«СТРУМОК»

«СТРУМОК» - генератор ключевых потоков, криптостойкость которого основана на регистре сдвига с линейной обратной связью, полиномиальном умножении в поле и нелинейной функцией FSM.

«СТРУМОК» использует 64 битную разрядность, то есть: выходной поток кратный 64, состояния РСЛОС также по 64 и прочие операции использую 64 битные слова.

Входными параметрами ПСШ являются: 512 бит секретного ключа *K*, 256 бит вектора инициализации *IV*. Данные параметры необходимы для инициализации начального состояния РСЛОС, состоящего из 16 64 битных регистров (*a[16]*), и 2 регистров состояния нелинейной функции FSM (*b[2]*).

Выходным параметром, является ключевой поток 64 разрядных слов – STR. Рекомендованная длина выходного потока 2^{50} 64 разрядных слов.

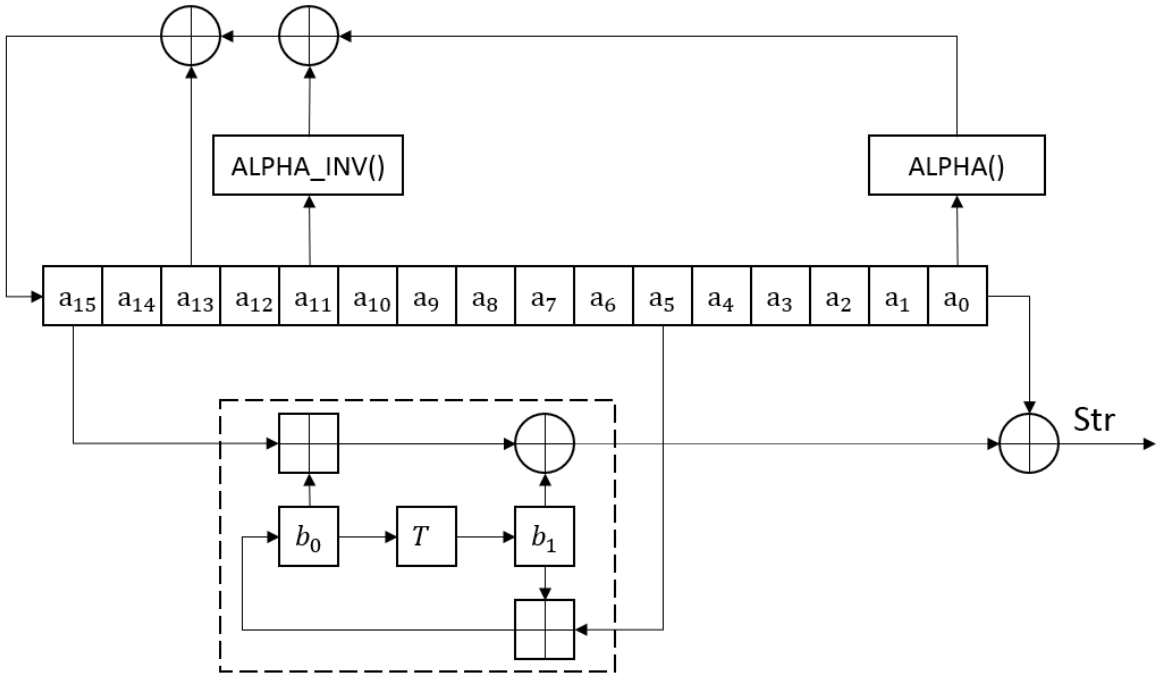

Рисунок М.0 – Схематическое изображение выполнения алгоритма «СТРУМОК»

В алгоритме данного шифра используются такие функции:

1. Инициализации (INIT()) – преобразовывает входные параметры, такие как 8 слов ключа (*Kay[8]*) и 4 слова вектора инициализации (*IV[4]*), в начальное состояние регистров (S_0), а именно: 16 слов регистра РСЛОС (*a[16]*), и 2 регистра состояния FSM (*b[2]*);

INIT(): Вход: UINT_64 Key[8], IV[4];
Выход: S_0 – (Начальное состояние (значения регистров a[i] и b[i]);

a)
```
for(j=0;j<8;j++)
{
    a[15-j]=Key[7-j];
    a[7-j]=~Key[7-j];
}
```

Присвоить регистрам $a_{15}, a_{14}, \dots, a_8$ значения $Key_7, Key_6, \dots, Key_0$, а регистрам $a_7, a_6, \dots, a_0$ значения $\sim Key_7, \sim Key_6, \dots, \sim Key_0$ соответственно, где $\sim Key_7$ – побитовая инверсия значения Key_7

b) a[15]^=IV[0];
a[12]^=IV[1];
a[10]^=IV[2];
a[9] ^=IV[3];

