

# АЛГОРИТМ БЛОЧНОГО СИММЕТРИЧНОГО ШИФРОВАНИЯ «ЛАБИРИНТ»

*к.т.н. Головашич С.А.*

## ВВЕДЕНИЕ

Специфика информационных потоков, циркулирующих в современных ИТ-системах, часто требует обеспечения конфиденциальности передаваемых данных, что, проще всего, достигается путём применения средств криптографической защиты информации (КЗИ). Неотъемлемым элементом подсистемы КЗИ в компьютерных системах стали блочные симметричные шифры (БСШ). Последнее десятилетие ознаменовалось эволюционной сменой стандартов в области алгоритмов КЗИ, которая не обошла и БСШ. Так, новый международный стандарт БСШ FIPS-197 [1] (победитель проекта AES [1] – алгоритм Rijndael), сегодня фактически вытеснил, широко распространённый ещё 10 лет назад, стандарт блочного шифрования DES и его 3-х каскадную версию – Triple-DES [2].

В настоящее время на Украине отсутствует национальный стандарт БСШ, и временно разрешён к применению стандарт бывшего СССР – ГОСТ 28147-89 [3], принятый в 1989 году. Однако, учитывая значительный прогресс в сфере методов и средств криптоанализа, на протяжении последних 10–20 лет, этот алгоритм уже переходит в класс “морально устаревших”. Косвенным подтверждением этого факта стали результаты проведенных в США и Европе конкурсов (проекты AES [4] и NESSIE [5] соответственно), которые были направлены на выбор новых криптопримитивов, удовлетворяющих возросшим требованиям безопасности. Так, в соответствии с требованиями проекта NESSIE, алгоритм ГОСТ 28147-89 [3] относится только к третьему (наименьшему) классу стойкости. Поэтому проблема разработки отечественного стандарта БСШ, удовлетворяющего наиболее жёстким требованиям безопасности и учитывающего последние достижения в области методов криптоанализа и криптозащиты симметричных алгоритмов, является весьма актуальной для Украины. Основной целью разработки алгоритма «Лабиринт» было решение указанной проблемы.

Предлагаемый в данной работе алгоритм «Лабиринт» является результатом эволюционного развития серии алгоритмов БСШ, разработанных автором в ходе работы над диссертацией (первый алгоритм – «Шторм») и последующих исследований в области проектирования БСШ (алгоритм «Торнадо» [6,7,8], версии 1–3). Все указанные алгоритмы, построены в соответствии с принципами и методами изложенными в диссертационной работе автора [9], эти принципы вобрали в себя опыт накопленный в рамках анализа кандидатов проектов AES и NESSIE. Они неоднократно представлялись и обсуждались на специализированных конференциях и семинарах, что позволило учесть мнения множества экспертов-криптографов, присутствовавших на них. Автор выражает благодарность всем участникам обсуждений указанных алгоритмов, эти дискуссии позволили создать алгоритм «Лабиринт» таким, каким он есть сейчас. Автор выражает особую благодарность научному руководителю кандидатской диссертационной работы д.т.н., профессору Горбенко Ивану Дмитриевичу за постоянную «корректировку курса» научных исследований, а также д.т.н., профессору Долгову Виктору Ивановичу за посвящение в «тайнства» мира криптоанализа БСШ, на заре моей научной деятельности. Кроме того, автор выражает благодарность своим коллегам к.т.н. Руженцеву В.И. и к.т.н. Лепехе А.Н. за помощь в независимом криптографическом и статистическом анализе различных версий алгоритма «Торнадо».

Основным принципом построения нового алгоритма БСШ «Лабиринт» стало использование предельно простых и хорошо апробированных базовых компонентов и

методов построения блочных шифров. Применение данного подхода не только позволило создать простой в реализации алгоритм, но также обеспечило «прозрачность» структуры криптоалгоритма и применимость известных методов оценки стойкости блочных шифров.

## 1. ОБОЗНАЧЕНИЯ И СОГЛАШЕНИЯ

Алгоритм «Лабиринт» является итеративным «инволютивным» шифром [10] и предусматривает три варианта длины блока (для применения в различных классах безопасности) и переменную длину ключа шифрования для каждого варианта длины блока.

Структура алгоритма «Лабиринт» идентична для различных длин блока, отличие заключается только в необходимом числе циклов шифрования. Поэтому описание алгоритма приводится в общем виде для блока длиной  $128n$  бит, где  $n$  – коэффициент кратности длины блока, который может принимать значения **1**, **2** или **4**, т.е. алгоритм поддерживает блоки длиной **128**, **256** и **512** бит. Для каждой длины блока алгоритм поддерживает 3 значения длины ключа в диапазоне от  $128n$  до  $256n$  с шагом  $64n$  бита (т.е. от 2 до 4 полублоков включительно).

Префикс **0x** будем использовать для обозначения шестнадцатеричных чисел. Например:  $0x1AF = 431$ .

### 1.1 Список символов и обозначений

$\mathbf{Z}_p$	– кольцо целых чисел по модулю $p$ ;
$\mathbf{F}_p$	– поле порядка $p$ (запись эквивалентна $\text{GF}(p)$ );
$\mathbf{B}$	– векторное пространство 8-битных элементов (байтов), $\mathbf{B} = \text{GF}(2)^8 = \{0,1\}^8$ ;
$\mathbf{W}$	– векторное пространство 64-битных элементов (слов), $\mathbf{W} = \mathbf{B}^8 = \text{GF}(2)^{64} = \{0,1\}^{64} = \mathbf{Z}_{2^{64}}$ ;
$\mathbf{H}$	– векторное пространство $64n$ -битных элементов (полублоков), $\mathbf{H} = \mathbf{W}^n = \text{GF}(2)^{64n} = \{0,1\}^{64n}$ ;
$\mathbf{T}$	– векторное пространство $128n$ -битных элементов (блоков), $\mathbf{T} = \mathbf{H}^2 = \text{GF}(2)^{128n} = \{0,1\}^{128n}$ ;
$\oplus$	– побитовая операция «исключающее ИЛИ» (XOR, сложение по модулю 2);
$\times$	– операция умножения матрицы на вектор (элементы принадлежат полю $\text{GF}(2^8)$ );
$+$	– сложение в кольце $\mathbf{Z}_{2^m}$ ;
$-$	– вычитание в кольце $\mathbf{Z}_{2^m}$ ;
$\lll l$	– циклический сдвиг на $l$ бит влево;
$\ggg l$	– циклический сдвиг на $l$ бит вправо;
$\text{mod } m$	– остаток от деления на $m$ ;
$x \parallel y$	– конкатенация операндов $x$ и $y$ ( $x$ – старшая часть, $y$ – младшая часть).

Для обозначения совокупностей элементов, представляющих единое целое, будем использовать два способа записи:

$(x_0, \dots, x_m)$	– способ записи множества либо последовательности элементов, для которых порядок взаимного расположения в памяти шифратора значения не имеет (с точки зрения применяемых операций);
$\langle x_m \parallel \dots \parallel x_0 \rangle$	– способ записи вектора, который отражает необходимый порядок расположения его элементов в памяти шифратора («младшие» справа).

Отдельные разряды будем обозначать  $b_i$ , байты  $B_i$ , а слова  $W_i$ , используя при этом *Little Indian* нотацию, т.е. нумерация разрядов, байтов, слов и т.д. выполняется от младших «весов» к старшим: разряд  $b_i$  имеет «вес»  $2^i$ , а байт  $B_i$  – «вес»  $2^{8i}$  и т.д.

