Оглавление

Алгоритм S-AES

Алгоритм S-AES (упрощенный AES [1, 2]) имеет такую же структуру, как и алгоритм AES. Отличия только в значениях параметров алгоритма. В S-AES они имеют существенно меньшую размерность. Так, если в AES матрица состояния шифра формируется из байт, то в упрощенной версии S-AES, из полубайт (слово из 4 бит - нибл). Полубайт b, состоящий из бит $b_3b_2b_1b_0$, рассматривается как полином с коэффициентами, принадлежащими множеству $\{0,1\}$:

$$b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

Например, полубайт с 16-м значением ‘A’ (1010_2) соответствует полиному $x^3 + x$.

Конечное множество таких полиномов, образованных из полубайтов, формируют поле $GF(2^4)$. Это означает, в частности, что над элементами этого множества определены операции: сложения, вычитания, умножения и деления.

Сложение полиномов в поле $GF(2^4)$

Сложение двух полиномов в поле $GF(2^4)$ заключается в сложении их коэффициентов при соответствующих степенях полиномов. Так как, коэффициенты принадлежат множеству $\{0,1\}$, следовательно, сложение выполняется с приведением результата по mod 2, что является операцией «исключающее или».

Пример

$12 + 7 = 1100 + 0111 = 1011 = 11.$

Или в виде полиномов: $(x^3 + x^2) + (x^2 + x + 1) = x^3 + x + 1$

Умножение полиномов в поле $GF(2^4)$

Умножить два элемента из поля $GF(2^4)$ означает умножить два соответствующих этим элементам полинома.

Пример

$$
\begin{aligned}
11\cdot 7 &= (x^3+x+1)\cdot(x^2+x+1) = \\
&= x^5+x^4+x^3+x^3+x^2+x+x^2+x+1= \\
&= x^5+x^4+1
\end{aligned}
$$

Для того чтобы результат умножения являлся элементом поля надо, гарантировать, что полученный в результате умножения полином не будет иметь степень большую, чем три. Для этого надо поделить полученный результат на полином 4-ой степени, т.е. выполнить приведение по модулю неприводимого полинома $m(x)$.

Неприводимый полином можно рассматривать как полином, который нельзя разложить на произведение полиномов меньших степеней (рис.1). Полученный остаток от деления и будет искомым значением – результатом перемножения двух полиномов.

Пример

Умножить 11 на 7 в $GF(2^4)$, $m(x)=x^4+x+1$

$$(x^3+x+1)(x^2+x+1)=(x^5+x^4+1) \bmod (x^4+x+1)=x^2$$

$$
\begin{array}{r|l}
 & x+1 \quad \text{(частное)} \\
\hline
x^4+x+1 & x^5+x^4+1 \\
 & +\ x^5+x^2+x \\
\cline{2-2}
 & x^4+x^2+x+1 \\
 & +\ x^4+x+1 \\
\cline{2-2}
 & x^2 \quad \text{(остаток)}
\end{array}
$$

$$11\cdot 7 = 1011\cdot 0111 = 0100 = 4$$

Полученный результат можно увидеть в таблице на рис.2, где приведены результаты перемножения элементов в поле $GF(2^4)$ по модулю неприводимого полинома $m(x) = x^4 + x + 1$.

k	Неприводимые полиномы		
1	x	$x+1$	
2	x^2+x+1		
3	x^3+x+1	x^3+x^2+1	
4	x^4+x+1	x^4+x^3+1	$x^4+x^3+x^2+x+1$
5	x^5+x^2+1	x^5+x^3+1	$x^5+x^3+x^2+x+1$
	$x^5+x^4+x^3+x+1$	$x^5+x^4+x^3+x^2+1$	$x^5+x^4+x^2+x+1$
6	x^6+x+1	x^6+x^3+1	x^6+x^5+1
	$x^6+x^4+x^2+x+1$	$x^6+x^4+x^3+x+1$	$x^6+x^5+x^2+x+1$
	$x^6+x^5+x^3+x^2+1$	$x^6+x^5+x^4+x^2+1$	$x^6+x^5+x^4+x+1$

Рисунок 1 – Неприводимые полиномы степени k

⊗	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
F	0	F	D	2	9	6	4	8	1	E	C	3	8	7	5	A

Рисунок 2 – Умножение в поле $GF(2^4)$, $m(x) = x^4 + x + 1$

Рассмотрим простой алгоритм умножения двух полиномов, который можно реализовать программно на компьютере. Надо перемножить два полинома $f(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ и $g(x) = b_3x^3 + b_2x^2 + b_1x + b_0$:

$$f(x)g(x) = b_3 f(x)x^3 + b_2 f(x)x^2 + b_1 f(x)x + b_0 f(x)$$

Видно, что основная операция – умножение полинома $f(x)$ на x:

$$f(x)\cdot x=(a_3x^3+a_2x^2+a_1x+a_0)\cdot x=$$
$$=a_3x^4+a_2x^3+a_1x^2+a_0x$$

Полученный результат надо привести по модулю неприводимого полинома $m(x)=x^4+x+1$.

Если $a_3=0$, то приводить не надо и результат умножение полинома $f(x)$ на x: $a_2a_1a_0 0$.

Если $a_3=1$, то

$$(f(x)\cdot x)\bmod m(x)=$$
$$=(a_3x^4)\bmod m(x)+(a_2x^3+a_1x^2+a_0x)\bmod m(x)=$$
$$=(x+1)+(a_2x^3+a_1x^2+a_0x)$$

Это означает, что в этом случае результат умножение полинома $f(x)$ на x: $a_2a_1a_0 0+11$.