Регистры $a_{15}, a_{12}, a_{10}, a_9$ заменить на значения $a_{15} \oplus IV_0, a_{12} \oplus IV_1, a_{10} \oplus IV_2, a_9 \oplus IV_3$ соответственно

c) b[0]=0;
b[1]=0;
Обнулить значения регистров $b_0, b_1,$

d) for(i=31;i>=0;i--)
$Next(S_i, INIT)$;

32 раза выполнить функцию NEXT (с модификацией для инициализации)

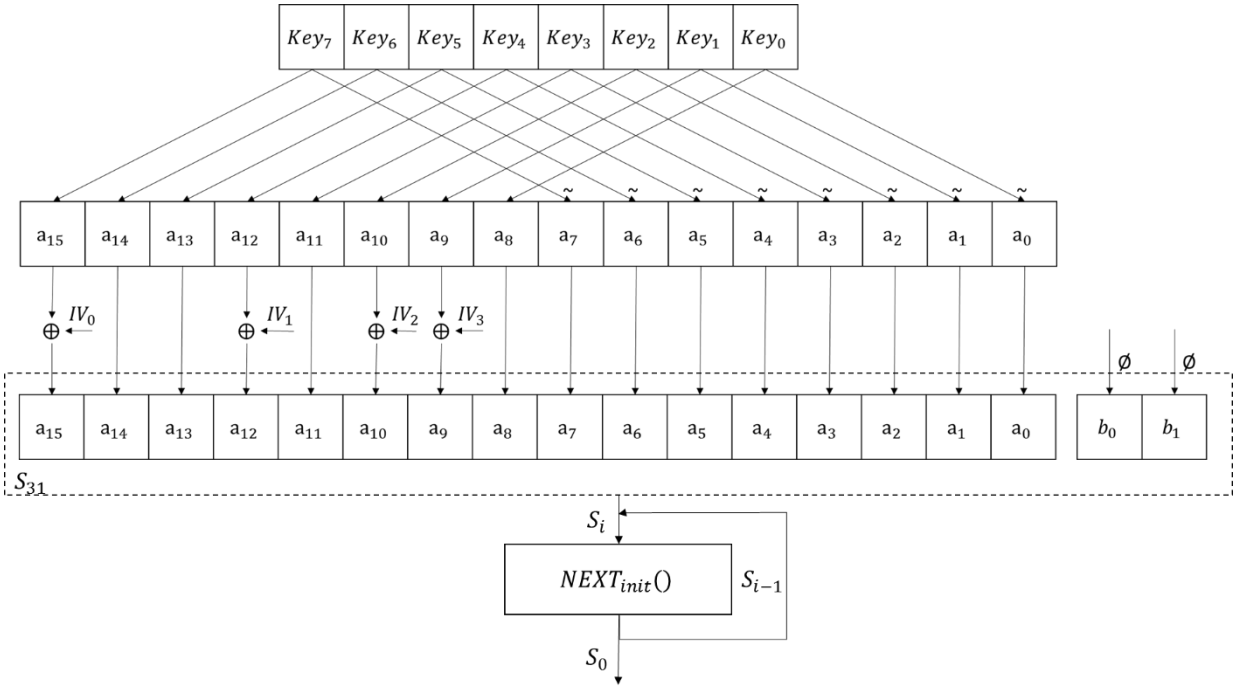

Рисунок М.1 – Схематическое изображение инициализации начального состояния

2. Генерации следующего состояния (NEXT()) – в качестве входного параметра принимает состояния регистров (S_i) и режим работы (для инициализации, для получения ключевого потока), преобразовывает значения на одну итерацию вперед (S_{i+1}), для режима инициализации и получения ключевого потока данная функция имеет различия;

NEXT(): Вход: S_i – (Состояние регистров (а именно регистров a[i] и b[i]), Вкл/Выкл модификация для инициализации;

Выход: S_{i+1};

a) b[1]=T(b[0]);
Меняем состояние регистра b_1, как $b_1 = T(b_0)$.

b) b[0]=b[1]+a[5];
Меняем состояние регистра b_0, как $b_0 = b_1 + a_5 mod(2^{64})$.

c) UINT_64 tmp = a[15]; //not to lose
Для того что бы не потерять значение регистра при сдвиге, необходимо временно данное значение сохранить $tmp = a_{15}$.