Например:

$$\begin{aligned} X \in \mathbf{H} = \mathbf{W}^4 (n = 4), \quad W_i \in \mathbf{W}, \quad B_i \in \mathbf{B}, \quad b_i \in \{0,1\} &\Rightarrow \\ X = \langle W_3 \parallel W_2 \parallel W_1 \parallel W_0 \rangle = \langle B_{31} \parallel \dots \parallel B_1 \parallel B_0 \rangle = \langle b_{255} \parallel \dots \parallel b_1 \parallel b_0 \rangle, \\ W_0 = \langle B_7 \parallel \dots \parallel B_1 \parallel B_0 \rangle = \langle b_{63} \parallel \dots \parallel b_1 \parallel b_0 \rangle, \\ W_1 = \langle B_{15} \parallel \dots \parallel B_9 \parallel B_8 \rangle = \langle b_{127} \parallel \dots \parallel b_{65} \parallel b_{64} \rangle, \\ B_0 = \langle b_7 \parallel \dots \parallel b_1 \parallel b_0 \rangle, \\ B_{15} = \langle b_{127} \parallel \dots \parallel b_{121} \parallel b_{120} \rangle; \\ X = 0x123456789FFFEEDDCCBBA99887766554433221100 &\Rightarrow \\ W_0 = 0x7766554433221100, \quad W_1 = 0xFFEEDDCCBBA9988, \\ B_0 = 0x00, \quad B_1 = 0x11, \quad B_{15} = 0xFF. \end{aligned}$$

Дальше будет использоваться следующая нотация:

- символы операций  $+$ ,  $-$ ,  $\lll$ ,  $\ggg$ , заключённые в квадратные скобки (например  $[+]$ ), будем использовать для обозначения операций выполняемых над словами (64 бита);
- верхний индекс (например  $\text{Fun}^n$ ) – обозначает количество повторений ( $n$ ) некоторого преобразования ( $\text{Fun}$ ) для векторных аргументов (длиной  $n$  элементов);
- нижний индекс (например  $X_i$ ) – обозначает номер (позицию) элемента в некоторой последовательности (векторе);
- верхний индекс в круглых скобках (например  $X^{(i)}$ ) – обозначает номера итерации (или цикла);
- нижний индекс в угловых скобках (например  $X_{\langle 64n \rangle}$ ) – обозначает разрядность вектора ( $X$ );
- символы с 1–3 штрихами (например  $X''$ ) – обозначают промежуточные результаты.

## 1.2 Основные определения

*Блоком* будем называть битовый кортеж длина которого соответствует разрядности входа шифратора, т.е.  $128n$  бит.

*Полублоком* будем называть битовый кортеж длиной  $64n$  битов. Блок ( $T$ ) состоит из двух полублоков: *левого*  $L$  («старшего») и *правого*  $R$  («младшего»), т.е.  $T = \langle L \parallel R \rangle$ .

*S-блоком* будем называть группу совместно вычисляемых нелинейных булевых функций от общего ограниченного множества аргументов (8 бит), реализуемых, как правило, табличным способом.

*Слоем* будем называть последовательность идентичных преобразований, выполняемых «параллельно», т.е. независимо для каждого из элементов вектора-аргумента.

*Числом ветвей активизации* линейного преобразования будем называть минимальное суммарное количество активных S-блоков на входе и выходе этого преобразования, при условии фактической активизации входа. Для байтовых S-блоков – это минимальное количество активных байтов до и после преобразования. В этом случае, число ветвей активизации может быть определено, как минимальное суммарное количество ненулевых байтов на входе и выходе этого преобразования, при отображении отличного от нуля входного значения.

*Циклической матрицей* будем называть такую квадратную матрицу все строки (и столбцы) которой могут быть получены циклическим сдвигом одной строки (или столбца). Такая матрица может быть однозначно определена только первым столбцом (или строкой).

Полином с коэффициентами (при формальных  $x$ ), соответствующими элементам первого столбца, будем называть *полиномом порождающим* циклическую матрицу.

*Циклической матрицей с максимальным числом ветвей* или просто *СМВN-матрицей* (*СМВN* – *Cyclic Maximum Branch Number*), будем называть циклическую матрицу размерностью  $t \times t$  (с элементами из поля  $GF(2^m)$ ), для которой операция умножения образует отображение векторов с максимальным числом «ветвей активизации». Одна ветвь соответствует одному элементу во входном либо выходном векторе. Максимальное достижимое число ветвей активизации для невырожденной матрицы составляет  $(t+1)$ . Полином степени  $(t-1)$ , порождающий такую матрицу будем называть *МВN-полиномом*. Отметим, что свойства такого преобразования, по сути своей, подобно *MDS-преобразованию*, т.е. умножению на *циклический разделимый код максимального расстояния* (*циклический МДР- или MDS-код*). Однако, учитывая, что данное преобразование является биективным, и мы не вносим избыточность в результат, эти два понятия (*МВN-* и *MDS-* преобразования) мы будем разделять. Процедуру умножения вектора на СМВN-матрицу также будем называть *СМВN-кодированием*, а порождающий полином *СМВN-кодом*.

Под одним *циклом* шифрования будем понимать две итерации цепи Фейстеля. Выбор такой нотации обусловлен двумя причинами:

- Управляемое ключом изменение каждого бита блока данных возможно после применения не менее чем 2 итераций цепи Фейстеля;
- сохранение «совместимости» с системой обозначений, использовавшейся в предыдущих публикациях и ряде зарубежных публикаций.

*Исходным (или пользовательским) ключом* будем называть битовый вектор, подаваемый на ключевой вход шифратора.

*Процедурой «разворачивания ключа»* будем называть преобразование, выполняемое до применения процедуры шифрования и предназначенное для формирования, на основе исходного ключа, ключевого материала, используемого процедурой шифрования. По своей длине, рабочий ключ может превосходить исходный до нескольких десятков раз.

*Рабочим ключом* будем называть ключевую последовательность, непосредственно используемую процедурой шифрования и полученную в результате работы процедуры разворачивания ключа».

*Подключом (рабочего ключа) или рабочим подключом* будем называть элемент рабочего ключа используемый на одной итерации или в отдельном преобразовании (например, начальном или конечном). Рабочий подключ, используемый на одной итерации, также, будем называть *ключом итерации*.

## 2. ЭЛЕМЕНТАРНЫЕ ПРЕОБРАЗОВАНИЯ АЛГОРИТМА «ЛАБИРИНТ»

Все элементарные преобразования в алгоритме «Лабиринт» имеют векторную структуру, т.е. блок либо полублок данных следует интерпретировать как последовательность слов, которые, в свою очередь, могут рассматриваться как последовательность из 8 байтов.

### 2.1 Векторные операции над словами

Ниже приведены обозначения векторных элементарных операций, выполняемых над словами, блоками либо полублоками, т.е. первый либо оба аргумента имеют вид последовательности из 1,  $2n$  либо  $n$  слов соответственно. Результат этих операций имеет ту же размерность, что и первый аргумент.