Пример

Умножить 11 на 7 в $GF(2^4)$, $m(x)=x^4+x+1$

$f(x)=x^3+x+1\,(1011)$

$g(x)=x^2+x+1\ (0111)$

$f(x)\cdot g(x)=f(x)\cdot x^2+f(x)\cdot x+f(x)$

$f(x)\cdot x: 0110+11=0101$

$f(x)\cdot x^2: 1010$

Тогда,

$$\begin{array}{lll} f(x)\cdot g(x): & 1011\oplus & (\,f(x)\,) \\ & 0101\oplus & (\,f(x)\cdot x\,) \\ & 1010 & (\,f(x)\cdot x^2\,) \\ \hline & 0100 & (\,x^2\,) \end{array}$$

Данный подход реализован в виде функции **gf_multiply_modular**:

```
def gf_multiply_modular(a, b, mod, n):
    """
    INPUTS
    a - полином (множимое)
    b - полином (множитель)
    mod - неприводимый полином
    n - порядок неприводимого полинома
    OUTPUTS:
    product - результат перемножения двух полиномов a и b
    """

    #  маска для наиболее значимого бита в слове
    msb = 2**(n - 1)
    # маска на все биты
    mask = 2**n - 1

    # r(x) = x^n mod m(x)
    r = mod ^ (2**n)
    product = 0  # результат умножения
    mm = 1

    for i in range(n):
        if b & mm > 0:
            # если у множителя текущий бит 1
            product ^= a
        #  выполняем последовательное умножение на x
        if a & msb == 0:
            # если старший бит 0, то просто сдвигаем на 1 бит
            a <<= 1
        else:
            # если старший бит 1, то сдвиг на 1 бит
            a <<= 1
            # и сложение по модулю 2 с r(x)
            a ^= r
            # берем только n бит
            a &= mask
        # формируем маску для получения очередного бита в множителе
        mm += mm
    return product
```

Вызов и результат работы этой функции представлен на рис.3.

```
m = int('10011', 2)
f = int('1011', 2)
g = int('0111', 2)
product = gf_multiply_modular(f, g, m, n=4)
print('f*g={}'.format(binary(product, 4)))
```

```
f*g=0100
```

Рисунок 3 – Вызов и результат работы функции **gf_multiply_modular**

Деление полиномов в поле $GF(2^4)$

Деление полиномов для нахождения частного и остатка от деления требуется в реализации расширенного алгоритма Евклида. Ниже приведена функция gf_divide, которая выполняет деление полинома a(x) на полином b(x). Алгоритм деления повторяет деление «в столбик».

```
def gf_divide(a, b):
    # деление полинома на полином
    # результат: частное, остаток (полиномы)
    dividend = a  # делимое
    divisor = b   # делитель

    a = 0
    # бит в делимом
    m = len(bin(dividend))-2
    # бит в делителе
    n = len(bin(divisor))-2
    s = divisor << m
    msb = 2 ** (m + n - 1)
    for i in range(m):
        dividend <<= 1
        if dividend & msb > 0:
            dividend ^= s
            dividend ^= 1
    maskq = 2**m - 1
    maskr = 2**n - 1
    r = (dividend >> m) & maskr
    q = dividend & maskq
    return q, r
```

Вызов и результат работы этой функции представлен на рис.4.

```
a = int('1011', 2)
b = int('0011', 2)
q, r = gf_divide(a,b)
print('q={},  r={}'.format(binary(q, 4), binary(r, 4)))
```

```
q=0110,  r=0001
```

Рисунок 4

Поиск обратного полинома по умножению в поле $GF(2^4)$

Поиск обратного значения по умножению в поле можно осуществить посредством применения расширенного алгоритма Евклида (рис.5).

EXTENDED EUCLID (m(x), b(x))

1. [A_1(x), A_2(x), A_3(x)] = [1, 0, m(x)]
   [B_1(x), B_2(x), B_3(x)] = [0, 1, b(x)]
2. if B_3(x)=1 return B_2(x) ($b(x)^{-1}$ mod m(x))
3. Q(x) = частное от A_3(x)/B_3(x)
   R(x) = остаток от A_3(x)/B_3(x)
4. [T_1(x), T_2(x), T_3(x)] = [A_1(x)+Q(x)B_1(x), A_2(x)+Q(x)B_2(x), R(x)]
5. [A_1(x), A_2(x), A_3(x)] = [B_1(x), B_2(x), B_3(x)]
   [B_1(x), B_2(x), B_3(x)] = [T_1(x), T_2(x), T_3(x)]
6. goto 2

Рисунок 5 – Расширенный алгоритм Евклида

Напишите функцию gf_mi(b, m, n):

```
def gf_mi(b, m, n):
    """
    INPUTS
    b (integer)– полином, для которого надо найти обратное по умножению
    m (integer) – неприводимый полином
    n (integer)- порядок неприводимого полинома
    OUTPUTS:
    b2 (integer) – полином, обратный по умножению к b
    """
```

Вызов и результат работы этой функции представлен на рис.6.

```
n = 4
a = 3
m = int('10011', 2)
inv_a = gf_mi(a, m, n)
print('a={}'.format(binary(a, n)))
print('inv_a={}'.format(binary(inv_a, n)))
product = gf_multiply_modular(a, inv_a, m, n)
print('a*inv_a={}'.format(binary(product, n)))
```

```
a=0011
inv_a=1110
a*inv_a=0001
```

