1. Криптосистема Хилла

Шифр замены. Только замена выполняется не символа на символ, а блока символов на блок символов. Такой шифр называют блочным. Рассмотрим случай, когда блок состоит из двух символов. Идея замены была предложена Хиллом в статьях: L. S. Hill, "Concerning certain linear transformation apparatus of cryptography", American Mathematical Monthly, Volume 38 (1931), 135-154. Lester S. Hill, Cryptography in an Algebraic Alphabet, The American Mathematical Monthly Vol.36, June–July 1929, pp. 306–312.

А. Разбиваем сообщение на блоки по 2 символа. Возьмем сообщение

THE GOLD IS BURIED IN ORONO.

После разбиения получаем

TH EG OL DI SB UR IE DI NO RO NO.

Т.к. у каждого символа есть свой числовой эквивалент (табл.1), то полученное разбиение будет выглядеть так:

19 7 4 6 14 11 3 8 18 1 20 17 8 4 3 8 13 14 17 14 13 14.

Таблица 1

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Б. Каждый блок из двух чисел P1 P2 открытого сообщения преобразуется в блок из двух чисел C1 C2 закрытого сообщения по следующей формуле:

$$C \equiv AP \pmod{26}$$

Где $C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}$, $P = \begin{bmatrix} P_1 \\ P_2 \end{bmatrix}$, A – матрица 2x2.

Пусть $A = \begin{pmatrix} 5 & 17 \\ 4 & 15 \end{pmatrix}$, тогда шифрование первого блока будет выглядеть так:

$$C_1 \equiv 5 \cdot 19 + 17 \cdot 7 \equiv 6 \pmod{26}$$
$$C_2 \equiv 4 \cdot 19 + 15 \cdot 7 \equiv 25 \pmod{26}$$

Если применить эту формулу ко всем блокам, то получим следующий результат:

6 25 18 2 23 13 21 2 3 9 25 23 4 14 21 2 17 2 11 l8 l7 2.

Или в символьном виде:

GZ SC XN VC DJ ZX EO VC RC LS RC.

Расшифрование выполняется по формуле:

$$P \equiv \bar{A}C \pmod{26}$$

Где $\bar{A}$ - обратная к А матрица по mod 26.

Для матрицы

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Если определитель

$$\Delta = \det A = ad - bc$$

является взаимно простым со значением модуля (в данном случае 26), то обратную матрицу $\bar{A}$ можно найти по следующей формуле:

$$\bar{A} = \bar{\Delta} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Здесь $\bar{\Delta}$ - обратное значение по умножению к Δ по модулю 26.

Для $A = \begin{pmatrix} 5 & 17 \\ 4 & 15 \end{pmatrix}$ обратная по модулю 26 будет матрица $\bar{A} = \begin{pmatrix} 17 & 5 \\ 18 & 23 \end{pmatrix}$.

Тогда, расшифровка, например, первого зашифрованного блока будет такой:

$$\begin{pmatrix} P_1 \\ P_2 \end{pmatrix} \equiv \begin{pmatrix} 17 & 5 \\ 18 & 23 \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} \pmod{26}$$

Расшифровать файл im3_hill_c_all.bmp. Ключ – матрица K=[[189 58] [21 151]].

1. Дешифровать png-файл b4_hill_c_all.png. Первые четыре байта в любом png-файле: 137, 80, 78, 71.
2. Дешифровать файл text2_hill_c_all.txt. Известно, что текст в файле начинается со слова Whose.
3. Алгоритм Евклида вычисления НОД:

```
EUCLID(a, b)
1.   A ← a; B ← b
2.   if B = 0  return  A = gcd(a, b)
3.   R = A mod B
4.   A ← B
5.   B ← R
6.   goto 2
```

Реализация в Python:

```
def gcd(a, b):
    # Return the GCD of a and b using Euclid's Algorithm
    while a != 0:
        a, b = b % a, a
return b
```

Пример расчета. Найти gcd(1970, 1066):

```
1970  = 1 x 1066 + 904      gcd(1066, 904)
1066  = 1 x 904 + 162       gcd(904, 162)
 904  = 5 x 162 + 94        gcd(162, 94)
 162  = 1 x 94 + 68         gcd(94, 68)
  94  = 1 x 68 + 26         gcd(68, 26)
  68  = 2 x 26 + 16         gcd(26, 16)
  26  = 1 x 16 + 10         gcd(16, 10)
  16  = 1 x 10 + 6          gcd(10, 6)
  10  = 1 x 6 + 4           gcd(6, 4)
   6  = 1 x 4 + 2           gcd(4, 2)
   4  = 2 x 2 + 0           gcd(2, 0)
```

Следовательно, gcd(1970, 1066) = 2.