- $x [+ ]^t y$  – сложение в кольце  $\mathbf{Z}_{2^{64}}$  (по модулю  $2^{64}$ ) одноимённых слов, составляющих вектора  $x$  и  $y$  (длиной по  $t$  слов каждый);  
 $x [- ]^t y$  – вычитание в кольце  $\mathbf{Z}_{2^{64}}$  (по модулю  $2^{64}$ ) слов, составляющих вектор  $y$  из одноимённых слов, составляющих вектор  $x$  (длина векторов равна  $t$  слов);  
 $x [ \ll \ll ]^t l$  – циклический сдвиг каждого из  $t$  слов, составляющих вектор  $x$ , на  $l$  бит влево ( $0 \leq l \leq 63$ );  
 $x [ \gg \gg ]^t l$  – циклический сдвиг каждого из  $t$  слов, составляющих вектор  $x$ , на  $l$  бит вправо ( $0 \leq l \leq 63$ ).

Для обозначения векторных (биективных) преобразований общего вида над словами будем использовать аналогичную нотацию:

$$Y = \text{Fun}^t(X, K).$$

Подобная запись означает, что входной ( $X$ ) и выходной ( $Y$ ) вектора данных состоят из  $t$  слов, а вектор управляющего сигнала  $K$  (для управляемых отображений) состоит из  $t$  последовательных кортежей одинаковой длины, и каждое слово результата  $Y$  получается в результате независимого применения преобразования  $\text{Fun}$  к соответствующему слову аргумента  $X$ , с использованием соответствующего управляющего кортежа ключа  $K$ .

## 2.2 Элементарные операции над байтами

Кроме перечисленных выше бинарных операций, к элементарным операциям также будем относить унарную операцию над байтами – *нелинейную подстановку* 8-битных векторов (**S-блок**). Применение этой операции к отдельному байту будем обозначать:  $y = S(x)$ ,  $x, y \in \mathbf{B}$ . Для обозначения преобразования обратного к данному, будем использовать следующую запись:  $x = S^{-1}(y)$ .

Для обозначения векторных преобразований над байтовыми строками, будем использовать обозначения аналогичные введенным выше, например:

$$Y = [S]^{8n}(X), \quad X, Y \in \mathbf{H}.$$

## 3. СТРУКТУРА АЛГОРИТМА «ЛАБИРИНТ»

### 3.1 Преобразования зашифрования и расшифрования

Алгоритм «Лабиринт» является «инволютивным» шифром, т.е. зашифрование и расшифрование выполняются на основе одной общей процедуры, отличие заключается только в противоположной последовательности применения ключей итераций  $K^{(i)}$ , а также ключей начального и конечного преобразований (подключи  $K^{1\Gamma}$  и  $K^{FT}$  соответственно).

Преобразования зашифрование / расшифрования состоит из двух фаз:

1. Процедура разворачивания ключа  $K\text{Shed}$ , на основе исходного ключа  $UK$  выполняет формирование подключей рабочего ключа  $WK$ . Порядок применения подключей определяется необходимым «направлением» преобразования (*mode*) – зашифрование (e) или расшифрование (d).
2. Процедура шифрования  $EF$  выполняет преобразование входного блока данных ( $P$  либо  $C$ ), на рабочем ключе  $WK$ .

$$\begin{aligned}
WK &= \text{KShed}(UK, mode = e \setminus d) \\
C &= E_{UK}(P) = EF_{WK_E}(P), \\
P &= D_{UK}(C) = EF_{WK_D}(C), \quad P, C \in \mathbf{T} \\
WK_E &= (K^{IT}, K^{(0)}, K^{(1)}, \dots, K^{(2r-1)}, K^{FT}) \\
WK_D &= (K^{FT}, K^{(2r-1)}, K^{(2r-2)}, \dots, K^{(0)}, K^{IT})
\end{aligned}$$

где  $\text{KShed}$  – процедура разворачивания ключа;  
 $mode$  – режим преобразования: зашифрование (e) / расшифрование (d);  
 $UK$  – исходный (пользовательский) ключ;  
 $WK_E$  – рабочий ключ зашифрования (прямая последовательность подключей);  
 $WK_D$  – рабочий ключ расшифрования (обратная последовательность подключей);  
 $K^{(i)}$  – ключ для  $i$ -й итерации;  
 $E_{UK}$  – преобразование зашифрования на исходном ключе  $UK$ ;  
 $D_{UK}$  – преобразование расшифрования на исходном ключе  $UK$ ;  
 $EF_{WK}$  – процедура шифрования на рабочем ключе  $WK$ ;  
 $P$  – блок «открытого» текста;  
 $C$  – блок криптограммы.

### 3.2 Процедура шифрования EF

Алгоритм «Лабиринт» является итеративным шифром, т.е. основу его процедуры шифрования составляет цикловое преобразование, которое повторяется заданное число раз (обозначим число циклов шифрования символом  $r$ ). Для алгоритма «Лабиринт» каждый цикл состоит из двух абсолютно идентичных итераций, однако, учитывая, что для обновления обоих полублоков, составляющих один блок, требуется, как минимум, две итерации, в данной спецификации понятие цикла отделено от понятия итерации. Кроме повторяемого циклового преобразования процедура шифрования включает начальное (IT) и конечное (FT) преобразования. Свойство инволютивности шифра достигается за счёт применения классической конструкции полублоковой цепи Фейстеля (рис. 1).

Алгоритм «Лабиринт», независимо от длины блока и ключа использует фиксированное количество итераций шифрования – 16 (или, иначе говоря, 8 циклов, т.е.  $r = 8$ ).

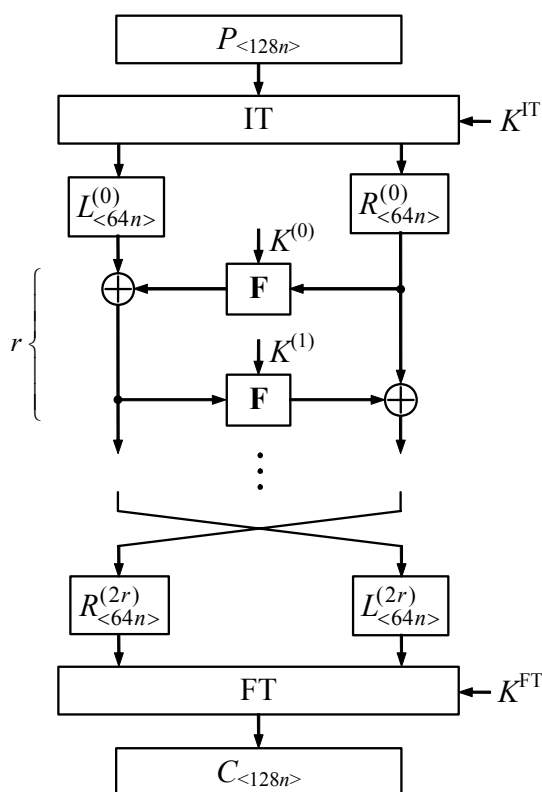


Рис. 1 – Структура процедуры шифрования EF

Процедура шифрования EF состоит из трёх шагов:

1. Исходный блок данных  $P$  (длиной  $128n$  бит) обрабатывается начальным (IT) преобразованием на ключе  $K^{IT}$ .
2. Результат 1-го шага разбивается на два *полублока* длиной по  $64n$  бита: левый  $L$  («старший») и правый  $R$  («младший»). Полученная пара полублоков преобразуется на 16 итерациях (8 циклах) цепи Фейстеля. Все итерации полностью идентичны и построены на базе общего нелинейного преобразования – F-функции, управляемой ключом итерации  $K^{(i)}$ .
3. Два полублока  $L$  и  $R$ , полученные в результате 8-циклового итеративного преобразования, меняются местами и обрабатываются конечным (FT) преобразованием на ключе  $K^{FT}$ . Полученный после FT преобразования двоичный вектор  $C$  (длиной  $128n$  бит) является результирующим блоком («криптограммой»).

