esse

Назаров Захар

December 2023

Содержание

1. Введение

В данной работе дается описание и некоторые аспекты безопасности следующих поточных шифров: RC4, Snow3G, SEAL, MUGI, семейство шифров A5(A5/1, A5/2 и A5/3) И LILI128. Устройство шифра LILI128 и его особенности рассматриваются более детально, также описаны 3 атаки на него.

2. Термины и определения

Для понимания работы требуется ознакомиться со следующими определениями:

Определение 2.1. Булева функция называется сбалансированной, если количество наборов переменных, на которых она принимает значение 0, равно количеству наборов переменных, на которых она принимает значение 1.

Определение 2.2. Нелинейностью (nonlinearity) булевой функции f степени n является значение минимального расстояния по Хеммингу до линейной функции:

$$N_f = \min_{g \in \mathcal{A}_n} d_H(f, g) \tag{2.1}$$

, где $\mathcal{A}_n$ - пространство линейных функции степени n, d_H - расстояние по Хеммингу между столбцами со значениями булевых функции.

Определение 2.3. Регистр сдвига, или сдвиговый регистр (англ. shift register) представляет собой устройство, состоящее из последовательно соединенных триггеров (по сути ячеек, хранящих 0 или 1)

Основной функцией регистра сдвига является последовательное сдвигание разрядов (битов) внутри него, после которого обычно снимается выходной бит для дальнейшего использования. Это сдвигание происходит в ответ на тактовый сигнал.

Определение 2.4. Регистр сдвига с линейной обратной связью (РСЛОС, англ. linear feedback shift register, LFSR) — сдвиговый регистр, у которого значение входного бита равно линейной булевой функции от значений всех битов регистра до сдвига.

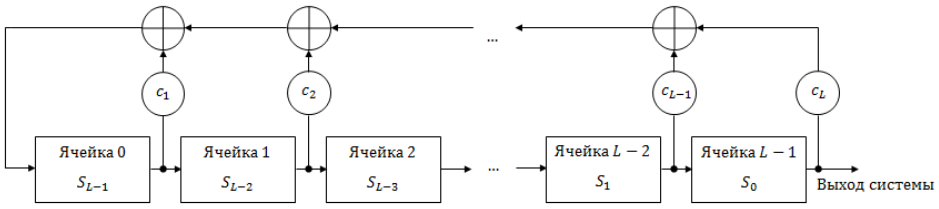


Рис. 1. Регистр сдвига с линейной обратной связью

Определение 2.5. Определим линейную сложность L(s) бинарной бесконечной последовательности s:

1) Если s - это последовательность из нулей, то L(s) = 0.
2) Если не существует LFSR, генерирующего s, то L(s) = inf
3) Если существует хотя бы один LFSR, генерирующий s, то L(s) - это длина самого короткого LFSR, который генерирует s.