Рисунок 6

Алгоритм расширения ключа

Ключ шифрования (16 бит): key = $k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15}$
Например, key = 1010011100111011. Разбиваем его на две части w_0 и w_1 (рис.7):

$w_0 = k_0k_1k_2k_3k_4k_5k_6k_7$	$w_1 = k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15}$
10100111	00111011

Рисунок 7

Согласно схеме на рис.9 для выполнения шифрования-расшифрования необходимо еще два дополнительных подключа (w_2, w_3) и (w_4, w_5) (рис.8). Эти ключи получают из ключа шифрования посредством применения алгоритма расширения ключа (рис.10).

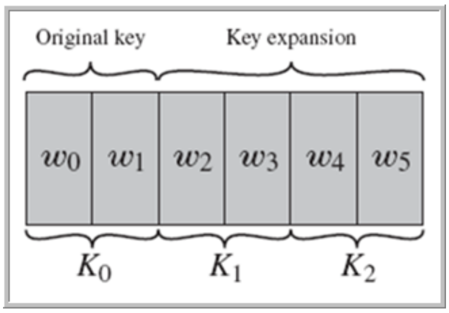

Рисунок 8 – Раундовые ключи

Алгоритм расширения ключа

1. Найти w_2

Как показано на схеме (рис.10) w_2 рассчитывается по следующей формуле:

$$w_2 = w_0 \oplus g(w_1) = w_0 \oplus \mathrm{Rcon}(1) \oplus \mathrm{SubNib}(\mathrm{RotNib}(\mathrm{w}_1)) \qquad (1)$$