Аналитически, процедура EF может быть представлена следующим образом:

$$C = EF_{WK}(P):$$

$$P, C, K^{IT}, K^{FT} \in \mathbf{T}, \quad L^{(i)}, R^{(i)}, K^{(i)} \in \mathbf{H}$$

$$WK = (K^{IT}, K^{(0)}, \dots, K^{(2r-1)}, K^{FT})$$

$$\langle L^{(0)} \parallel R^{(0)} \rangle = IT_{K^{IT}}(P)$$

$$i = \overline{0, 2r-1}: \begin{cases} R^{(i+1)} = L^{(i)} \oplus F_{K^{(i)}}(R^{(i)}) \\ L^{(i+1)} = R^{(i)} \end{cases}$$

$$C = FT_{K^{FT}}(\langle R^{(2r)} \parallel L^{(2r)} \rangle).$$

В приведенных выше соотношениях символом  $i$  обозначается номер итерации, а символом  $r$  – количество циклов шифрования (один цикл – две итерации).

На каждой итерации F-функция использует  $64n$ -битный подключ  $K^{(i)}$ . Длина ключей начального ( $K^{IT}$ ) и конечного ( $K^{FT}$ ) преобразований составляет по  $128n$  бит каждый.

### 3.3 Базовая F-функция

Конструкция F-функции алгоритма «Лабиринт» построена в соответствии с принципами позволяющими обеспечить свойства рассеивания и размножения активизации [11].

F-функция состоит из четырех элементарных преобразований:

- сложение слов полублока со словами ключа итерации по модулю  $2^{64}$ ;
- фиксированная байтовая перестановка P;
- фиксированная нелинейная подстановка байтов составляющих полублок;
- фиксированное преобразование линейного смешивания.

Подробно эти преобразования будут рассмотрены ниже.

F-функция использует один подключ длиной  $64n$  бит и имеет следующее аналитическое представление:

$$Y = F_{K^{(i)}}(X) = SL^n(P(X [+]^n K^{(i)})), \quad X, Y, K^{(i)} \in \mathbf{H}.$$

В общем виде F-функция может быть представлена схемой рис. 2.

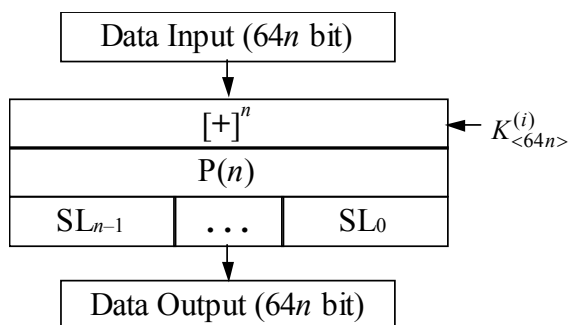


Рис. 2 – F-функция общего вида ( $n = 1, 2$  или  $4$ )

Частный случай F-функции, когда  $n = 1$  (т.е. длина полублока составляет 64 бита), представлен на рис. 3. В этом случае ( $n = 1$ ) преобразование  $P_{\text{byte}}$ , как будет показано ниже, является тождественным, поэтому на схеме (рис. 3) оно отсутствует.

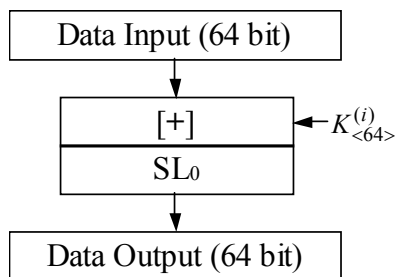


Рис. 3 – 64-битная F-функция ( $n = 1$ )

При выполнении векторных операций (т.е. сложение с итеративным ключом  $K^{(i)}$  и преобразование  $SL^n$ ), полублок данных рассматривается как последовательность из  $n$  слов

(по 64 бита), к каждому из которых независимо применяется соответствующее преобразование:

$$X, Y, K^{(i)} \in \mathbf{H}, \quad W_j, W'_j, K_j^{(i)} \in \mathbf{W}, \quad j = \overline{0, n-1}$$

$$X = \langle W_{n-1} \parallel \dots \parallel W_1 \parallel W_0 \rangle$$

$$Y = \langle W'_{n-1} \parallel \dots \parallel W'_1 \parallel W'_0 \rangle$$

$$K^{(i)} = \langle K_{n-1}^{(i)} \parallel \dots \parallel K_1^{(i)} \parallel K_0^{(i)} \rangle$$

$$Y = X [+ ]^n K^{(i)} \Rightarrow W'_j = W_j [+ ] K_j^{(i)}$$

$$Y = \text{SL}^n(X) \Rightarrow W'_j = \text{SL}_j(W_j).$$

### 3.4 Инволютивная операция линейного смешивания IMix

В составе начального (IT) и конечного (FT) преобразований используется операция *инволютивного линейного смешивания* (IMix). Данное преобразование предназначено для предотвращения возможности отдельной активизации полублоков с помощью характеристик с малым количеством активных байтов. Особенностью этого преобразования является свойство инволютивности, т.е. двухкратное применение преобразования IMix приводит к тождественному отображению аргумента.

Для описания данного преобразования воспользуемся следующими обозначениями:

$$\langle L' \parallel R' \rangle = \text{IMix}(\langle L \parallel R \rangle): \quad \mathbf{T} \rightarrow \mathbf{T}$$

$$L, R \in \mathbf{H}, \quad L_j, R_j, \Sigma_j \in \mathbf{W}$$

$$L = \langle L_{n-1} \parallel \dots \parallel L_1 \parallel L_0 \rangle, \quad R = \langle R_{n-1} \parallel \dots \parallel R_1 \parallel R_0 \rangle$$

где  $L$  и  $R$  – соответственно, левый и правый полублоки.

С учётом введенных обозначений преобразование IMix может быть записано следующим образом:

for  $j = \overline{0, n-1}$ :

$$L'_j = L_j \oplus \Sigma_{(j-1) \bmod n}, \quad R'_j = R_j \oplus \Sigma_{(j-1) \bmod n}$$

$$\Sigma_j = \begin{cases} (L_j \oplus R_j) [ \ll \ll ] 28, & j = 0 \\ L_j \oplus R_j, & j > 0 \end{cases}$$

### 3.5 Начальное IT и конечное FT преобразования

Начальное IT (рис. 4) и конечное FT (рис. 5) преобразования включают сложение блока данных с рабочим подключом (длиной  $128n$  бит) и фиксированное преобразование блока данных. Фиксированные (т.е. не зависящие от ключа) составляющие преобразований IT и FT являются взаимно обратными и включают два «слоя»:

- нелинейная подстановка («S–слой»);
- линейное смешивание («L–слой»).

«L–слой» обоих преобразований удобно рассматривать как последовательность из двух операций:

- циклический сдвиг слов левого и правого полублоков на 2 разряда влево и вправо;
- линейное смешивания полублоков IMix.