Расширенный алгоритм Евклида:

```
EXTENDED EUCLID(m, b)
1. (A1, A2, A3) ← (1, 0, m); (B1, B2, B3) ← (0, 1, b)
2. if B3 = 0 return A3 = gcd(m, b); no inverse
3. if B3 = 1 return B3 = gcd(m, b); B2 = B^1 mod m
```

4. $Q = \left\lfloor \frac{A3}{B3} \right\rfloor$

```
5. (T1, T2, T3) ← (A1 - QB1, A2 - QB2, A3 - QB3)
6. (A1, A2, A3) ← (B1, B2, B3)
7. (B1, B2, B3) ← (T1, T2, T3)
8. goto 2
```

Реализация в Python:

```
def findModInverse(a, m):
    # Returns the modular inverse of a % m, which is
    # the number x such that a*x % m = 1

    if gcd(a, m) != 1:
        return None  # no mod inverse if a & m aren't relatively prime

    # Calculate using the Extended Euclidean Algorithm:
    u1, u2, u3 = 1, 0, a
    v1, v2, v3 = 0, 1, m
    while v3 != 0:
        q = u3 // v3  # // is the integer division operator
        v1, v2, v3, u1, u2, u3 = (u1 - q * v1), (u2 - q * v2), (u3 - q * v3), v1, v2, v3
return u1 % m
```

Найдем обратное по умножению для 550 по mod 1759:

Q	A1	A2	A3	B1	B2	B3
	1	0	1759	0	1	550
3	0	1	550	1	3	109
5	1	3	109	5	16	5
21	5	16	5	106	339	4
1	106	339	4	111	355	1

Используя расширенный алгоритм Евклида, найти обратное значение по умножению дл 1234 по mod 4321. Проверить с помощью функции findModInverse.

4. Режимы шифрования

Режим шифрования CBC (рис.1, 2).