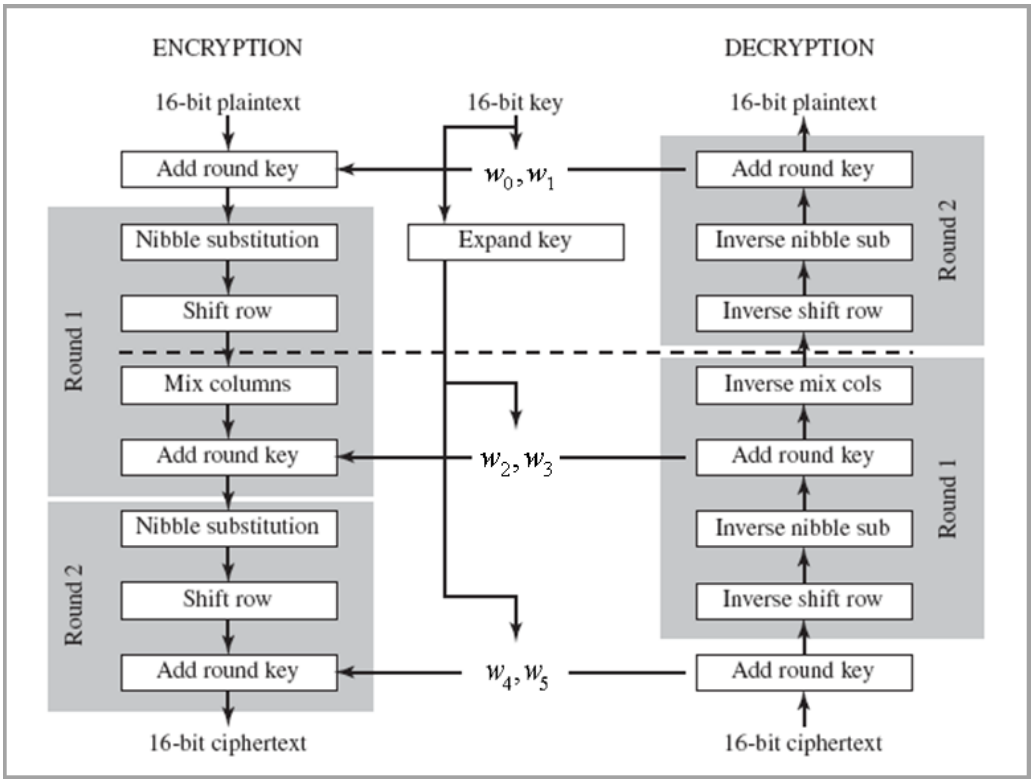

Рисунок 9 – Шифрование-расшифрование

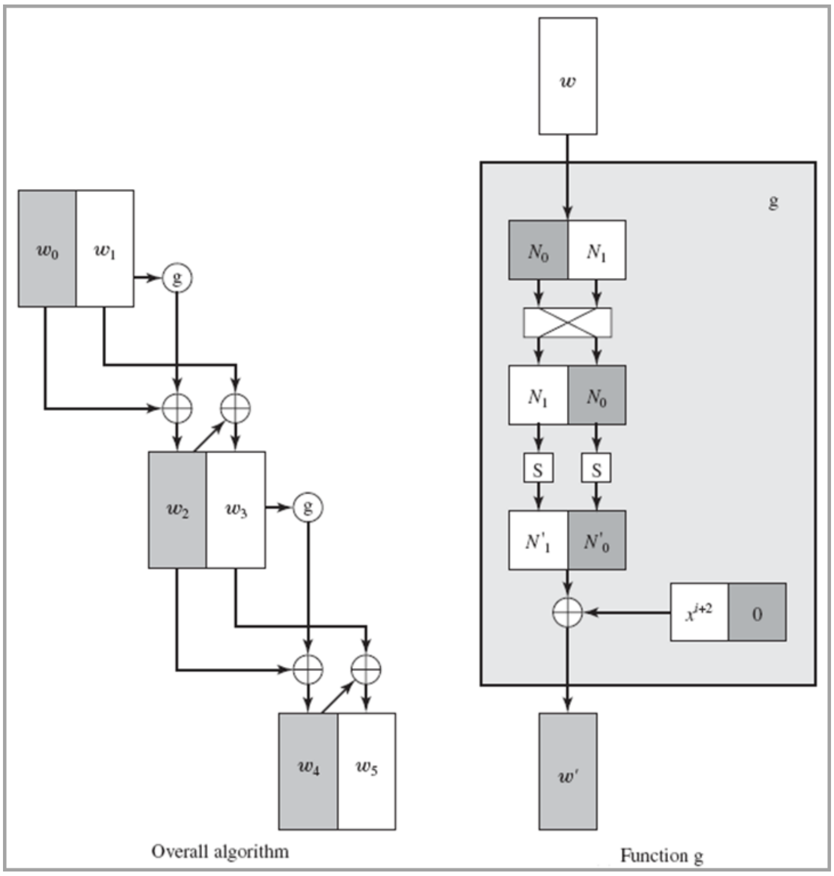

Рисунок 10 – Алгоритм расширения ключа

В этом выражении $\mathrm{Rcon}(1)$ – это 16-битовое число, в котором младшие четыре бита всегда установлены в 0, а старшие 4 бита рассчитываются по формуле:

$$x^{i+2} \bmod m(x) \qquad (2)$$

где i - номер подключа, начиная с 1.

Согласно (2) при $i = 1$

$$x^3 \bmod (x^4 + x + 1) = x^3 = 1000$$

При $i = 2$

$$x^4 \bmod (x^4 + x + 1) = x + 1 = 0011$$

Тогда, $\mathrm{Rcon}(1) = 10000000$, $\mathrm{Rcon}(2) = 00110000$.

Преобразование RotNib(w) меняет местами старшую и младшую 4-битовые части в 8-битовом слове w. Пример, для $w_1 = 00111011$ RotNib(w_1)= 10110011.

Преобразование SubNib – это применение операции замены к двум 4-битовым словам по таблице замен S-box (рис.11).

		j			
		00	**01**	**10**	**11**
i	**00**	9	4	A	B
	01	D	1	8	5
	10	6	2	0	3
	11	C	E	F	7

S-Box

Рисунок 11 – Таблица замен S-Box

При выполнении замены старшие два бита в слове рассматриваются как индекс i строки в таблице замен, а младшие два бита в слове рассматриваются как индекс j столбца в таблице замен. Тогда, результатом операции замены будет слово из таблицы замен на пересечении i-ой строки и j-го столбца. Например, слово 1011 заменится на слово 0011 (0×3), а слово 0011 заменится на 1011 ($0 \times \mathrm{B}$). Таким образом, в результате применения преобразования SubNib к 10110011 получим 00111011.

Рассмотрев все нюансы вычисления части подключа по формуле (2) можно убедиться, что для ключа шифрования key = 1010011100111011 значение w_2 будет таким:

$$
\begin{aligned}
w_2 = w_0 \oplus g(w_1) = &\ 1010\ 0111 \\
&\oplus 1000\ 0000 \\
&\oplus 0011\ 1011 = 00011100
\end{aligned}
$$

2. Найти w_3

Значение w_3 определяется по формуле

$$w_3 = w_2 \oplus w_1$$

Рассмотрим пример. Для key = 1010011100111011, w_1= 00111011 и найденного на предыдущем шаге значения w_2, получаем

$$
\begin{aligned}
w_3 = w_2 \oplus w_1 = &\ 00011100 \\
&\oplus 00111011 = 00100111
\end{aligned}
$$

Получение w_3 означает, что сформирован первый подключ $K_1 = (w_2, w_3)$. В рассматриваемом примере он будет равен $K_1 = 00011100\ 0010\ 0111$

3. Найти (w_4, w_5)

Значения (w_4, w_5) формируют второй подключ K_2. Для их вычисления надо сделать ту же последовательность действий, что и для получения подключа K_1:

$$
\begin{aligned}
&w_4 = w_2 \oplus g(w_3) = w_2 \oplus \mathrm{Rcon}(2) \oplus \mathrm{SubNib}(\mathrm{RotNib}(\mathrm{w}_3)) \\
&w_5 = w_4 \oplus w_3
\end{aligned}
\qquad (3)
$$

В рассматриваемом примере второй подключ будет равен:

$$K_2 = (w_4, w_5) = 0111011001010001$$

Алгоритм шифрования блока данных

Схема шифрования с раундовыми ключами приведена на рис.12.