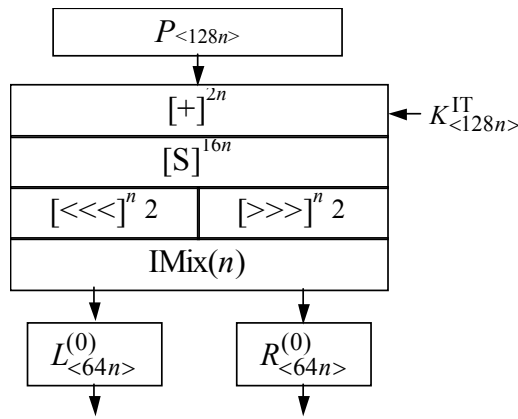


Рис. 4 – Начальное IT – преобразование

Начальное преобразование (IT) может быть представлено следующей последовательностью операций:

- $$\langle L^{(0)} \parallel R^{(0)} \rangle = \text{IT}_{K^{\text{IT}}}(P): L, R \in \mathbf{H}, P, T, K^{\text{IT}} \in \mathbf{T}$$
- 1)  $T = P [+ ]^{2n} K^{\text{IT}}$
  - 2)  $\langle L' \parallel R' \rangle = [S]^{16n}(T)$
  - 3)  $L'' = L' [ \ll \ll ]^n 2, R'' = R' [ \gg \gg ]^n 2$
  - 4)  $\langle L^{(0)} \parallel R^{(0)} \rangle = \text{IMix}(\langle L'' \parallel R'' \rangle)$

где  $P$  – исходный блок текста на входе шифратора («открытый текст»);  
 $L^{(0)}, R^{(0)}$  – соответственно, левый и правый полублоки на выходе IT-преобразования.

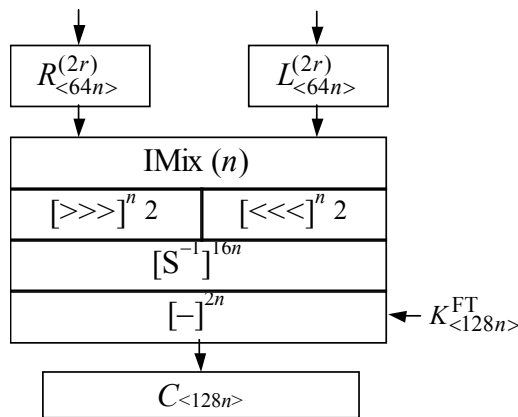


Рис. 5 – Конечное FT-преобразование

Конечное преобразование (FT) может быть представлено следующей последовательностью операций:

$$C = \text{FT}_{K^{\text{FT}}}(R^{(2r)} \parallel L^{(2r)}): L, R \in \mathbf{H}, C, T, K^{\text{FT}} \in \mathbf{T}$$

$$1) \langle L' \parallel R' \rangle = \text{IMix}(\langle R^{(2r)} \parallel L^{(2r)} \rangle)$$

$$2) L'' = L' [\gg]^{n/2}, R'' = R' [\ll]^{n/2}$$

$$3) T = [S^{-1}]^{16n} (\langle L'' \parallel R'' \rangle)$$

$$4) C = T [-]^{2n} K^{\text{FT}}$$

где  $C$  – результирующий блок текста на выходе шифратора («криптограмма»);  
 $L^{(2r)}, R^{(2r)}$  – соответственно левый и правый полублоки после последней итерации цепи Фейстеля.

Следует обратить внимание, что после выполнения последней итерации цепи Фейстеля, левый и правый полублоки должны поменяться местами, и только после этого результирующий блок поступает на вход ФТ-преобразования. Однако, в ряде случаев, с целью повышения эффективности реализации, указанная перестановка может быть совмещена с ФТ-преобразованием.

### 3.6 Фиксированная перестановка байтов $P(n)$

Вид байтовой перестановки  $P(n)$  определяется количеством 64-битных слов в полублоке (т.е. параметром  $n$ ). Байтовая перестановка выполняется в соответствии со следующим соотношением:

$$P(n): B'_i = B_{(9i \bmod 8n)}, B_i \in \mathbf{B}, i = \overline{0, 8n-1}$$

где  $B_j$  – байт-источник (где  $j$  – позиция байта в исходном полублоке);  
 $B'_i$  – байт-получатель (где  $i$  – позиция байта в результирующем полублоке).

На рис. 6 – 8 показаны схемы перестановки байтов для всех допустимых значений  $n$  (т.е. 1, 2 и 4). На приведенных схемах каждая ячейка соответствует одному байту, «большое» число в ячейке – исходная позиция (64-битного) слова в полублоке, а нижний индекс – исходная позиция байта внутри соответствующего (64-битного) слова.

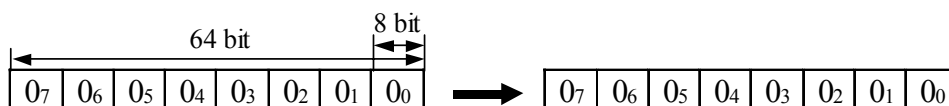


Рис. 6 – Перестановка  $P$  для случая  $n = 1$

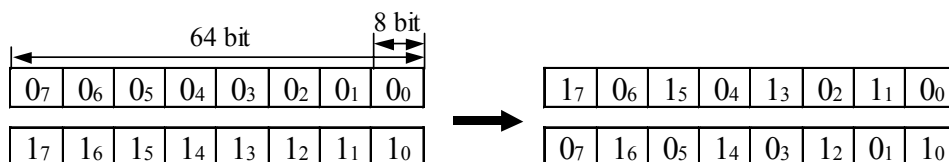


Рис. 7 – Перестановка  $P$  для случая  $n = 2$

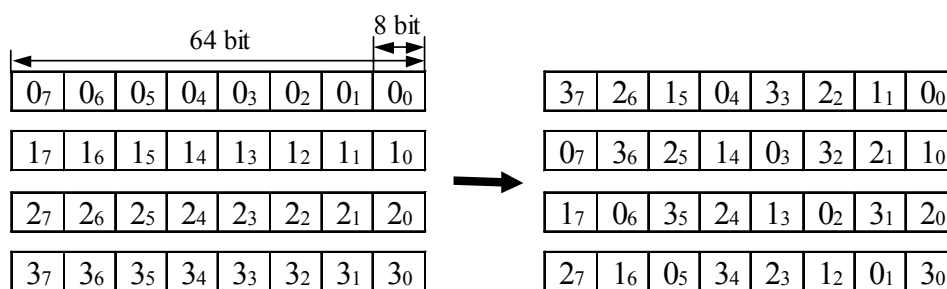


Рис. 8 – Перестановка P для случая  $n = 4$

### 3.7 Узел замены (S-box)

S-блок (или узел замены) определяет фиксированное биективное преобразование отдельного байта. S-блок реализуется в виде таблицы подстановки размером 256 байт.

Ниже (табл. 1) приведено табличное описание используемого в данной спецификации узла подстановки (S-блока). Первый столбец и первая строка (выделенные жирным шрифтом) используются для указания индексов элементов подстановки. Индексы строк и столбцов таблицы, а также значения элементов таблицы указаны в шестнадцатеричной системе счисления (префикс 0x не используется для сокращения записи). «Логическое ИЛИ» индексов строки и столбца соответствует входному значению, результат подстановки которого находится на пересечении соответствующих строки и столбца. Например, результат подстановки шестнадцатеричного числа 0x53 будет равен 0x54.