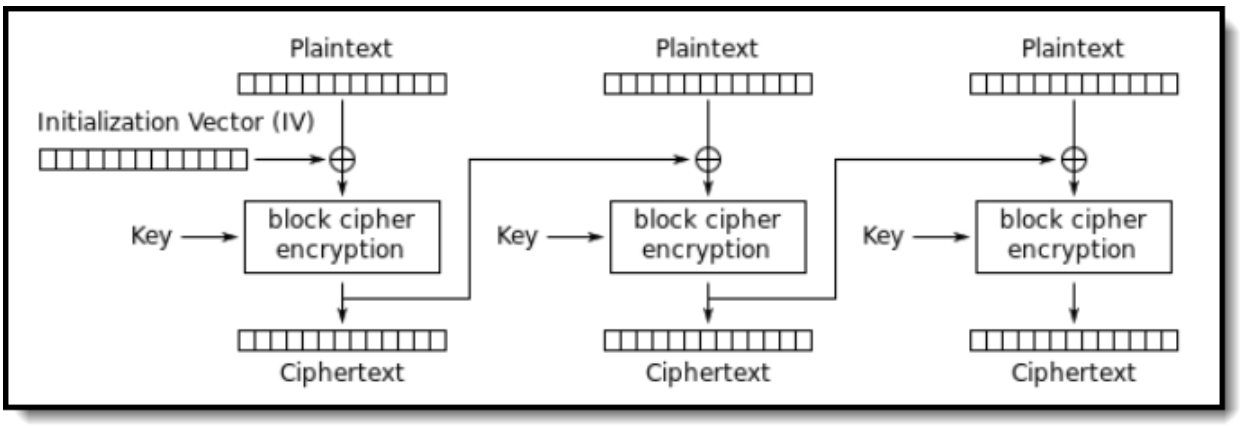

Рисунок 1 – Шифрование в режиме CBC

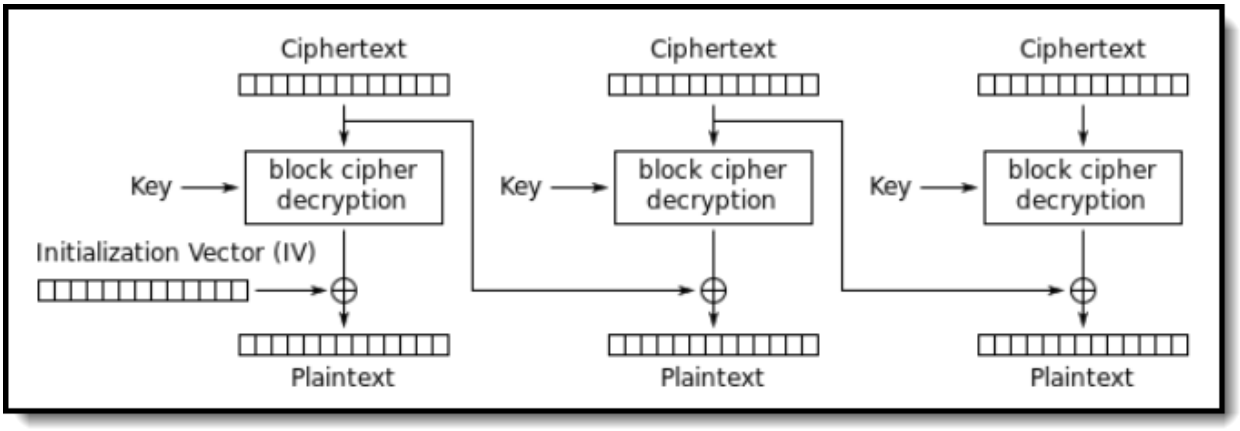

Рисунок 2 – Расшифрование в режиме CBC

Режим шифрования OFB (рис.3, 4).

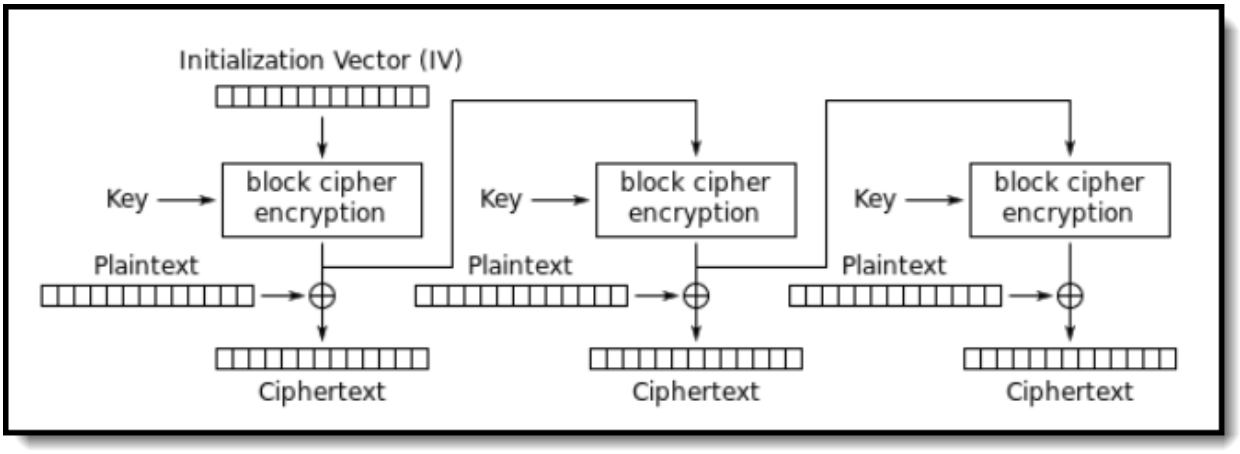

Рисунок 3 – Шифрование в режиме OFB

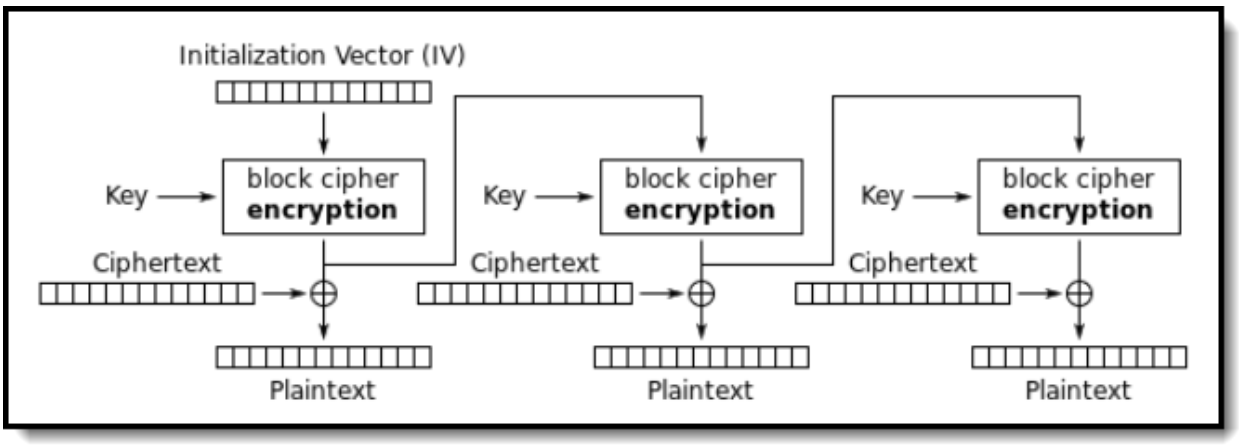

Рисунок 4 – Расшифрование в режиме OFB

Режим шифрования CFB (рис.5, 6).