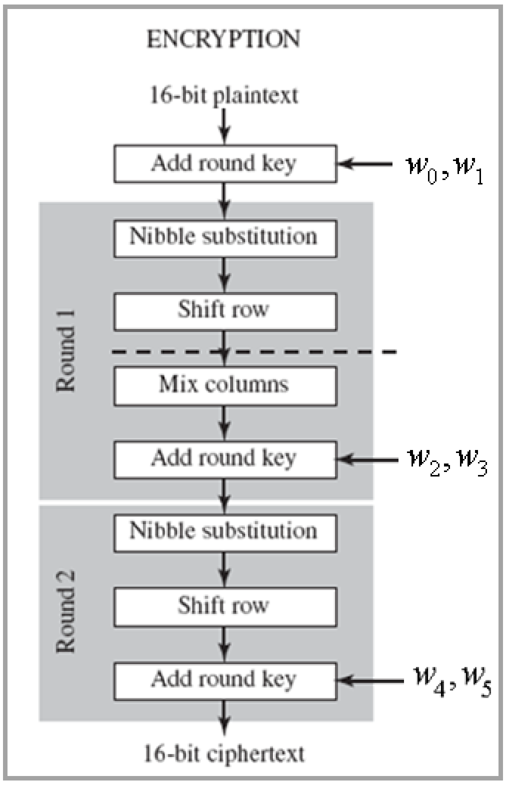

Рисунок 12

Блок данных (16 бит) $P = b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15}$ для шифрования представляется в виде матрицы 2x2 (матрица состояния S) как показано на рис.13.

$$\begin{array}{|c|c|}\hline S_{0,0} & S_{0,1} \\ \hline S_{1,0} & S_{1,1} \\ \hline \end{array} = \begin{array}{|c|c|}\hline b_0 b_1 b_2 b_3 & b_8 b_9 b_{10} b_{11} \\ \hline b_4 b_5 b_6 b_7 & b_{12} b_{13} b_{14} b_{15} \\ \hline \end{array}$$

Рисунок 13 – Матрица состояния шифра

Так, для блока данных P=0110111101101011 матрица состояния будет состоять из элементов так, как это показано на рис.14:

$$\begin{array}{|c|c|}\hline S_{0,0} & S_{0,1} \\ \hline S_{1,0} & S_{1,1} \\ \hline \end{array} = \begin{array}{|c|c|}\hline b_0b_1b_2b_3 = 0110 = 0\times 6 & b_8b_9b_{10}b_{11} = 0110 = 0\times 6 \\ \hline b_4b_5b_6b_7 = 1111 = 0\times \mathrm{F} & b_{12}b_{13}b_{14}b_{15} = 1011 = 0\times \mathrm{B} \\ \hline \end{array}$$

Рисунок 14 – Пример матрицы шифрования

В алгоритме шифрования матрица состояния последовательно обрабатывается посредством применения следующих преобразований:

- Сложение с раундовым ключом (**Add round key**)
- Замена элементов матрицы состояния S (**Nibble Substitution**)
- Перестановка элементов в матрице состояния S (**Shift Row**)
- Перемешивание элементов в столбцах матрицы S (**Mix Columns**).

1. Сложение с раундовым ключом (Add round key)

В виде схемы сложение показано на рис.15.

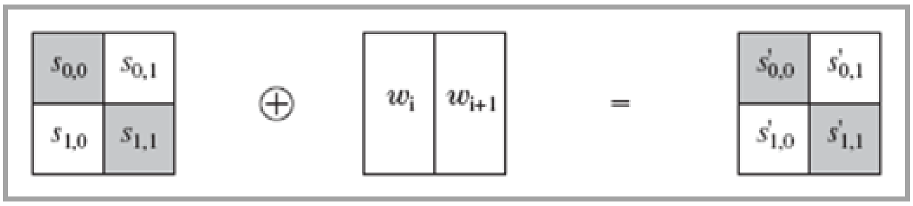

Рисунок 15 - Add round key

Для рассматриваемого примера (key = 1010011100111011 и P=0110111101101011) матрица состояния изменится с

$$\begin{array}{|c|c|}\hline 0\times 6 & 0\times 6 \\ \hline 0\times \mathrm{F} & 0\times \mathrm{B} \\ \hline \end{array} \text{ на } \begin{array}{|c|c|}\hline 0\times \mathrm{C} & 0\times 5 \\ \hline 0\times 8 & 0\times 0 \\ \hline \end{array}$$

2. Замена элементов матрицы состояния S (Nibble Substitution)

На рис.16 в виде схемы показано, как изменяются элементы матрицы состояния в результате применения этого преобразования.

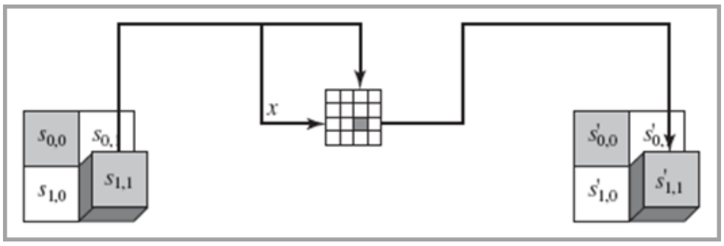

Рисунок 16 – Схема замены

Для рассматриваемого примера матрица состояния изменится с

$0\times C$	0×5
0×8	0×0

на

$0\times C$	0×1
0×6	0×9

3. Перестановка элементов в матрице состояния S (Shift Row)

Какие именно элементы и как они переставляются в строках матрицы состояния показано на рис.17.

