc_c_all_decrypt.bmp', cbc_d)

data = read_write_file.read_data_1byte('decrypt_data/im3_vigener_cbc_c_all_decrypt.bmp')
cbc_e = em.cbc_e(data[50:], key, iv, crypto_mode='vigenere')
read_write_file.write_data_1byte('encrypt_data/im3_vigener_cbc_c_all_50.bmp', data[:50] + cbc_e)
```

Результат:



Оригинал



Режим CBC

7. Расшифровать файл im4\_vigener\_ofb\_c\_all.bmp. Шифр Виженера.Режим OFB.  
Key = MODULATOR, iv = 217. Зашифровать, оставив первые 50 байт без изменения.

Реализация:

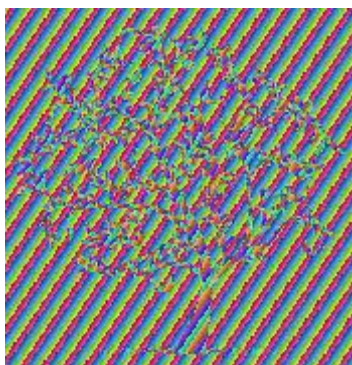
```
data = read_write_file.read_data_1byte('encrypt_data/im4_vigener_ofb_c_all.bmp')
key = 'MODULATOR'
iv = 217
ofb_d = em.ofb_d(data, key, iv, crypto_mode='vigener')
read_write_file.write_data_1byte('decrypt_data/im4_vigener_ofb_c_all_decrypt.bmp', ofb_d)

data = read_write_file.read_data_1byte('decrypt_data/im4_vigener_ofb_c_all_decrypt.bmp')
ofb_e = em.ofb_e(data[50:], key, iv, crypto_mode='vigener')
read_write_file.write_data_1byte('encrypt_data/im4_vigener_ofb_c_all_50.bmp', data[:50] + ofb_e)
```

Результат:



Оригинал



Режим OFB

8. Расшифровать файл im5\_vigener\_cfb\_c\_all.bmp. Шифр Виженера.Режим CFB.  
Key = MONARCH, iv = 172. Зашифровать, оставив первые 50 байт без изменения.

Реализация:

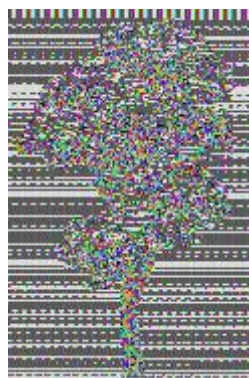
```
data = read_write_file.read_data_1byte('encrypt_data/im5_vigener_cfb_c_all.bmp')
key = 'MONARCH'
iv = 172
cfb_d = em.cfb_d(data, key, iv, crypto_mode='vigener')
read_write_file.write_data_1byte('decrypt_data/im5_vigener_cfb_c_all_decrypt.bmp', cfb_d)

data = read_write_file.read_data_1byte('decrypt_data/im5_vigener_cfb_c_all_decrypt.bmp')
cfb_e = em.cfb_e(data[50:], key, iv, crypto_mode='vigener')
read_write_file.write_data_1byte('encrypt_data/im5_vigener_cfb_c_all_50.bmp', data[:50] + cfb_e)
```

Результат:



Оригинал



Режим CFB



9. Расшифровать файл im6\_vigener\_ctr\_c\_all.bmp. Шифр Виженера. Режим CTR.  
Key = MONOLITH, iv = 167. Зашифровать, оставив первые 50 байт без изменения.

Реализация:

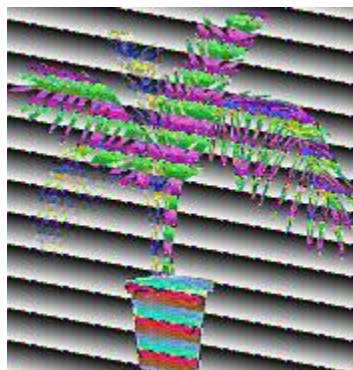
```
data = read_write_file.read_data_1byte('encrypt_data/im6_vigener_ctr_c_all.bmp')
key = 'MONOLITH'
iv = 167
ctr_d = em.ctr_d(data, key, iv, crypto_mode='vigener')
read_write_file.write_data_1byte('decrypt_data/im6_vigener_ctr_c_decrypt.bmp', ctr_d)