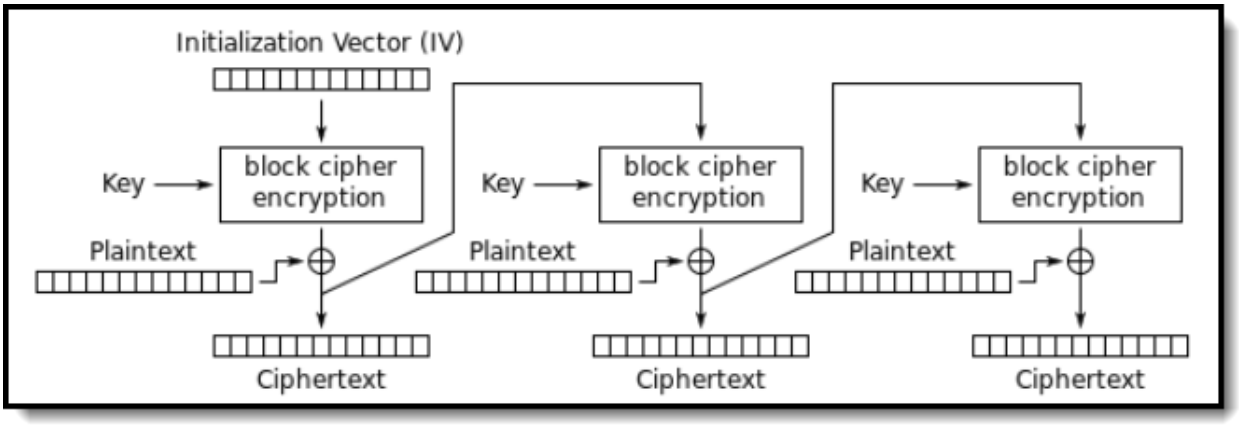

Рисунок 5 – Шифрование в режиме CFB

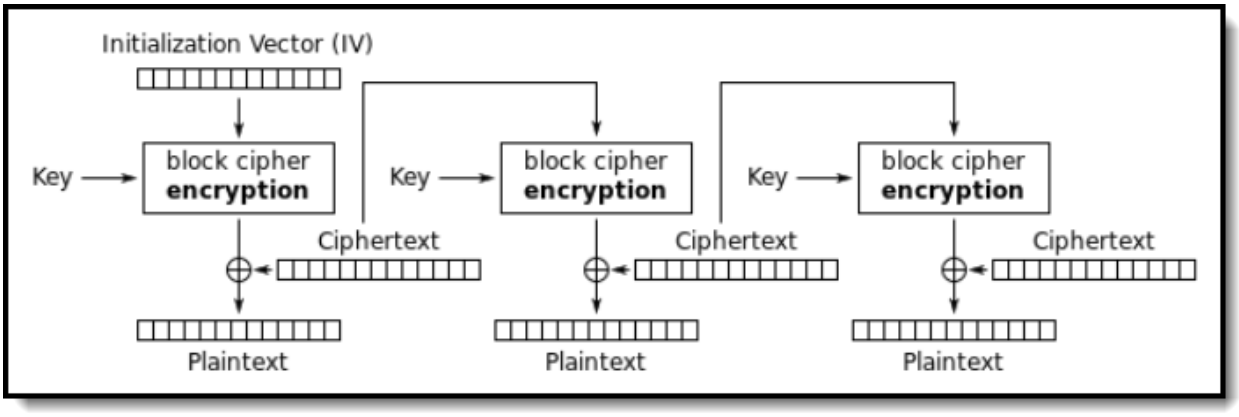

Рисунок 6 – Расшифрование в режиме CFB

Режим шифрования CTR (рис.7, 8).