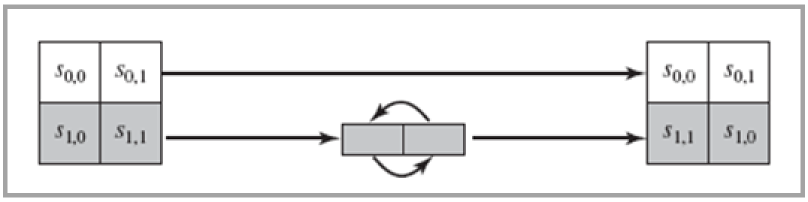

Рисунок 17 – Shift Row

Для рассматриваемого примера матрица состояния изменится с

0×C	0×1
0×6	0×9

на

0×C	0×1
0×9	0×6

4. Перемешивание элементов в столбцах матрицы состояния S (Mix Columns)

Для того, чтобы перемешать элементы в столбцах матрицы состояния применяют умножение этих столбцов слева на матрицу такой же размерности как и матрица состояния шифра (рис.18).

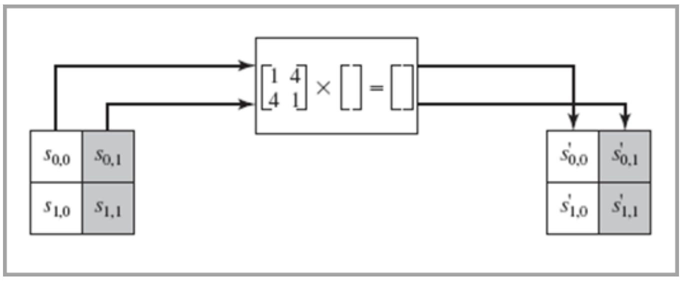

Рисунок 18

Уравнения умножения на заданную матрицу и пример умножения приведены на рис.19.

$$\begin{bmatrix}1 & 4\\4 & 1\end{bmatrix}\begin{bmatrix}s_{0,0} & s_{0,1}\\s_{1,0} & s_{1,1}\end{bmatrix} = \begin{bmatrix}s'_{0,0} & s'_{0,1}\\s'_{1,0} & s'_{1,1}\end{bmatrix}$$

$$\begin{aligned} S'_{0,0} &= S_{0,0} \oplus (4 \cdot S_{1,0}) \\ S'_{1,0} &= (4 \cdot S_{0,0}) \oplus S_{1,0} \\ S'_{0,1} &= S_{0,1} \oplus (4 \cdot S_{1,1}) \\ S'_{1,1} &= (4 \cdot S_{0,1}) \oplus S_{1,1} \end{aligned}$$

$$\begin{bmatrix}1 & 4\\4 & 1\end{bmatrix}\begin{bmatrix}6 & 4\\C & 0\end{bmatrix} = \begin{bmatrix}3 & 4\\7 & 3\end{bmatrix}$$

Рисунок 19

Для рассматриваемого примера матрица состояния изменится с

0×C	0×1
0×9	0×6

на

0×E	0×A
0×C	0×2

Алгоритм расшифрования зашифрованного блока данных

Схема расшифрования с раундовыми ключами приведена на рис.20

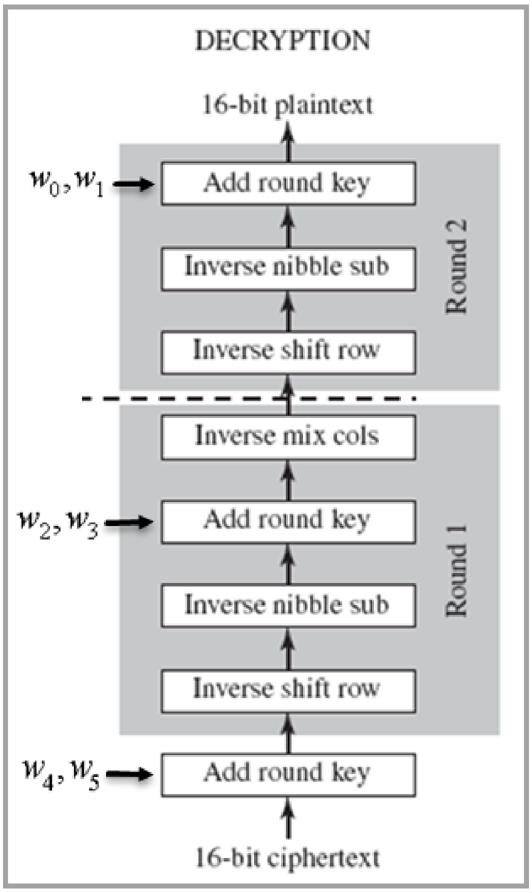

Рисунок 20 – Алгоритм расшифрования

В алгоритме расшифрования матрица шифрования последовательно обрабатывается посредством применения следующих преобразований:

- Сложение с раундовым ключом (**Add round key**)
- Обратная перестановка элементов в матрице S (**Inverse Shift Row**)
- Обратная замена элементов матрицы состояния S (**Inverse Nibble Substitution**)
- Обратное перемешивание элементов в столбцах матрицы состояния S (**Inverse Mix Columns**)

1. Сложение с раундовым ключом по модулю 2 (Add round key)

Точно такое же преобразование, как и при шифровании.

2. Обратная перестановка элементов в матрице состояния S (Inverse Shift Row)

Точно такое же преобразование, как и при шифровании.