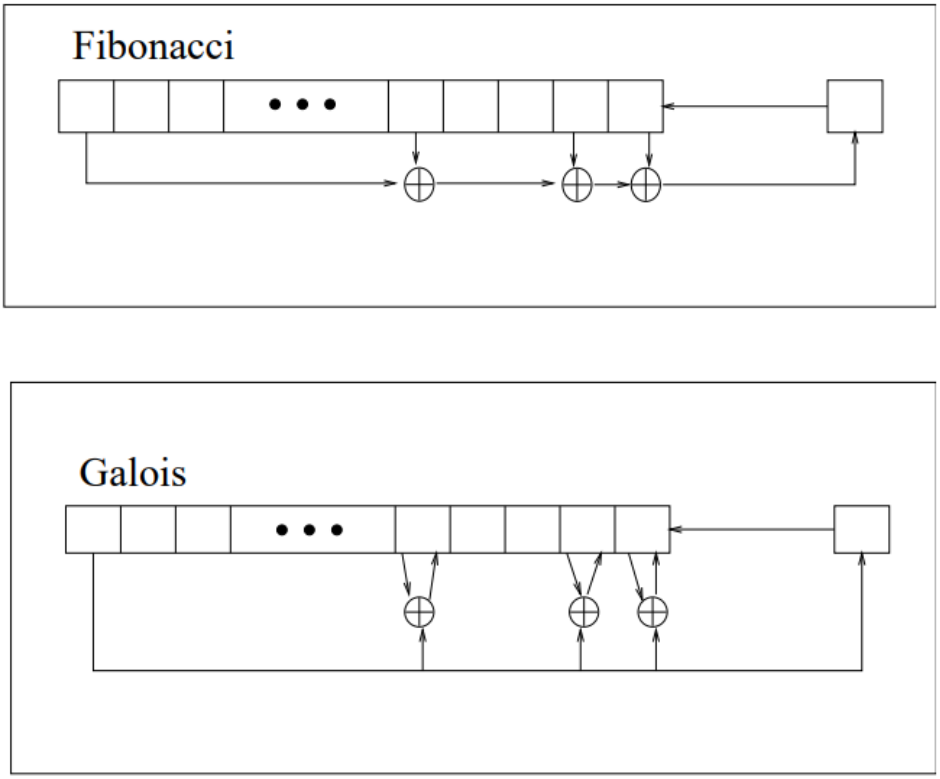


Рис. 2. Конфигурация Фибоначчи и конфигурация Галуа

Определение 2.6. Конфигурация Фиббоначи - это способ реализации LSFR, при котором есть одна функция обратной связи. За 1 такт работы регистра происходит вычисление первого бита с помощью этой функции, а также сдвиг регистра.

Определение 2.7. Конфигурация Галуа - это способ реализации LSFR, при котором для каждой ячейки есть своя функция обратной связи. За 1 такт работы регистра происходит обновление каждой ячейки с помощью своей функции, а также сдвиг регистра.

Замечание 2.1. Так как глубина схем обратной связи для отдельных битов обычно меньше, чем для функции обратной связи в конифгурации Фибоначчи, то для конфигурации Галуа время 1 такта может быть уменьшено [2].

Определение 2.8. Корреляционная атака (или атака типа "разделяй и властвуй") - это класс атак на основе открытых текстов для взлома потоковых шифров, ключевая последовательность которых генерируется путем объединения вывода нескольких LSFR (но не всех) с использованием логической функции.

Определение 2.9. Быстрая корреляционная атака [4] (англ. fast correlation attack, fca) - это корреляционная атака, которая происходит существенно быстрее чем обычная корреляционная атака за счет того, что проблему рассматривают в контексте теории кодирования.

Определение 2.10. Атака компромисса между временем/памятью/данными (time-memory tradeoff atack) на потоковые шифры [7] - тип криптографической атаки, в которой заранее вычисляется таблица, содержащая пары вида (начальное состяние, ключевая последовательность фиксированной длины для этого состояния), и затем производится поиск подпоследовательности ключевой последовательности в этой таблице. При попадании в таблицу с высокой вероятностью, соотвествующее начальное состояние является правильным. Такое название объясняется тем, что злоумышленник ищет компромисс между памятью, которая у него есть, наблюдаемыми данными и количеством операции, ведь улучшение одного параметра ведет к ухудшению другого.

Определение 2.11. Сеть Фейстеля - это одна из архитектур блочного шифра. Исходный текст делится на блоки, и над каждым блоком производится одно и тоже преобразование. На вход этому преобразованию подается блок и ключ. Вот так выглядит это преобразование:

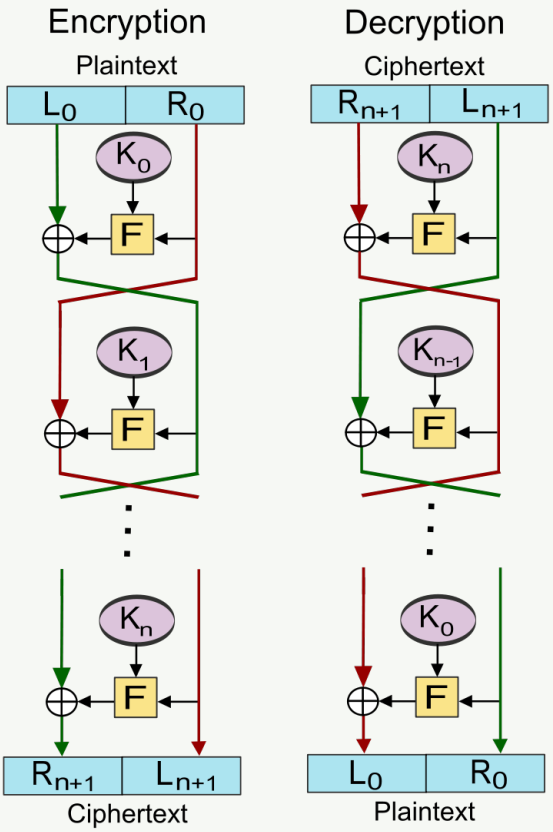


Рис. 3. Сеть Фейстеля [17]