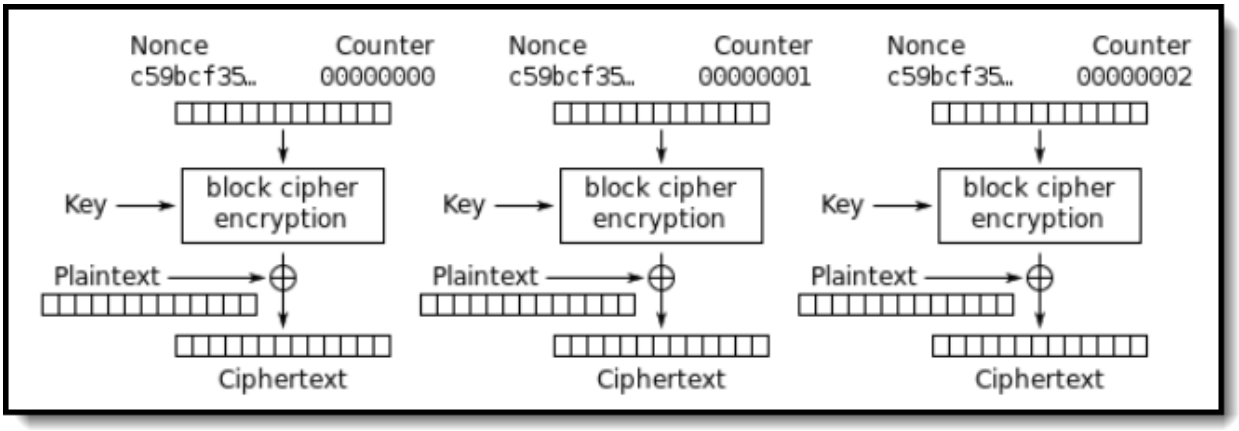

Рисунок 7 – Шифрование в режиме CTR

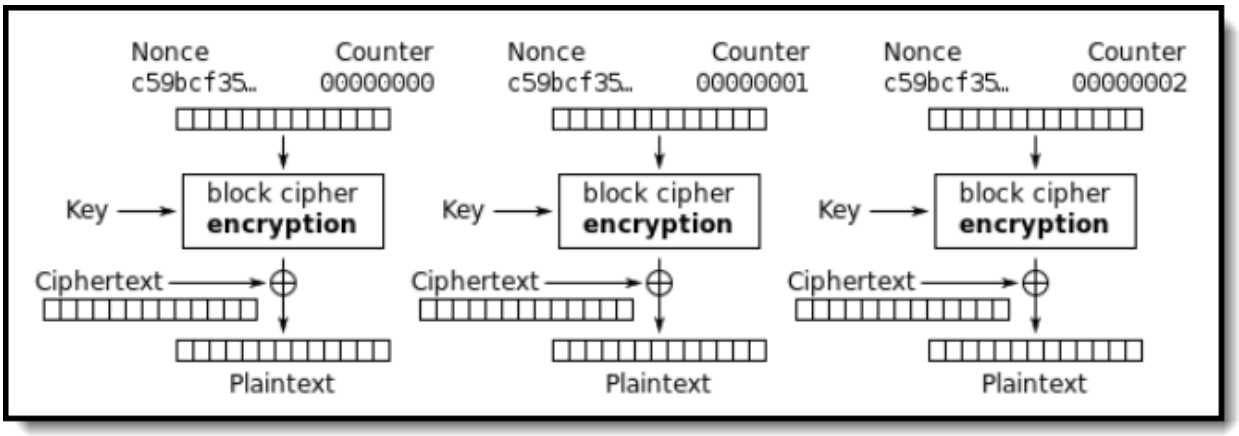

Рисунок 8 – Расшифрование в режиме CTR

6.Расшифровать файл im7_caesar_cbc_c_all.bmp – зашифрованное шифром Цезаря изображение в формате bmp. Режим шифрования CBC (рис. 1, 2). Ключ равен 123. Вектор инициализации равен 5. Зашифровать в режиме ECB и в режиме CBC, оставив первые 50 байт без изменения. Сравнить полученные изображения.

7. Расшифровать файл im8_caesar_ofb_c_all.bmp – зашифрованное шифром Цезаря изображение в формате bmp. Режим шифрования OFB (рис.

3, 4). Ключ равен 56. Вектор инициализации равен 9. Зашифровать в режиме ECB и в режиме OFB, оставив первые 50 байт без изменения. Сравнить полученные изображения.

8.Расшифровать файл im9_caesar_cfb_c.bmp – зашифрованное шифром Цезаря изображение в формате bmp. Режим шифрования CFB (рис. 5, 6). Ключ равен 174. Вектор инициализации равен 9. Зашифровать в режиме ECB и в режиме OFB, оставив первые 50 байт без изменения. Сравнить полученные изображения.

9.Расшифровать файл im10_caesar_ctr_c_all.bmp – зашифрованное шифром Цезаря изображение в формате bmp. Режим шифрования CTR (рис. 7, 8). Ключ равен 223. Вектор инициализации равен 78. Зашифровать в режиме ECB и в режиме CTR, оставив первые 50 байт без изменения. Сравнить полученные изображения.