3. Обратная замена элементов матрицы состояния S (Inverse Nibble Substitution)

Отличие от прямой замены элементов заключается только в использовании разных таблиц замен. В данном случае используется обратная таблица замен (рис.21).

		j			
		00	**01**	**10**	**11**
i	**00**	A	5	9	B
	01	1	7	8	F
	10	6	0	2	3
	11	C	4	D	E

Inverse S-Box

Рисунок 21 – Обратная таблица замен

4. Обратное перемешивание элементов в столбцах матрицы состояния S (Inverse Mix Columns)

Для обратного преобразования надо использовать обратную матрицу к той, которая использовалась для прямого перемешивания (рис.22).

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} \\ s'_{1,0} & s'_{1,1} \end{bmatrix}$$

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} \\ s_{1,0} & s_{1,1} \end{bmatrix}$$

Рисунок 22

Пример реализации

Ниже приведен пример реализации алгоритма расширения ключа и основных функций для алгоритма шифрования. Реализовать алгоритм шифрования и расшифрования массива 16-ти битовых чисел. В приведенной реализации предполагается, что доступны функции mux, divide_into_two, gf_multiply_modular.

```
class AES():
    S_Box = np.array([['9', '4', 'a', 'b'], ['d', '1', '8', '5'], ['6', '2', '0', '3'], ['c', 'e', 'f', '7']])
    S_InvBox = np.array([['a', '5', '9', 'b'], ['1', '7', '8', 'f'], ['6', '0', '2', '3'], ['c', '4', 'd', 'e']])
    RCON1 = int('10000000', 2)
    RCON2 = int('00110000', 2)

    modulus = int('10011', 2)
    column_Matrix = list([['1', '4'], ['4', '1']])
    column_InvMatrix = list([['9', '2'], ['2', '9']])
    state_matrix = []

    def __init__(self):
        """
        раундовые ключи. рассчитываются в функции key_schedule
        """
        pass

    def sbox(self, v):
        """
        Замена 4-битового значения по таблице S_Box
        """
        r, c = divide_into_two(v, 4)
        rez = self.S_Box[r, c]
        return int(rez, 16)

    def g(self, w, i):
        """
        g функция в алгоритме расширения ключа
        """
        n00, n11 = divide_into_two(w, 8)
        n0 = self.sbox(n00)
        n1 = self.sbox(n11)
        n1n0 = mux(n1, n0, 4)
        if i == 1:
            rez = n1n0 ^ self.RCON1
        else:
            rez = n1n0 ^ self.RCON2
        return rez

    def key_expansion(self, key):
        """
        Алгоритм расширения ключа
        """
```

```
        w0, w1 = divide_into_two(key, 16)
        w2 = w0 ^ self.g(w1, 1)

        w3 = w1 ^ w2
        w4 = w2 ^ self.g(w3, 2)
        w5 = w3 ^ w4
        return key, mux(w2, w3, 8), mux(w4, w5, 8)

    def to_state_matrix(self, block):
        """
        Формирование матрицы состояния из 16-ти битового числа
        """
        b1, b2 = divide_into_two(block, 16)
        b11, b12 = divide_into_two(b1, 8)
        b21, b22 = divide_into_two(b2, 8)
        self.state_matrix = [[b11, b21], [b12, b22]]

    def add_round_key(self, k):
        """
        Сложение с раундовым ключом (Add round key)
        """
        k1, k2 = divide_into_two(k, 16)
        k11, k12 = divide_into_two(k1, 8)
        k21, k22 = divide_into_two(k2, 8)

        self.state_matrix[0][0] ^= k11
        self.state_matrix[1][0] ^= k12
        self.state_matrix[0][1] ^= k21
        self.state_matrix[1][1] ^= k22

    def nibble_substitution(self):
        """
        Замена элементов матрицы состояния S (Nibble Substitution)
        """
        self.state_matrix[0][0] = self.sbox(self.state_matrix[0][0])
        self.state_matrix[0][1] = self.sbox(self.state_matrix[0][1])
        self.state_matrix[1][0] = self.sbox(self.state_matrix[1][0])
        self.state_matrix[1][1] = self.sbox(self.state_matrix[1][1])

    def shift_row(self):
        """
        Перестановка элементов в матрице состояния S (Shift Row)
        """
        a = self.state_matrix[1][0]
        self.state_matrix[1][0] = self.state_matrix[1][1]
        self.state_matrix[1][1] = a

    def mix_columns(self):
        """
        Перемешивание элементов в столбцах матрицы S (Mix Columns)
        """
        m00 = int(self.column_Matrix[0][0], 16)
        m01 = int(self.column_Matrix[0][1], 16)
        m10 = int(self.column_Matrix[1][0], 16)
        m11 = int(self.column_Matrix[1][1], 16)

        st00 = self.state_matrix[0][0]
        st10 = self.state_matrix[1][0]
        a = gf_multiply_modular(m00, st00, self.modulus, 4)
        b = gf_multiply_modular(m01, st10, self.modulus, 4)
```

```
            c = gf_multiply_modular(m10, st00, self.modulus, 4)
            d = gf_multiply_modular(m11, st10, self.modulus, 4)
            self.state_matrix[0][0] = a ^ b
            self.state_matrix[1][0] = c ^ d

            st00 = self.state_matrix[0][1]
            st10 = self.state_matrix[1][1]
            a = gf_multiply_modular(m00, st00, self.modulus, 4)
            b = gf_multiply_modular(m01, st10, self.modulus, 4)
            c = gf_multiply_modular(m10, st00, self.modulus, 4)
            d = gf_multiply_modular(m11, st10, self.modulus, 4)
            self.state_matrix[0][1] = a ^ b
            self.state_matrix[1][1] = c ^ d


        def from_state_matrix(self):
            """
            Формирование 16-ти битового числа из матрицы состояния
            """
            b1 = mux(self.state_matrix[0][0], self.state_matrix[1][0], 4)
            b2 = mux(self.state_matrix[0][1], self.state_matrix[1][1], 4)
            return mux(b1, b2, 8)

        def encrypt(self, plaintext, k0, k1, k2):
            """
            Алгоритм шифрования блока с заданными раундовыми ключами
            """
            pass

        def decrypt(self, ciphertext, k0, k1, k2):
            """
            Алгоритм расшифрования блока с заданными раундовыми ключами
            """

        def encrypt_data(self, data, key):
            """
            шифрование 8-битовых чисел в data на ключе key
            """
            k0, k1, k2 = self.key_expansion(key)
            pass


        def decrypt_data(self, data, key):
            """
            шифрование 8-битовых чисел в data на ключе key
            """
            k0, k1, k2 = self.key_expansion(key)
            pass
```