Исходный текст (блок) делится на две части: R_0, L_0. Из ключа создаются подключи, количество которых ровно количеству раундов. После раунда (L_i, R_i) изменяется следующим образом:

1) $L_{i+1} = R_i$

2) $R_{i+1} = L_i \oplus F(R_i, K_i)$

Происходит N раундов, и (L_n, R_N) является выходом преобразования. Причем последний раунд происходит без перестановки:

1) $L_{i+1} = L_i \oplus F(R_i, K_i)$

2) $R_{i+1} = R_i$

Функция F может быть разной. Расшифрование происходит в обратном порядке.

Определение 2.12. Пусть у нас есть семейство функции $F(k, x)$, где k - это параметр, а x - переменная. Для каждого k определена функция $F_k(x)$, которые действуют из множества A в множество B. Функция $F(k, x)$, при случайном и равновероятном выбранном k называется псевдослучайной, если $F_k(x)$ неотличима (за разумное время) от истинно случайной функции, действующей из A в B.

3. RC4

В 1987 году сотрудник компании RSA Security Рональд Ривест создал потоковый шифр RC4. В течение 7 лет шифр являлся коммерческой тайной. Вскоре описание RC4 было опубликовано в группе новостей usenet.

3.0.1. Описание шифра RC4

RC4 генерирует ключевую последовательность длины сообщения, которая затем накладывается на исходное сообщение и получается шифротекст. На вход алгоритму подаются 2 параметра: секретный ключ K длиной l байт (которая не может быть больше 256 байт) и n - длина выходной ключевой последовательности. Затем идут 2 этапа: инициализация и генерация ключевой последовательности.

Инициализация генератора

С помощью секретного ключа K, генерируется случайная подстановка длины n следующим образом:

Algorithm 1 RC4 key scheduling

1. {initialization}
2. **for** i **from** 0 **to** $n-1$ **do**
3. $S[i] := i$
4. **end for**
5. $j := 0$
6. {generate a random permutation}
7. **for** i **from** 0 **to** $n-1$ **do**
8. $j := (j + S[i] + K[i \mod l]) \mod n$
9. Swap $S[i]$ and $S[j]$
10. **end for**

Рис. 4. Инициализация RC4 [29]

Этап генерации ключевой последовательности

Генерация ключевой последовательности происходит следующим образом:

Algorithm 2 RC4 pseudo random generator

1. {initialization}
2. $i := 0$
3. $j := 0$
4. {generate pseudo random sequence}
5. **loop**
6. $i := (i + 1) \mod n$
7. $j := (j + S[i]) \mod n$
8. Swap $S[i]$ and $S[j]$
9. $k := (S[i] + S[j]) \mod n$
10. **print** $S[k]$
11. **end loop**

Рис. 5. Генератор псевдослучайной последовательности RC4 [29]

3.0.2. Безопасность шифра

В 2007 году было показано [30], что первый байт ключевого потока коррелирован с первыми тремя байтами ключа. В 2015 году была представлена реальная атака на протокол TLS, испольщующий RC4 [31]. С помощью около $2^{27} - 2^{30}$

пар открытого текста и шифротекста за 52 часа они смогли восстановить ключ.

4. Snow3G

Шифры семейства Snow были разработаны Лундским Университетом (Швеция) для безопасной передачи мобильных данных. В 2006 году была выпущена новая версия шифра Snow - Snow3G [27]. Это поточный шифр, работающий в синхронном режиме, который производит ключевую последовательность, кратную 32-битам.