Приведенное значение узла замены не является обязательным для алгоритма «Лабиринт» и приведено в данной спецификации в качестве примера. S-блок построен в соответствии с критериями отбора и конструкцией изложенной в [12]. Алгебраическая структура подстановки аффинно эквивалентна конструкции Нибберг-Динга, детальное описание алгебраической конструкции S-блока приведено в разделе 4.2. Значение узла замены может быть заменено другим, при условии, что он будет удовлетворять требованиям приведенным разделе 4.2. Отметим, что значение S-блока должно выбираться совместно с параметрами MBN-преобразования.

Таблица 1.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	2B	F8	D2	D5	35	A2	3F	C6	BB	C2	A3	F1	9F	6F	1D	E1
10	60	7E	E0	7F	2D	AC	E3	0D	BC	9D	C0	FE	3B	D1	1B	C3
20	80	63	C9	46	79	E7	89	E9	1C	AB	17	97	5A	20	30	EC
30	71	B8	B2	02	06	F2	E5	FD	28	D3	3E	3C	D0	BA	CE	29
40	10	B9	50	08	A1	A8	7D	40	01	15	7C	78	33	69	EB	0E
50	6E	7B	77	54	92	58	95	C1	98	EE	1F	9B	96	51	26	61
60	2A	CC	B4	0C	DF	A7	27	9E	32	37	B3	FC	0A	AD	2C	19
70	B1	11	C8	AA	90	18	45	36	75	94	8E	CB	16	BD	FB	48
80	E6	F4	73	BE	07	6A	42	F7	41	4C	05	EA	DC	76	D8	6C
90	74	87	A5	8B	1A	9C	4E	6B	0B	24	91	34	4A	2E	F3	E8
A0	DA	64	7A	8F	EF	D4	93	AF	66	13	CF	82	59	D7	31	4F
B0	C4	65	03	BF	D9	68	C5	E2	84	A6	23	99	C7	B0	5B	62
C0	A0	12	83	ED	8C	00	57	DD	22	FF	9A	A4	F5	F9	3D	6D
D0	FA	5C	49	39	43	5E	86	0F	B7	67	52	CA	14	38	DB	25
E0	3A	70	E4	1E	04	55	72	DE	56	47	CD	B6	8D	85	88	D6
F0	A9	F0	5F	AE	09	8A	81	53	21	B5	F6	4B	4D	5D	44	2F

Алгоритм «Лабиринт» построен на базе одного общего S-блока, однако в составе конечного FT-преобразования используется обратный (или инверсный), к приведенному в

таблице 1, S-блок (обозначается  $S^{-1}$ ), который определяет преобразование обратное к «прямому» S-блоку, т.е.:

$$S: \mathbf{B} \rightarrow \mathbf{B}$$

$$\forall x: y = S(x) \Rightarrow x = S^{-1}(y), \quad x, y \in \mathbf{B}.$$

Отметим, что, обычно, преобразования этого класса реализуются табличным способом. При этом, особенно при программной реализации, преобразование на узле замены может быть эффективно совмещено с реализацией последующего фиксированного линейного преобразования табличным способом, т.е. оба преобразования могут быть выполнены с помощью одной общей таблицы.

### 3.8 MBN-преобразование

Это преобразование определяет биективное линейное отображение слов, составляющих полублок. MBN-преобразование может быть представлено в виде матричного умножения – квадратная матрица размерностью  $8 \times 8$  байт, образованная циклическим MBN-кодом, умножается справа на вектор-столбец длиной 8 байт (соответствующий слову-аргументу). Полученный вектор-столбец (длиной 8 байт) соответствует слову-результату. Элементы матрицы (байты), и элементы векторов аргумента / результата, интерпретируются как элементы поля  $GF(2^8)$ , образованного выбранным неприводимым (над полем  $GF(2)$ ) полиномом 8-й степени  $f_{MBN}(x)$ .

Учитывая циклическую структуру используемого MBN-кода данное преобразование данное преобразование может быть записано следующим образом:

$$CMBN: \mathbf{W} \rightarrow \mathbf{W}$$

$$Y = CMBN(X), \quad X, Y \in \mathbf{W}, \quad B_j \in \mathbf{B}$$

$$X = \langle B_7 \parallel \dots \parallel B_1 \parallel B_0 \rangle$$

$$Y = \bigoplus_{j=0}^7 LCM(B_j) \lll (8 \times j), \quad LCM: \mathbf{B} \rightarrow \mathbf{W},$$

где LCM – операция умножения одного байта на циклический MBN-код.

LCM-отображение определяется операцией умножения элемента поля  $GF(2^8)$ , соответствующего байту-аргументу, на фиксированный полином 7 степени с коэффициентами из  $GF(2^8)$ . Используемый полином формирует циклический MBN-код, а также обладает рядом дополнительных свойств, указанных разделе 4.3. С точки зрения реализации алгоритма, принципиальным является свойство цикличности порождаемого кода, т.к. на нём базируется структура MBN-преобразования.

Указанное выше LCM-отображение имеет следующую аналитическую запись:

$$LCM: \mathbf{B} \rightarrow \mathbf{W}$$

$$Y = LCM(X), \quad X \in \mathbf{B}, \quad Y \in \mathbf{W}$$

$$g(x) = g_7 x^7 + \dots + g_1 x + g_0$$

$$Y = \langle g_7 X \parallel \dots \parallel g_1 X \parallel g_0 X \rangle, \quad X, g_i, (g_i X) \in F_{2^8}, \quad g(x), Y \in F_{2^8}$$

где  $g(x)$  – полином, порождающий циклический MBN-код;

$X$  – входной байт;

$Y$  – выходное слово.

В данной спецификации используется следующий полином, порождающий CMBN-код:

$$g(x) = 0C x^7 + 0F x^6 + 1F x^5 + 0A x^4 + 0F x^3 + 03 x^2 + 01x + 01,$$

с коэффициентами из поля  $GF(2^8)$ , которое задано следующим неприводимым (над полем  $GF(2)$ ) полиномом 8-й степени (в шестнадцатеричном представлении  $0x12D$ ):

$$f_{MBN}(x) = x^8 + x^5 + x^3 + x^2 + 1.$$

Отметим, что приведенное значение полинома  $g(x)$ , порождающего циклический MBN-код (а также полинома  $f_{MBN}(x)$ ), не является обязательным для алгоритма «Лабиринт» и может быть заменено другим, удовлетворяющим требованиям, указанным разделе 4.3. Возможность произвольного выбора S-блока и полинома порождающего MBN-код, а также «прозрачность» критериев их отбора позволяет решить проблему возможных «лазеек» (или «потайных дверей»), заложенных авторами алгоритма.

### 3.9 Преобразование SL

SL-преобразование определяет фиксированное биективное отображение слов. Это преобразование представляет собой объединение нелинейного преобразования байтов (S-блок), составляющих отдельное 64-битное слово, с последующим линейным «смешиванием» этих, уже преобразованных на S-блоке, байтов. Вид SL-преобразования зависит от позиции ( $t$ ) слова ( $X$ ), внутри полублока, к которому оно применяется.