d) if(INIT mode)
a[15] = (a[0]*alpha)^a[13]^(a[11]*alpha_inv)^FSM(a[15],b[0],b[1]);
else { a[15]= (a[0]*alpha)^a[13]^(a[11]*alpha_inv); }
Если функция NEXT() используется для инициализации, то есть $NEXT_{init}()$, то регистр $a_{15} = ALPHA(a_0) \oplus a_{13} \oplus ALPHA\_INV(a_{11}) \oplus FSM(a_{15}, b_0, b_1)$, иначе $a_{15} = ALPHA(a_0) \oplus a_{13} \oplus ALPHA\_INV(a_{11})$.

e) for(i=0;i<14;i++)
a[i]=a[i+1];
a[14]=tmp;
Сдвигаем оставшиеся значения регистров таким образом: $a_0 = a_1; a_1 = a_2; \dots; a_{13} = a_{14}$ и так как a_{15} был изменен ранее, то $a_{14} = tmp$.

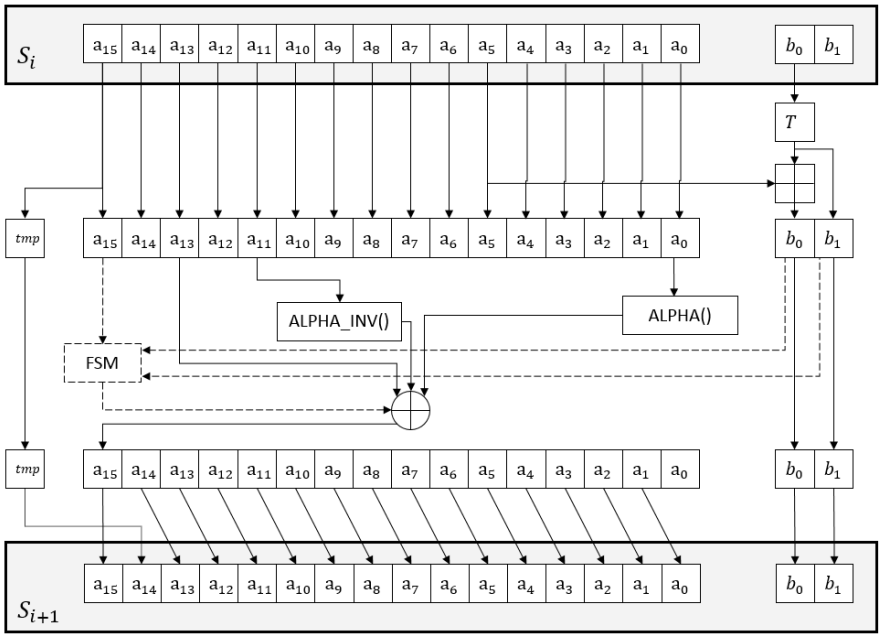

Рисунок М.2 – Схематическое изображение генерации последующего состояния (FSM() только для функции $NEXT_{init}()$)

3. Получения 64 битного ключевого потокового слова (*STR()*) – входным параметром является текущее состояние регистров (S_i), а выходным – 64 бита ключевого потока;

STR(): Вход: S_i – (Состояние регистров (значения регистроы a[i] and b[i]);
Выход: UINT_64 Str (64 быта ключевого потока);

a) return Str = FSM(a[15],b[0],b[1])^a[0];

Выходное ключевое слово $Str = FSM(a_{15}, b_0, b_1) \oplus a_0$. Функция STR() не меняет состояния регистров, а лишь выводит 64 бита потока для состояния в данной итерации, поэтому для получения потока большей длины необходимо чередовать функции генерации следующего состояния NEXT() с функцией получения ключевого потока STR().

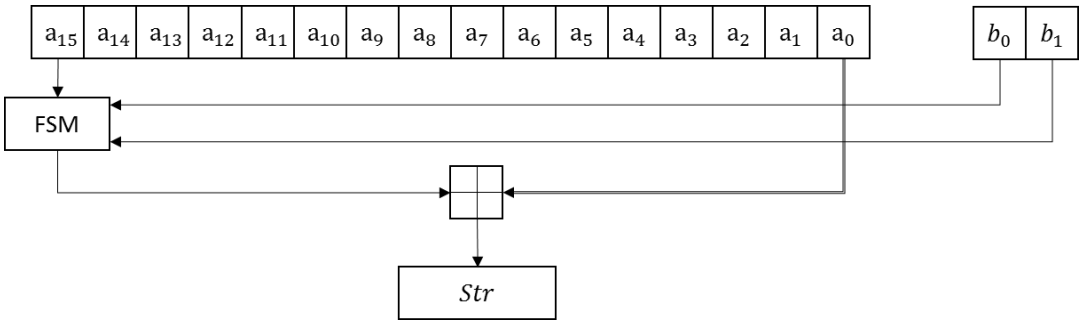

Рисунок М.3 – Схематическое изображение получения 64 бит ключевого слова

4. Автомат с конечным состоянием (FSM()) – вспомогательная функция (используется в $NEXT_{init}$ и STR), на вход принимает регистры a, b_0, b_1, а на выход 64 битный регистр.