4.0.1. Описание шифра Snow3G

Потоквый шифр Snow3G принимает на вход 2 параметра: 128-битный секретный ключ и 128-битный инициализирующий вектор. Шифр имеет 608-битное внутреннее состояние, которое формируются двумя компонентами: регистром и конечным автоматом (finite state machine, FSM). Схема Snow3G выглядит следующим образом:

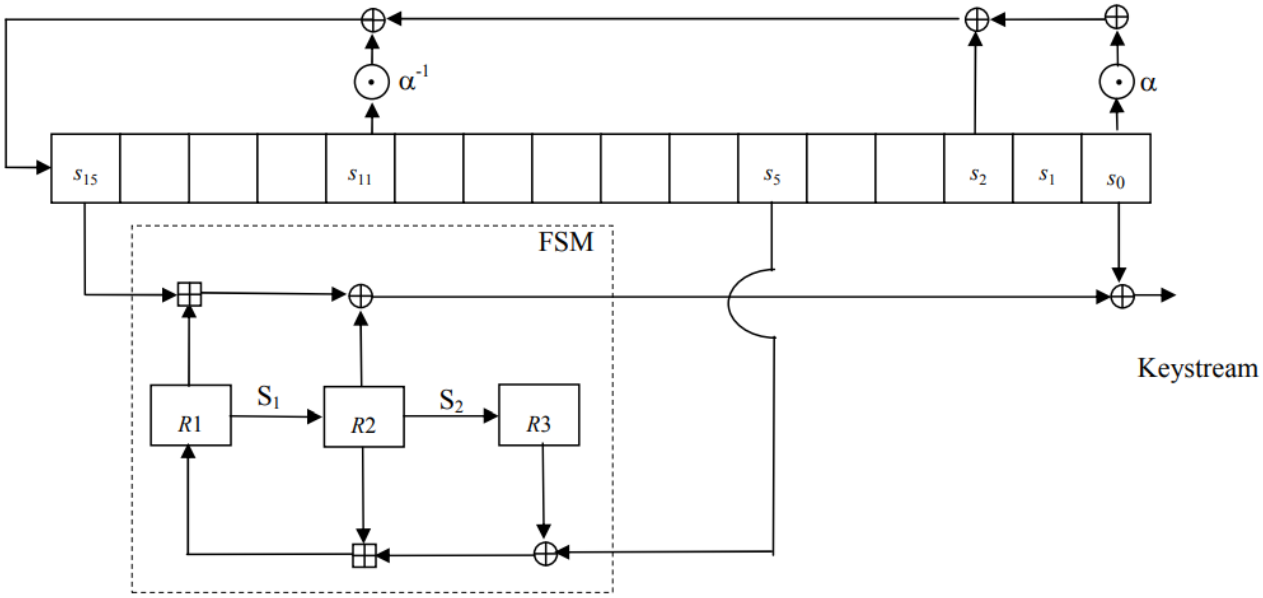


Рис. 6. Enter Caption

Регистр является LSFR и состоит из 16 32-битных ячеек. Функция обратной связи выглядит так:

$$s_{t+16} = \alpha * s_t \oplus s_{t+2} \oplus \alpha^{-1} * s_{t+11}.\alpha \in [0, \ldots, 2^{32} - 1] \tag{4.1}$$

Тактирование FSM.

FSM состоит из 3 32-битных регистров R_1, R_2, R_3 и двух S-box S_1, S_2 (S_1 взят из AES, S_2 разработан специально для Snow3G, описание последнего в [27]). FSM принимает на вход s_5 и s_{15} из LSFR, а на выходе выдает 32-битное значение F. Это происходит следующим образом:

1) $F = (s_{15} \oplus R_1) \oplus R_2$
2) $r = R_2 \oplus (R_3 \oplus s_5)$
3) $R_3 = S_2(R_2)$
4) $R_2 = S_1(R_1)$
5) $R_1 = r$

Инициализация генератора.

На вход генератор получает 128-битный ключ k, который разбивается на 4 равные части k_0, k_1, k_2, k_3 и 128-битное инициализирующее значение IV, которое также бьется на IV_0, IV_1, IV_2, IV_3. 11 = 0xffffffff. Инициализация регистра происходит следующим образом:

$s_{15} = \mathbf{k_3} \oplus \mathbf{IV_0}$	$s_{14} = \mathbf{k_2}$	$s_{13} = \mathbf{k_1}$	$s_{12} = \mathbf{k_0} \oplus \mathbf{IV_1}$
$s_{11} = \mathbf{k_3} \oplus \mathbf{1}$	$s_{10} = \mathbf{k_2} \oplus \mathbf{1} \oplus \mathbf{IV_2}$	$s_9 = \mathbf{k_1} \oplus \mathbf{1} \oplus \mathbf{IV_3}$	$s_8 = \mathbf{k_0} \oplus \mathbf{1}$
$s_7 = \mathbf{k_3}$	$s_6 = \mathbf{k_2}$	$s_5 = \mathbf{k_1}$	$s_4 = \mathbf{k_0}$
$s_3 = \mathbf{k_3} \oplus \mathbf{1}$	$s_2 = \mathbf{k_2} \oplus \mathbf{1}$	$s_1 = \mathbf{k_1} \oplus \mathbf{1}$	$s_0 = \mathbf{k_0} \oplus \mathbf{1}$

Рис. 7. Инициализация регистра [27]

FSM инициализируется нулями: $R_1 = R_2 = R_3 = 0$. Затем производится следующие шаги в цикле 32 раз, не производя ключевой последовательности:

1) FSM тактируется и выдает F.
2) LSFR тактируется и добавляет в обновляемый бит F: $s_{t+16} = s_{t+16} \oplus F$ (при этом обычное вычисление обратной функции уже произошло).

Генерация ключевой последовательности.

Пусть уже прошел этап инициализации. Сначала 1 раз тактируется FSM (его выход F никак не используется). Затем 1 раз тактируется LSFR (его выход никак не используется). Далее производится заданное количество n 32-битных слов, повторяя следующие шаги n раз:

1) FSM тактируется и выдает F.
2) Выходное 32-битное слово вычисиляется как: $z_t = F \oplus s_0$.
3) LSFR тактируется.

4.0.2. Безопасность шифра

Группа 3GPP провели многосторонний анализ безопасности Snow3G [28], пробуя применить разные атаки, особенно которые были эффективны против прошлых версии шифра. В результате они заключили, что шифр обеспечивает адекватную защиту конфиденциальности.

5. SEAL

Одним из важнейших параметров любого шифра является его скорость шифрования. Многие добивались этой скорости за счет аппаратно-ориентированных шифров и их аппаратной реализации. Phillip Rogaway и Don Coppersmith были следующей точки зрения: не всегда есть возможность купить криптографическое оборудование и использовать аппаратно-ориентированный шифр. Программная реализация таких шифров обычно существенно проигрывала в скорости и являлась непрактичной. Поэтому в 1993 году Rogaway и Don Coppersmith разработали потоковый шифр SEAL (Software-optimized Encryption Algorithm) [20], оптимизированный для программной реализации.

5.0.1. Описание шифра SEAL

SEAL является семейством псевдослучайных функций. Его параметрами являются: a - 160-битный ключ, n - 32-битная переменная (в алгоритме ее роль обычно играет счетчик сообщений) и L - длина выходной последовательности. $SEAL(a, n, L)$ - семейством псевдослучайных функций, а $SEAL(a, *, L) = SEAL_{a,L}()$ - функция от параметра n, которая неотличима от случайной. Также авторы отмечают, что a - может быть ключом переменной длины, просто

стоит от него будет взять хэш-функцию $SHA-1$, чтобы спроецировать его в 160-битный ключ для SEAL. Предлагаемая схема шифрования потока сообщений $(m_1, m_2, \ldots, m_N)$ такова: $c_i = (n, x \oplus SEAL(a, n, L))$, где n - номер сообщения, а $L = |x|$.

Алгоритму на вход подаются: (a, n, L). Сначала из a - генерируются таблицы T, R, S с помощью $SHA-1$ (подробннее об этом в [20]). Перед описанием основного алгоритма нужно ознакомиться с процедурой initialize:

procedure Initialize$(n, \ell,\ A, B, C, D,\ n_1, n_2, n_3, n_4)$

$A \leftarrow n \oplus R[4\ell]$;
$B \leftarrow (n \gg 8) \oplus R[4\ell + 1]$;
$C \leftarrow (n \gg 16) \oplus R[4\ell + 2]$;
$D \leftarrow (n \gg 24) \oplus R[4\ell + 3]$;

for $j \leftarrow 1$ **to** 2 **do**
  $P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \gg 9$;
  $P \leftarrow B$ & 0x7fc; $C \leftarrow C + T[P/4]$; $B \leftarrow B \gg 9$;
  $P \leftarrow C$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \gg 9$;
  $P \leftarrow D$ & 0x7fc; $A \leftarrow A + T[P/4]$; $D \leftarrow D \gg 9$;