data = read_write_file.read_data_1byte('decrypt_data/im6_vigener_ctr_c_decrypt.bmp')
ctr_e = em.ctr_e(data[50:], key, iv, crypto_mode='vigener')
read_write_file.write_data_1byte('encrypt_data/im6_vigener_ctr_c_50.bmp', data[:50] + ctr_e)
```

Результат:



Оригинал



Режим CTR

10. Расшифровать файл im15\_affine\_cbc\_c\_all.bmp. Шифр аффинный. Режим CBC.  
a= 129, b= 107, iv = 243. Зашифровать, оставив первые 50 байт без изменения.

Реализация:

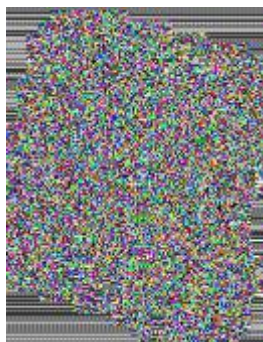
```
data = read_write_file.read_data_1byte('encrypt_data/im15_affine_cbc_c_all.bmp')
a = 129
b = 107
iv = 243
key = [a, b]
cbc_d = em.cbc_d(data, key, iv, crypto_mode='affine')
read_write_file.write_data_1byte('decrypt_data/im15_affine_cbc_c_all_decrypt.bmp', cbc_d)

data = read_write_file.read_data_1byte('decrypt_data/im15_affine_cbc_c_all_decrypt.bmp')
cbc_e = em.cbc_e(data[50:], key, iv, crypto_mode='affine')
read_write_file.write_data_1byte('encrypt_data/im15_affine_cbc_c_all_50.bmp', data[:50] + cbc_e)
```

Результат:



Оригинал



Режим CBC

11. Расшифровать файл im16\_affine\_ofb\_c\_all.bmp. Шифр аффинный. Режим OFB.

$a = 233$ ,  $b = 216$ ,  $iv = 141$ . Зашифровать, оставив первые 50 байт без изменения.

Реализация:

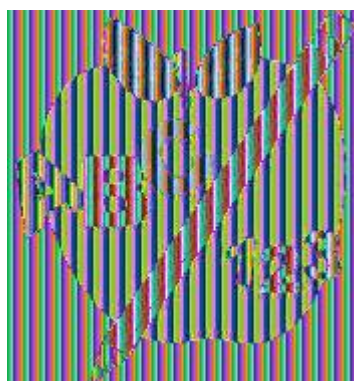
```
data = read_write_file.read_data_1byte('encrypt_data/im16_affine_ofb_c_all.bmp')
a = 233
b = 216
iv = 141
key = [a, b]
ofb_d = em.ofb_d(data, key, iv, crypto_mode='affine')
read_write_file.write_data_1byte('decrypt_data/im16_affine_ofb_c_all_decrypt.bmp', ofb_d)

data = read_write_file.read_data_1byte('decrypt_data/im16_affine_ofb_c_all_decrypt.bmp')
ofb_e = em.ofb_e(data[50:], key, iv, crypto_mode='affine')
read_write_file.write_data_1byte('encrypt_data/im16_affine_ofb_c_all_50.bmp', data[:50] + ofb_e)
```

Результат:



Оригинал



Режим OFB

12. Расшифровать файл im17\_affine\_cfb\_c\_all.bmp. Шифр аффинный. Режим CFB.

$a = 117$ ,  $b = 239$ ,  $iv = 19$ . Зашифровать, оставив первые 50 байт без изменения.

Реализация:

```
data = read_write_file.read_data_1byte('encrypt_data/im17_affine_cfb_c_all.bmp')
a = 117
b = 239
iv = 19
key = [a, b]
cfb_d = em.cfb_d(data, key, iv, crypto_mode='affine')
read_write_file.write_data_1byte('decrypt_data/im17_affine_cfb_c_all_decrypt.bmp', cfb_d)

data = read_write_file.read_data_1byte('decrypt_data/im17_affine_cfb_c_all_decrypt.bmp')
cfb_e = em.cfb_e(data[50:], key, iv, crypto_mode='affine')
read_write_file.write_data_1byte('encrypt_data/im17_affine_cfb_c_all_50.bmp', data[:50] + cfb_e)
```

Результат:



Оригинал



Режим CFB

13. Расшифровать файл im18\_affine\_ctr\_c\_all.bmp не удалось.

Поэтому используем другой файл с тем же ключом и вектором инициализации.

$a = 13$ ,  $b = 181$ ,  $iv = 78$ . Зашифровать в режиме CTR оставив первые 50 байт без изменения. Шифр аффинный.

Реализация:

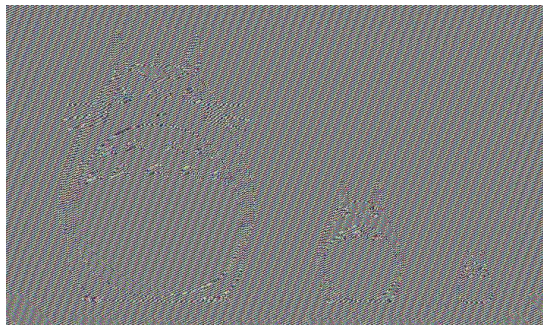
```
data = read_write_file.read_data_1byte('data/f8.bmp')
a = 13
b = 181
iv = 92
key = [a, b]
ctr_e = em.ctr_e(data[50:], key, iv, crypto_mode='affine')
read_write_file.write_data_1byte('encrypt_data/f8_ctr_a_e_50.bmp', data[:50] + ctr_e)

data = read_write_file.read_data_1byte('encrypt_data/f8_ctr_a_e_50.bmp')
ctr_d = em.ctr_e(data[50:], key, iv, crypto_mode='affine')
read_write_file.write_data_1byte('decrypt_data/f8_ctr_a_e_50_decrypt.bmp', data[:50] + ctr_d)
```

Результат:



Оригинал



Режим CTR



## Алгоритм шифрования на основе сети SPN.

Реализуем файл spn.py:

Зададим таблицы замен и перестановок:

```
s = [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7]
p = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15]
```

Реализуем функции обращения к таблицам s и p, а также обратные функции:

```
def sbbox(x):
    return s[x]

def pbbox(x):
    y = 0
    for i in range(len(p)):
        if x & (1 << i) != 0:
            y ^= (1 << p[i])
    return y

def asbox(x):
    return s.index(x)

def apbox(x):
    y = 0
    for i in range(len(p)):
        if x & (1 << i) != 0:
            y ^= (1 << p.index(i))
    return y
```

Реализуем функции формирования списка раундовых ключей для шифрования и расшифровывания:

```
def round_keys(k):
    rk = []
    rk.append((k >> 16) & (2**16 - 1))
    rk.append((k >> 12) & (2**16 - 1))
    rk.append((k >> 8) & (2**16 - 1))
    rk.append((k >> 4) & (2**16 - 1))
    rk.append(k & (2**16 - 1))
    return rk

def round_keys_to_decrypt(key):
    k = round_keys(key)
    l = []
    l.append(k[-1])
    for i in range(3, 0, -1):
        l.append(apbox(k[i]))
    l.append(k[0])
    return l
```

Реализуем функции mux(), demux(), mix():

```
def demux(x):
    y = []
    for i in range(4):
        y.append((x >> (i * 4)) & 0xf)
    return y

def mux(x):
    y = 0
    for i in range(4):
        y ^= (x[i] << (i * 4))
    return y

def mix(p, k):
    return p ^ k
```

Реализуем функции round(), decrypt\_round():

```
def round(p, k):
    u = mix(p, k)
    v = []
    for x in demux(u):
        v.append(sbox(x))
    w = pbox(mux(v))
    return w

def decrypt_round(p, k):
    u = mix(p, k)
    v = []
    for x in demux(u):
        v.append(asbox(x))
    w = apbox(mux(v))
    return w
```

Реализуем функции last\_round(), decrypt\_last\_round():

```
def last_round(p, k1, k2):
    u = mix(p, k1)
    v = []
    for x in demux(u):
        v.append(sbox(x))
    u = mix(mux(v), k2)
    return u

def decrypt_last_round(p, k1, k2):
    u = mix(p, k1)
    v = []
    for x in demux(u):
        v.append(asbox(x))
    u = mix(mux(v), k2)
    return u
```

Реализуем функции encrypt(), decrypt():

```
def encrypt(p, rk, rounds):
    x = p
    for i in range(rounds-1):
        x = round(x, rk[i])
    x = last_round(x, rk[rounds-1], rk[rounds])
    return x

def decrypt(p, lk, rounds):
    x = p
    for i in range(rounds-1):
        x = decrypt_round(x, lk[i])
    x = decrypt_last_round(x, lk[rounds-1], lk[rounds])
    return x
```

Реализуем функции encrypt\_data(), decrypt\_data():

```
def encrypt_data(data, key, rounds):
    e = []
    rk = round_keys(key)
    for x in data:
        e.append(encrypt(x, rk, rounds))
    return e

def decrypt_data(data, key, rounds):
    d = []
    lk = round_keys_to_decrypt(key)
    for x in data:
        d.append(decrypt(x, lk, rounds))
    return d
```

1. Расшифровать файл d5\_spn\_c\_all.bmp, зашифрованный шифром на основе сети SPN.  
Key = 34523456231.

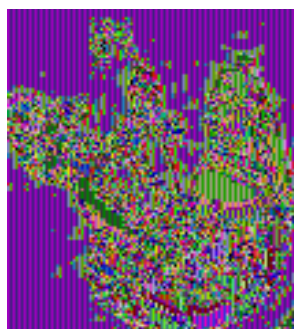
Реализация:

```
data = read_write_file.read_data_2byte('encrypt_data/d5_spn_c_all.bmp')
key = 34523456231
rounds = 4
spn_d = spn.decrypt_data(data, key, rounds)
spn_e = spn.encrypt_data(spn_d[500:], key, rounds)
read_write_file.write_data_2byte('decrypt_data/d5_spn_c_all_500.bmp', spn_d[:500] + spn_e)
```

Результат:



Оригинал



SPN