FSM(): Вход: a,b[0],b[1];
Выход: UINT_64 q;

a) return q =(a+b[0])^b[1];

Функция конечного автомата выдает на выходе регистр q как $q = (a + b_0) mod(2^{64}) \oplus b_1$. Данный регистр используется для генерации ключевого потока либо для генерации нового состояния регистра a_{15}.

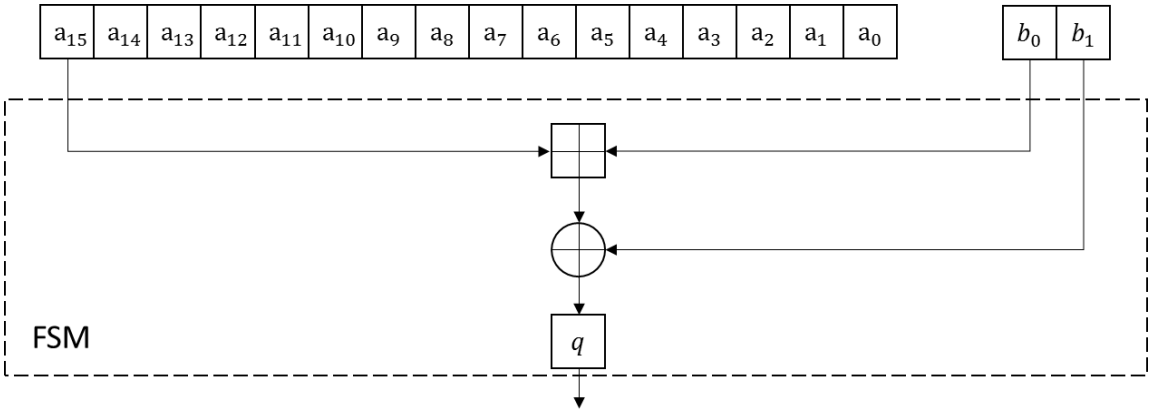

Рисунок М.4 – Схематическое изображение FSM()

5. Функция подстановки (T()) – вспомогательная функция (используется в FSM()), на вход принимает 64 битный регистр b_0, на выходе 64 бита для регистра b_1. Представим b_1 в виде вектора байт$(b_{1_0}, b_{1_1}, \dots, b_{1_7})$. Для каждого b_{1_i} выполним подстановку используя соответствующую таблицу из шифра Калина. Таким образом, что $b_{1_i} = \Pi_{i\ mod(4)}(b_{1_i})$, где $i = 0..7$. После подстановки получим вектор $(b_{1_0}, b_{1_1}, \dots, b_{1_7})$ с новыми значениями, который представляем в виде матрицы с одним столбцом. Данную матрицу умножаем на матрицу, сгенерированную вектором $v = (0x01, 0x01, 0x05, 0x01, 0x08, 0x06, 0x07, 0x04)$, как результат МДР-кода. Перемножения матриц b_1 и V выполняется в поле *GF(2)[x],* где *f(x) = $x^8 + x^4 + x^3 + x + 1$*.