10. 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. Сравнить полученные изображения.

11. Расшифровать файл im3_vigener_cbc_c_all.bmp. Шифр Виженера. Режим CBC. Ключ: MODELING, вектор инициализации: 67. Зашифровать, оставив первые 50 байт без изменения.

12. Расшифровать файл im4_vigener_ofb_c_all.bmp. Шифр Виженера. Режим OFB. Ключ: MODULATOR, вектор инициализации: 217. Зашифровать, оставив первые 50 байт без изменения.

13. Расшифровать файл im5_vigener_cfb_c_all.bmp. Шифр Виженера. Режим CFB. Ключ: MONARCH, вектор инициализации: 172. Зашифровать, оставив первые 50 байт без изменения.

14. Расшифровать файл im6_vigener_ctr_c_all.bmp. Шифр Виженера. Режим CTR. Ключ: MONOLITH, вектор инициализации: 167. Зашифровать, оставив первые 50 байт без изменения.

15. 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.

16. 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.

17. 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.

18. Расшифровать файл im18_affine_ctr_c_all.bmp. Шифр аффинный. Режим CTR. a= 13 b= 181 iv = 92. Зашифровать, оставив первые 50 байт без изменения.

19. Рассмотрим алгоритм шифрования, построенный на основе сети SPN, структура которого показано на рис. 9. S-блок замены для данного алгоритма представлен в табл.2.

Таблица 2

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

Перестановка бит внутри блока задается таблицей 3.

Таблица 3

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

Псевдокод алгоритма шифрования показан на рис.10.