Преобразование  $SL_t$  имеет следующее аналитическое описание:

$$SL_t : \mathbf{W} \rightarrow \mathbf{W}$$

$$Y = SL_t(X), \quad t = \overline{0, n-1}$$

$$X = \langle B_7 \parallel \dots \parallel B_1 \parallel B_0 \rangle, \quad B_j \in \mathbf{B}, \quad X, Y, xc_t \in \mathbf{W}$$

$$1) X' = \langle S(B_7) \parallel \dots \parallel S(B_1) \parallel S(B_0) \rangle$$

$$2) Y' = MBN(X')$$

$$3) Y = Y' \oplus xc_t$$

где  $X$  – входное слово (64 бита);

$Y$  – выходное слово (64 бита);

$t$  – номер (позиция внутри полублока) слова  $X$ , к которому будет применено данное преобразование;

$xc_t$  – значение XOR-константы, уникальное для каждого значения  $t$ , определяется из следующей таблицы.

Таблица 2.

$t$	0	1	2	3
$xc_t$	0x0	0x1111111111111111	0x2222222222222222	0x8888888888888888

### 3.10 Процедура «разворачивания ключа»

Из приведенной выше структуры шифрующего преобразования алгоритма «Лабиринт» следует, что процедура шифрования требует рабочий ключ длиной  $16 \times 64n + 2 \times 128n$  бит (16 ключей итераций по  $64n$  бит, а также ключи начального и конечного преобразований по  $128n$  бит). Для формирования рабочего ключа указанной длины на основе исходного (пользовательского) ключа существенно меньшего объема используется процедура разворачивания ключа. Процедура разворачивания ключа алгоритма «Лабиринт» поддерживает переменную длину пользовательского ключа, которая может находиться в

диапазоне от 2 до 4 полублоков включительно, т.е. от  $128n$  бит до  $256n$  бит, с шагом  $64n$  бит – один полублок.

Для сокращения требований к необходимому объему оперативной памяти, алгоритм использует принцип повторного использования ключевого материала на различных итерациях шифрования. Для хранения «развёрнутого» рабочего ключа необходим буфер объёмом 8 полублоков, т.е.  $8 \times 64n$  бит.

Процедура разворачивания ключа состоит из двух этапов:

- инициализация буфера рабочего ключа на основе исходного ключа пользователя;
- выборка подключей из буфера рабочего ключа.

Процедура инициализации заключается в загрузке исходного ключа в буфер рабочего ключа и дополнение этого ключа до полного заполнения буфера ключевым материалом, сформированным из исходного ключа. Для формирования «дополнительного» ключевого материала используется базовая F-функция шифратора, на базе которой построен простой криптографический генератор псевдослучайных последовательностей – исходный ключевой материал циклически зашифровывается в режиме «связки шифрблоков» (СВС-режим [13]).

Ниже приведено формальное описание процедуры разворачивания ключа (учитывая, что схематическое представление данной процедуры является не достаточно иллюстративным, ниже приводится только аналитическое описание).

Для описания процедуры разворачивания ключа воспользуемся следующими обозначениями:

$$\begin{aligned} \text{INPUT: } & l_{\text{UK}}, uk_1, uk_2, \dots, uk_{l_{\text{UK}}} \quad (2 \leq l_{\text{UK}} \leq 4) \\ \text{OUTPUT: } & WK = (K^{\text{IT}}, K^{(0)}, K^{(1)}, \dots, K^{(2r-1)}, K^{\text{FT}}) \\ & uk_j, k_j, k_{\text{KS}}, K^{(i)} \in \mathbf{H}, \quad K^{\text{IT}}, K^{\text{FT}} \in \mathbf{T}. \end{aligned}$$

где  $l_{\text{UK}}$  – длина исходного ключа, введенного пользователем;

$uk_j$  –  $j$ -й полублок исходного ключа;

$k_j$  –  $j$ -я ячейка буфера рабочего ключа (длиной один полублок);

$k_{\text{KS}}$  – ключ, используемый F-функцией процедуры разворачивания ключа (длиной один полублок,);

$WK$  – «развёрнутый» рабочий ключ.

Процедура разворачивания ключа начинается с загрузки пользовательского ключа в шифратор, а именно в буфер рабочего ключа, одновременно выполняется инициализация ключа  $k_{\text{KS}}$ :

$$k_{\text{KS}} = 0$$

$$\text{for } j = \overline{0, l_{\text{UK}} - 1}: \quad k_j = uk_j, \quad k_{\text{KS}} = k_{\text{KS}} \oplus uk_j$$

$$k_{\text{KS}} = k_{\text{KS}} [\lll]^{29}.$$

Как видно из приведенного описания, ключ  $k_{\text{KS}}$  вычисляется как сумма по модулю 2 всех полублоков пользовательского ключа. После накопления указанной суммы, выполняется циклический сдвиг на 29 разрядов каждого из  $n$  слов составляющих полублок  $k_{\text{KS}}$ . Дальнейшее заполнение буфера рабочего ключа выполняется согласно следующему рекуррентному соотношению:

$$\text{for } j = \overline{l_{\text{UK}}, 7}: \quad k_{\text{KS}} = \delta(k_{\text{KS}}), \quad k_j = F_{k_{\text{KS}}}(k_{j-1}) \oplus k_{j-l_{\text{UK}}}.$$

где  $\delta$  – функция «обновления» ключа  $k_{\text{KS}}$ , которая применяется перед каждым использованием F-функции в процедуре разворачивания ключа.

Функция  $\delta$  имеет следующий вид:

$$\delta: \mathbf{H} \rightarrow \mathbf{H}$$

$$Y = \delta(X), \quad X = \langle W_{n-1} \parallel \dots \parallel W_0 \rangle, \quad Y = \langle W'_{n-1} \parallel \dots \parallel W'_0 \rangle, \quad X, Y \in \mathbf{H}, \quad W_j \in \mathbf{W}$$

$$W'_j = (W_j + 0x1084210842108421) \oplus 0x4444444444444444, \quad j = \overline{0, n-1}.$$

Полученное заполнение буфера рабочего ключа  $\{k_i\}$  используется процедурой зашифрования следующим образом:

$$K^{IT} = \langle k_7 \parallel k_4 \rangle$$

$$\text{for } i = \overline{0, 7}: K^{(i)} = k_i$$

$$\text{for } i = \overline{8, 15}: K^{(i)} = k_{(3i \bmod 8)}$$

$$K^{FT} = \langle k_6 \parallel k_3 \rangle.$$

Для выполнения процедуры расшифрования все подключи «зашифрования» располагаются в обратном порядке, что аналитически может быть записано следующим образом:

$$K^{IT} = \langle k_6 \parallel k_3 \rangle$$

$$\text{for } i = \overline{0, 7}: K^{(i)} = k_{(3 \times (7-i) \bmod 8)}$$

$$\text{for } i = \overline{8, 15}: K^{(i)} = k_{7-(i \bmod 8)}$$

$$K^{FT} = \langle k_7 \parallel k_4 \rangle.$$

Эквивалентно последовательность выбора подключей итераций  $K^{(i)}$ , для преобразования зашифрования, может быть задана в табличном виде (табл. 3). При выполнении расшифрования эти подключи выбираются в обратном порядке:

Таблица 3.

$i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$k_j$	$k_0$	$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	$k_6$	$k_7$	$k_0$	$k_3$	$k_6$	$k_1$	$k_4$	$k_7$	$k_2$	$k_5$

Как видно из приведенной таблицы, каждый полублок буфера рабочего ключа используется только на двух итерациях цепи Фейстеля.