Следует обратить внимание, что в этом примере для заданной матрицы, которая участвует в процедуре перемешивания элементов в столбцах матрицы состояния (Mix Columns) $\begin{pmatrix} 1 & 4 \\ 4 & 1 \end{pmatrix}$ задана и ее обратная матрица

$\begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix}$. Надо написать функцию, которая вычисляет обратную матрицу размерности 2x2 для заданной матрицы. Для этого понадобится рассмотренная ранее функция gf_mi.

Тест

Для ключа K = 1010011100111011 должны быть сформированы следующие раундовые ключи:

k0=1010011100111011

k1=0001110000100111

k2=0111011001010001

Для блока данных P = 0110111101101011 должна быть сформирована матрица состояния:

0x6 0x6

0xf 0xb

Матрица состояния после процедуры «Сложение с раундовым ключом (Add round key)»:

0xc 0x5

0x8 0x0

Матрица состояния после процедуры «Замена элементов матрицы состояния S (Nibble Substitution)»:

0xc 0x1

0x6 0x9

Матрица состояния после процедуры «Перестановка элементов в матрице состояния S (Shift Row)»:

0xc 0x1

0x9 0x6

Матрица состояния после процедуры «Перемешивание элементов в столбцах матрицы S (Mix Columns)»:

0xe 0xa

0xc 0x2

После выполнения всех процедур в алгоритме шифрования матрица состояния (результат шифрования):

0x0 0x3

0x7 0x8

Зашифрованной матрице состояния соответствует зашифрованный блок данных: 0000011100111000

Задания по обработке файлов алгоритмом S-AES

1. Расшифровать файл dd1_saes_c_all.bmp – зашифрованное шифром S_AES изображение в формате bmp. Матрица для преобразования MixColumns: [['1', '4'], ['4', '1']]. Неприводимый многочлен: x^4+x+1. Режим шифрования ECB. Ключ равен 834. Зашифровать в режиме ECB, оставив первые 50 байт без изменения.

2. Расшифровать файл im43_saes_c_all.bmp – зашифрованное шифром S_AES изображение в формате bmp. Матрица для преобразования MixColumns: [['b', '4'], ['e', 'd']]. Неприводимый многочлен: x^4+x+1. Режим шифрования ECB. Ключ равен 2318. Зашифровать в режиме ECB, оставив первые 50 байт без изменения.

3. Расшифровать файл dd5_saes_cbc_c_all.bmp – зашифрованное шифром S_AES изображение в формате bmp. Матрица для преобразования MixColumns: [['a', 'c'], ['8', '6']]. Неприводимый многочлен: x^4+x+1. Режим шифрования CBC. Ключ равен 1021. Вектор инициализации равен 456. Зашифровать, оставив первые 50 байт без изменения.

4. Расшифровать файл dd8_saes_ofb_c_all.bmp – зашифрованное шифром S_AES изображение в формате bmp. Матрица для преобразования MixColumns: [['5', '3'], ['2', 'c']]. Неприводимый многочлен: x^4+x^3+1. Режим шифрования OFB. Ключ равен 12345. Вектор инициализации равен 5171. Зашифровать, оставив первые 50 байт без изменения.

5. Дешифровать файл t20_saes_ofb_c_all.txt. Шифр SAES. Режим OFB. Известны младшие биты ключа: 011110110, вектор инициализации 3523,

Матрица для преобразования MixColumns: [['3', '8'], ['2', 'b']]. Неприводимый многочлен: x^4+x+1.

6. Расшифровать файл dd10_saes_cfb_c_all.bmp – зашифрованное шифром S_AES изображение в формате bmp. Матрица для преобразования MixColumns: [['7', 'd'], ['4', '5']]. Неприводимый многочлен: x^4+x+1. Режим шифрования CFB. Ключ равен 24545. Вектор инициализации равен 9165. Зашифровать, оставив первые 50 байт без изменения.

7. Расшифровать файл dd12_saes_ctr_c_all.bmp – зашифрованное шифром S_AES изображение в формате bmp. Матрица для преобразования MixColumns: [['7', '3'], ['2', 'e']]. Неприводимый многочлен: x^4+x+1. Режим шифрования CTR. Ключ равен 2645. Вектор инициализации равен 23184. Зашифровать, оставив первые 50 байт без изменения.

Литература

[1] Raphael Chung-Wei Phan, «Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students», Published in Cryptologia, XXVI (4), 2002

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