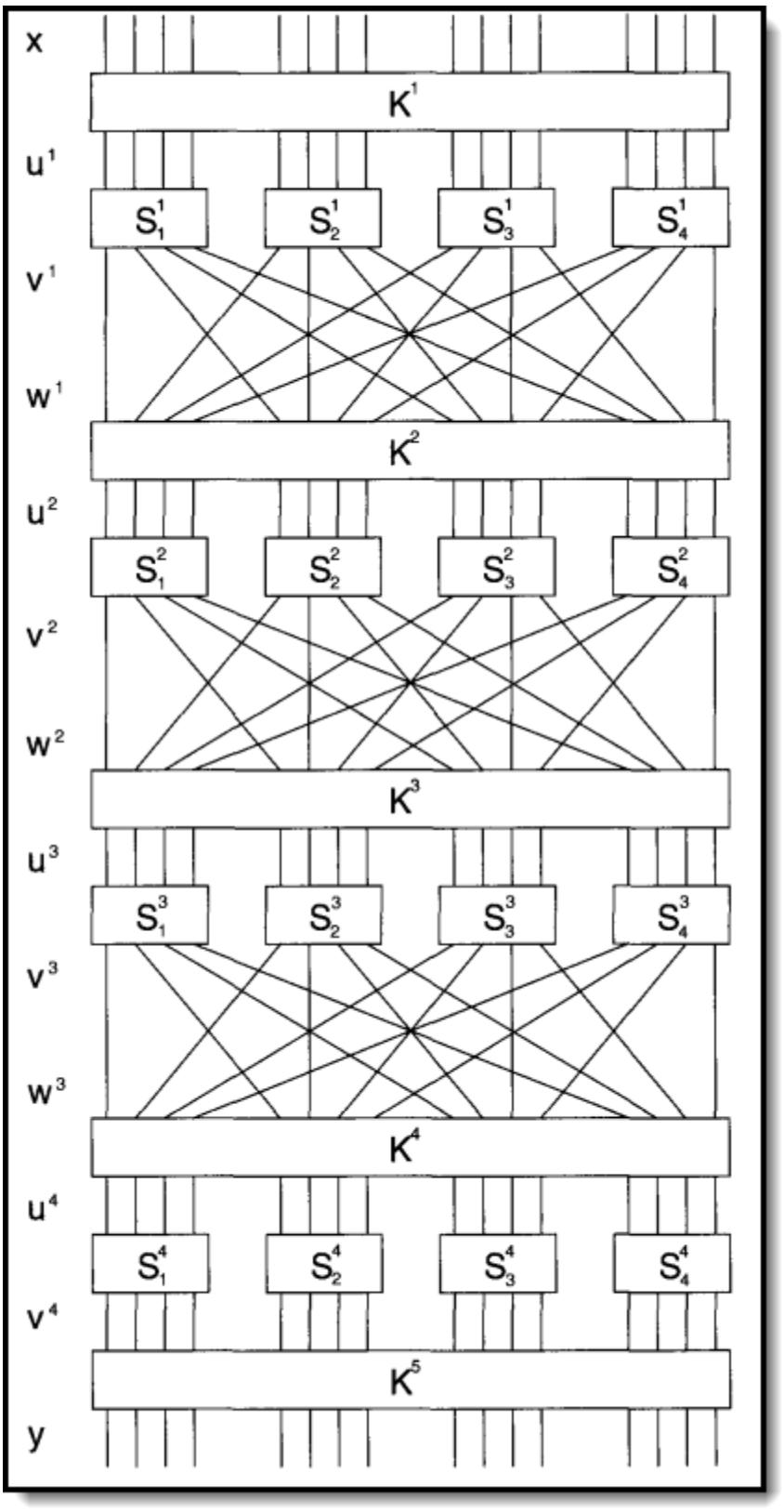

Рисунок 9 – Структура алгоритма шифрования, построенного на основе сети SPN

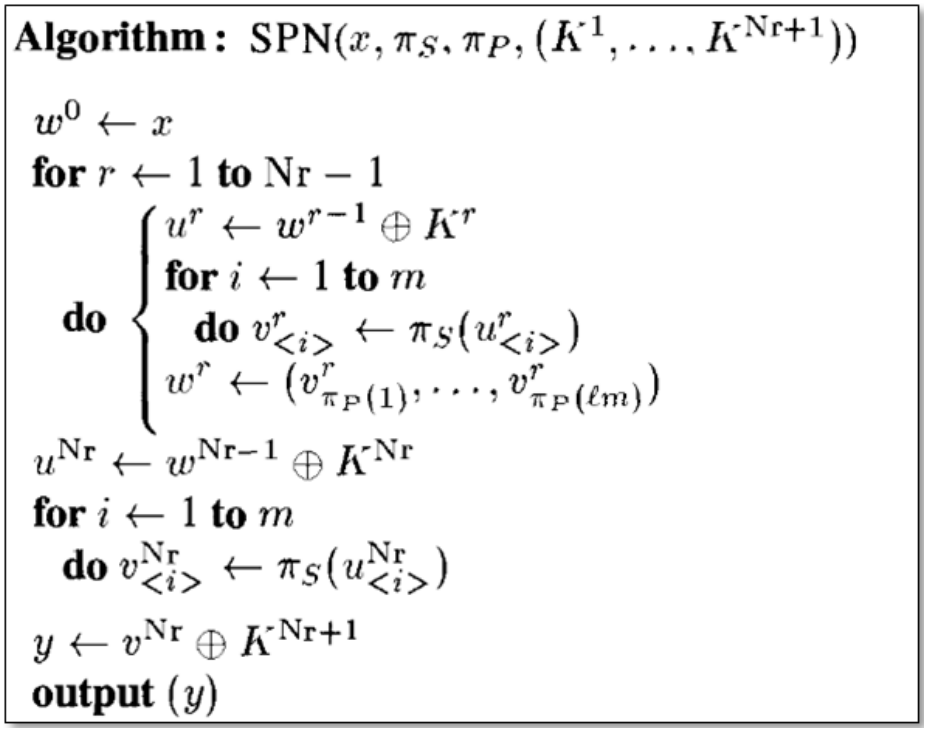

Рисунок 10 – Псевдокод алгоритма шифрования

а) Написать функцию (табл. 4) round_keys_to_decrypt(key), где key – ключ шифрования. Функция формирует список раундовых ключей для расшифрования $[L_0, L_1, L_2, L_3, L_4]$. Раундовые ключи для расшифрования рассчитываются из раундовых ключей шифрования $[K_0, K_1, K_2, K_3, K_4]$ по следующим формулам:

$$L_0 = K_4$$
$$L_1 = \pi_S^{-1}(K_3)$$
$$L_2 = \pi_S^{-1}(K_2)$$
$$L_3 = \pi_S^{-1}(K_1)$$
$$L_4 = K_0$$

Таблица 4

```
def round_keys_to_decrypt(self, key):
    K = self.round_keys(key)
    L = [ ]
    pass
    return L
```

Результат работы функции приведен в табл. 5.

Таблица 5

key =00111010100101001101011000111111 K0=0011101010010100, L0=1101011000111111 K1=1010100101001101, L1=0100111000110101 K2=1001010011010110, L2=1010011100011010 K3=0100110101100011, L3=1101001110000101 K4=1101011000111111, L4=0011101010010100

б) Написать функцию decrypt_round(self, p, k), где p – данные (16 бит), поступающие на вход раунда расшифрования, k-раундовый подключ. Функция один в один совпадает с функцией round(self, p, k), за исключением вызова функций замены и перестановки. В функции decrypt_round надо использовать функции asbox и apbox.