Одним из достоинств данной схемы разворачивания ключа, и всей конструкции шифра в целом, является применимость одного и того же материала «развёрнутого» ключа, как для преобразования зашифрования, так и расшифрования.

## ЗАКЛЮЧЕНИЕ

Алгоритм «Лабиринт» удовлетворяет требованиям первого (максимального) класса стойкости, в соответствии с условиями проекта NESSIE (длина блока не менее 128 бит, длина ключа не менее 256 бит). Структура алгоритма «Лабиринт» позволяет эффективно его реализовать как на 64-битных, так и на 32-битных микропроцессорах. А учитывая последние достижения в сфере специализированных микроконтроллеров, а именно тенденцию перевода смарт-карт на 32-битные архитектуры и / или оснащение их РКИ-криптоускорителем, данный алгоритм может быть эффективно реализован, в том числе, и на смарт-картах. С другой стороны, выбранная конструкция алгоритма, обеспечивает возможность эффективной аппаратной реализации, при этом возможно существенное снижение затрат

энергонезависимой памяти, за счёт аппаратной реализации операции умножения в поле  $GF(2^8)$ , и снижение вычислительной сложности, за счёт «устранения» операций фиксированного сдвига. Кроме того, алгоритм построен на базе классической цепи Фейстеля, что обеспечивает инволютивность шифрующего преобразования и возможность использования общего аппаратного решения для процедур шифрования и расшифрования.

Существенным достоинством предлагаемой конструкции шифра является возможность применения единожды «развёрнутого» рабочего ключа для выполнения обоих преобразований шифрования и расшифрования, без дополнительных модификаций ключевого материала. Это оказывается весьма существенно когда на одном ключе необходимо выполнять оба преобразования и позволяет до двух раз сократить затраты оперативной памяти и вычислительные затраты на выполнение процедуры разворачивания ключа.

Конструкция циклового преобразования обеспечивает возможность увеличения длины обрабатываемого блока без необходимости увеличения количества циклов шифрования, а конструкция схемы разворачивания ключа обеспечивает независимость количества циклов шифрования от длины исходного ключа. Эти два свойства алгоритма «Лабиринт» позволяют получить практически постоянную производительность шифра при различных значениях показателей стойкости – длинах блока и ключа.

Возможность произвольного выбора параметров узла замены (S-блока) и линейного MBN-преобразования, в пределах строго определённых ограничений к их криптографическим показателям, позволяет, с одной стороны, устранить потенциальную опасность ввода «закладок» (или «потайных дверей») со стороны автора алгоритма, а с другой – обеспечивает возможность введения дополнительного секретного параметра, если параметры указанных элементов составляют долговременный секретный ключ. Такие долговременные ключи могут формироваться по специальной методике, обеспечивающей контроль, не только указанных в данной спецификации, минимальных критериев «фильтрации», но и ряда дополнительных. Формирование указанных долговременных ключей, для государственных организаций, может выполняться соответствующим уполномоченным органом и поставляться пользователям в установленном порядке.

Конструкция алгоритма «Лабиринт» учитывает известные методы криптоанализа БСШ и позволяет защититься от них. В основу алгоритма были положены хорошо изученные принципы построения шифров (ГОСТ–подобная структура и AES–подобный узел нелинейного усложнения и линейного смешивания), прошедшие апробацию временем, а также был учтён опыт использования сравнительно новых конструкций и принципов построения БСШ [10, 11, 12]. Их совместное применение позволило создать алгоритм, объединивший в себе достоинства каждого из «прародительских» алгоритмов, и при этом избавиться от присущих им недостатков. Поэтому, для успешного криптонападения на полученный алгоритм требуется поиск принципиально новых методов криптоанализа БСШ. С другой стороны, накопленный опыт и математический аппарат исследования стойкости алгоритмов ГОСТ 28147-89 [3] и AES [1] может быть применён для первичной оценки безопасности алгоритма «Лабиринт», что существенно упрощает эту задачу. Конструкция алгоритма построена в соответствии с принципом «"слабой" цикловой функции, повторяемой многократно», который, по мнению автора, продемонстрировал большую эффективность, по сравнению с принципом «"сильной" цикловой функции, повторяемой ограниченное число раз» [14].

Алгоритм «Лабиринт» может использоваться в любом из пяти стандартных режимов применения БСШ, а также в «усиленных» режимах поточного шифрования, предложенных автором [15]. Именно в «усиленных» режимах применения БСШ, сегодня, может быть востребовано использование больших длин блока (256 или даже 512 бит).

## ЛИТЕРАТУРА

1. Federal Information Processing Standards Publication 197 (FIPS PUB 197). Specification for the Advanced Encryption Standard (AES). // Department of Commerce, National Institute of Standards and Technology, Information Technology Laboratory. November 26, 2001.
2. Federal Information Processing Standards Publication 46-3 (FIPS PUB 46-3). Specification for the Data Encryption Standard (DES). // U.S. Department of Commerce / National Institute of Standards and Technology. October 25, 1999.
3. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. М.: Госстандарт СССР.
4. AES discussion forum: <http://aes.nist.gov>
5. NESSIE Call for Cryptographic Primitives, Version 2.2, 8th March 2000: <http://cryptonessie.org/>.
6. Горбенко И.Д., Головашич С.А. Алгоритм блочного симметричного шифрования «Торнадо». Спецификация преобразования // Радиотехника. Всеукр. межвед. научн.-техн. сб. 2003. – Вып. 134. – С. 62–80.
7. Долгов В.И., Головашич С.А., Руженцев В.И. Криптостойкость шифра «Торнадо». // Радиотехника. Всеукр. межвед. научн.-техн. сб. 2003. – Вып. 134. – С. 81–88.
8. Головашич С.А., Лепеха А.И. Статистический анализ БСШ «Торнадо» // Радиотехника. Всеукр. межвед. научн.-техн. сб. 2003. – Вып. 134. – С. 89–96.
9. Головашич С.А. Методы построения высокостойких блочных симметричных шифров и схем их применения. // Диссертация на соискание учёной степени кандидата технических наук. Харьковский национальный университет радиоэлектроники. Харьков. 2001.
10. Головашич С.А. Принцип построения инволютивных шифров // Проблемы бионики. Харьков. 2001. – Вып. 54. – С. 118–125.
11. Головашич С.А. Метод конструирования цикловых функций БСШ // Автоматизированные системы управления и приборы автоматики. Всеукр. межвед. научн.-техн. сб. 2001. – Вып. 117. – С. 155–161.
12. Головашич С.А. Метод построения управляемых S-блоков с предельными показателями нелинейности // Радиотехника. Всеукр. межвед. научн.-техн. сб. 2001. – Вып. 123. – С. 215–221.
13. NIST Special Publication 800-38A 2001 Edition – Recommendation for Block Cipher Modes of Operation. Methods and Techniques. // Computer Security Division Information Technology Laboratory National Institute of Standards and Technology, Gaithersburg, MD 20899-8930. December 2001.
14. «Supporting Document on E2», Nippon Telegraph and Telephone Corporation, June 14, 1998.
15. Головашич С.А. Безопасность режимов блочного шифрования // Радиотехника. Всеукр. межвед. научн.-техн. сб. 2001. – Вып. 119. – С. 135–145.