$$\begin{matrix} b_{1_0} \to \Pi_0 \to \\ b_{1_1} \to \Pi_1 \to \\ b_{1_2} \to \Pi_2 \to \\ b_{1_3} \to \Pi_3 \to \\ b_{1_4} \to \Pi_0 \to \\ b_{1_5} \to \Pi_1 \to \\ b_{1_7} \to \Pi_2 \to \\ b_{1_7} \to \Pi_3 \to \end{matrix} \begin{vmatrix} b_{1_0} \\ b_{1_1} \\ b_{1_2} \\ b_{1_3} \\ b_{1_4} \\ b_{1_5} \\ b_{1_7} \\ b_{1_7} \end{vmatrix} \times \begin{vmatrix} 0x01 & 0x01 & 0x05 & 0x01 & 0x08 & 0x06 & 0x07 & 0x04 \\ 0x04 & 0x01 & 0x01 & 0x05 & 0x01 & 0x08 & 0x06 & 0x07 \\ 0x07 & 0x04 & 0x01 & 0x01 & 0x05 & 0x01 & 0x08 & 0x06 \\ 0x06 & 0x07 & 0x04 & 0x01 & 0x01 & 0x05 & 0x01 & 0x08 \\ 0x08 & 0x06 & 0x07 & 0x04 & 0x01 & 0x01 & 0x05 & 0x01 \\ 0x01 & 0x08 & 0x06 & 0x07 & 0x04 & 0x01 & 0x01 & 0x05 \\ 0x05 & 0x01 & 0x08 & 0x06 & 0x07 & 0x04 & 0x01 & 0x01 \\ 0x01 & 0x05 & 0x01 & 0x08 & 0x06 & 0x07 & 0x04 & 0x01 \end{vmatrix} = \begin{vmatrix} b_{1_0} \\ b_{1_1} \\ b_{1_2} \\ b_{1_3} \\ b_{1_4} \\ b_{1_5} \\ b_{1_7} \\ b_{1_7} \end{vmatrix}$$

6. Полиномиальное умножение (ALPHA()) – на вход подается регистр a_0, на выходе получаем результат полиномиального перемножения $\alpha \otimes a_0$ в поле $GF(2^{64})$. Особенностью данного умножение является то что, мы имеем таблицу предвычисления M_a, и умножения выполняется путем сдвига входного 64-битного слова (вектора), так что бы получить вектор $(0, c_0, c_1, c_2, ..., c_6)$ из вектора $(c_0, c_1, c_2, ..., c_7)$, далее сложить по модулю 2, к полученному вектору, рядок $M_a(C_7)$ построенной таблицы. То есть $a_0 \times \alpha = (a_0 \gg 8) \oplus (M_\alpha[a_0 \gg 56] \ll 56)$.
7. Инверсное полиномиальное умножение (ALPHA_INV()) – на вход подается регистр a_{11}, на выходе получаем результат полиномиального перемножения $\alpha^{-1} \otimes a_{11}$ в поле $GF(2^{64})$. Операция аналогична полиномиальному умножению (ALPHA()), но используется таблица для инверсного альфа $M_{\alpha^{-1}}$, то есть $a_{11} \times \alpha^{-1} = (a_0 \gg 8) \oplus (M_{\alpha^{-1}}[a_{11} \gg 56] \ll 56)$

Input: UINT_64 Key[8], IV[4];
Output: UINT_64 Stream;
Interim value: UINTT_64 a[16] – LFSR states, b[2] – FSM states

FUNCTION:

INIT():

Input: UINT_64 Key[8], IV[4];

Output: S_0 – (Start state (values and position a[i] and b[i]);

e) for(j=0;j<8;j++)
{
a[15-j]=Key[7-j];
a[7-j]=~Key[7-j];
}

f) a[15]^=IV[0];
a[12]^=IV[1];
a[10]^=IV[2];
a[9] ^=IV[3];

g) b[0]=0;
b[1]=1;

h) for(i=31;i>=0;i--)
$Next(S_i, INIT)$;

NEXT():

Input: S_i – (Start state (values and position a[i] and b[i]), INIT mode;

Output: S_{i+1};

f) b[2]=T(b[1]);

g) b[1]=b[2]+a[5];

h) UINT_64 tmp = a[15]; //not to lose

i) if(INIT mode)

a[15] = (a[0]*alpha)^a[2]^(a[11]*alpha_inv)^FSM(a[15],b[0],b[1]);

else { a[15]= (a[0]*alpha)^a[2]^(a[11]*alpha_inv); }

j) for(i=0;i<14;i++)

a[i]=a[i+1];

a[14]=tmp;

STR():

Input: S_i – (Start state (values and position a[i] and b[i]);

Output: UINT_64 Str;

b) return Str = FSM(a[15],b[0],b[1])^a[0];

T(): /*Уточнить – узнать*/

Input: w;

Output q;

a) q=SUB[w]

Alpha():

/*Уточнить – узнать*/

Alpha_Inv():

/*Уточнить – узнать*/

FSM():

Input: a,b[0],b[1];

Output: UINT_64 q;

b) return q =(a+b[0])^b[1];

Общий алгоритм работы:

Key[8] = {0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7};

IV[4] = {0x0, 0x1, 0x2, 0x3};

a[16] = {0x0, …, 0x0};

b[2] = {0x0, 0x0};

N = 100; // определяет пользователь

Str[N];

```
INIT(Key, IV);
for(i=0;i<N;i++)
{
    Str[i]=STR();
    NEXT(null); //null – режим работы не для INIT()
}
```