в) Написать функцию decrypt_last_round(self, p, k1, k2), где p – данные (16 бит), поступающие на вход последнего раунда расшифрования, k1, k2-последние раундовые подключи расшифрования. Функция один в один совпадает с функцией last_round(self, p, k1, k2), за исключением вызова функции замены. В функции decrypt_last_round надо использовать функцию asbox.

г) Написать функцию decrypt (self, p, L, rounds), где p – данные (16 бит), поступающие на вход функции расшифрования, L-список раундовых ключей расшифрования, rounds – количество раундов. Функция один в один совпадает с функцией encrypt(self, p, K, rounds), за исключением вызова функций round и last_round. В функции decrypt надо использовать функции decrypt_round и decrypt_last_round.

д) Написать функцию encrypt_data(self, data, key, rounds), где data – данные, прочитанные из файла, key – ключ шифра, rounds – количество

раундов. В этой функции надо сформировать список раундовых ключей шифрования и для каждого числа (16 бит) в списке data вызывать функцию encrypt. Функция возвращает список зашифрованных данных.

е) Написать функцию decrypt_data(self, data, key, rounds), где data – данные, прочитанные из зашифрованного файла, key – ключ шифра, rounds – количество раундов. В этой функции надо сформировать список раундовых ключей расшифрования и для каждого числа (16 бит) в списке data вызвать функцию decrypt. Функция возвращает список расшифрованных данных.

ж) Зашифровать и расшифровать содержимое файла (любого) с помощью функций encrypt_data и decrypt_data. Для получения содержимого файла в виде списка значений использовать функцию read_data_2byte:

```
data = read_write_file.read_data_2byte(fname)
```

Для записи функции в файл использовать функцию write_data_2byte:

```
read_write_file.write_data_2byte(fname, data)
```

Убедиться, что расшифрованный после зашифрования файл совпадает с исходным.

з) Расшифровать файл d5_spn_c_all.bmp – зашифрованное шифром на основе сети SPN (рис. 9, 10) изображение в формате bmp. Ключ равен 34523456231.

Полученное изображение в формате bmp зашифровать. Сохранить в файле следующие данные: первые 500 байт – исходные (незашифрованные) данные, все последующие байты – зашифрованные. Полученный файл открыть в редакторе. Вставить в отчет исходное и зашифрованное таким образом изображение.

20. Программа шифрования

Программа, которая выполняет шифрование 16-ти битового числа по алгоритму на рис.10 представлена в табл. 6.

Таблица 6

```
def binary(x, k):
    y = binary1(x)
    if len(y) < k:
        return "0"*(k-len(y))+y
    return y

def binary1(x):
    if x != 0:
        y = binary1(x >> 1) + str(x&1)
        if y == "":
            return "0"
        else:
            return y
    else:
        return ""


class SPN1():

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

    #S-box
    s = [14, 4, 13, 1, 2, 15, 11, 8,
         3, 10, 6, 12, 5, 9, 0, 7]

    # s-box
    def sbox(self, x):
        return self.s[x]

    # p-box
    def pbox(self, x):
        y = 0
        for i in range(len(self.p)):
            if (x & (1 << i)) != 0:
                y ^= (1 << self.p[i])
        return y


    # break into 4-bit chunks
    def demux(self, x):
        y = []
        for i in range(0, 4):
            y.append((x >> (i*4)) & 0xf)
        return y


    #convert back into 16-bit state
```

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

    def round_keys(self, 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

    # Key mixing
    def mix(self, p, k):
        v = p ^ k
        return v

    #round function
    def round(self, p, k):
        #XOR key
        u = self.mix(p, k)
        v = []
        # run through substitution layer
        for x in self.demux(u):
            v.append(self.sbox(x))
        # run through permutation layer
        w = self.pbox(self.mux(v))
        return w

    def last_round(self, p, k1, k2):
        #XOR key
        u = self.mix(p, k1)
        v = []
        # run through substitution layer
        for x in self.demux(u):
            v.append(self.sbox(x))
        #XOR key
        u = self.mix(self.mux(v), k2)
        return u

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


def main():
    e = SPN1()
    x = int('0010011010110111', 2)
    rounds = 4
    k = int('00111010100101001101011000111111', 2)
    rk = e.round_keys(k)
    print('k=', binary(k, 16))
    print('k1=', binary(rk[0], 4))
    print('k2=', binary(rk[1], 4))
```

```
    print('k3=', binary(rk[2], 4))
    print('k4=', binary(rk[3], 4))
    print('k5=', binary(rk[4], 4))
    y = e.encrypt(x, rk, rounds)
    print('y=', binary(y, 4))


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