$(n_1,\ n_2,\ n_3,\ n_4) \leftarrow (D,\ B,\ A,\ C)$;

$P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \gg 9$;
$P \leftarrow B$ & 0x7fc; $C \leftarrow C + T[P/4]$; $B \leftarrow B \gg 9$;
$P \leftarrow C$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \gg 9$;
$P \leftarrow D$ & 0x7fc; $A \leftarrow A + T[P/4]$; $D \leftarrow D \gg 9$;

Рис. 8. Процедура initialize [20]

Ей на вход подаются n и L. По окончании процедуры устанвливаются значения для A, B, C, D, n_1, n_2, n_3 и n_4. Заметим, что перед этим таблицы T и R уже созданы с помощью a.

Основной алгоритм шифра SEAL:

function SEAL(a, n, L)
$y = \lambda$;
for $\ell \leftarrow 0$ **to** ∞ **do**
  Initialize$(n, \ell,\ A, B, C, D,\ n_1, n_2, n_3, n_4)$;
  for $i \leftarrow 1$ **to** 64 **do**
1 $P \leftarrow A$ & 0x7fc; $B \leftarrow B + T[P/4]$; $A \leftarrow A \gg 9$; $B \leftarrow B \oplus A$;
2 $Q \leftarrow B$ & 0x7fc; $C \leftarrow C \oplus T[Q/4]$; $B \leftarrow B \gg 9$; $C \leftarrow C + B$;
3 $P \leftarrow (P + C)$ & 0x7fc; $D \leftarrow D + T[P/4]$; $C \leftarrow C \gg 9$; $D \leftarrow D \oplus C$;
4 $Q \leftarrow (Q + D)$ & 0x7fc; $A \leftarrow A \oplus T[Q/4]$; $D \leftarrow D \gg 9$; $A \leftarrow A + D$;
5 $P \leftarrow (P + A)$ & 0x7fc; $B \leftarrow B \oplus T[P/4]$; $A \leftarrow A \gg 9$;
6 $Q \leftarrow (Q + B)$ & 0x7fc; $C \leftarrow C + T[Q/4]$; $B \leftarrow B \gg 9$;
7 $P \leftarrow (P + C)$ & 0x7fc; $D \leftarrow D \oplus T[P/4]$; $C \leftarrow C \gg 9$;
8 $Q \leftarrow (Q + D)$ & 0x7fc; $A \leftarrow A + T[Q/4]$; $D \leftarrow D \gg 9$;
9 $y \leftarrow y \ \|\ B + S[4i-4] \ \|\ C \oplus S[4i-3] \ \|\ D + S[4i-2] \ \|\ A \oplus S[4i-1]$;
10 **if** $|y| \geq L$ **then return** $y_0 y_1 \ldots y_{L-1}$;
11 **if** $odd(i)$ **then** $(A,\ B,\ C,\ D) \leftarrow (A + n_1,\ B + n_2,\ C \oplus n_1,\ D \oplus n_2)$
      else $(A,\ B,\ C,\ D) \leftarrow (A + n_3,\ B + n_4,\ C \oplus n_3,\ D \oplus n_4)$;

Рис. 9. Алгоритм шифра SEAL [20]

Инициализация происходит каждые $32 * 4 * 64$ битов, пробегая по l. Параметр L по факту является критерием

останова. В строках 1-8 происходит генерация нашей псевдослучайной функции с помощью специальных констант, битовых операции, сложения 2 беззнаковых чисел и обращения к ранее вычисленным таблицам T и S. Заметим, что для фиксированного a и n, максимальное значение $L = 64 * 1024 * 8$ или 64 килобайта.

5.0.2. Безопасность шифра

Авторы работы [21] смогли показать, что с помощью 2^{34} битов ключевой последовательности можно отличить SEAL от истинно случаной функцией. То есть SEAL не является псевдослучайной функцией. Позже начали появляться более надежные версии шифра SEAL.

6. MUGI

В 2002 году был создан генератор псевдослучаных чисел MUGI [22], в основе которого лежат принципы шифра PANAMA [23]. При разработке MUGI преследовалось 2 цели:

1) Шифр должен иметь возможность эффективно реализовываться на разном аппаратном оборудовании.

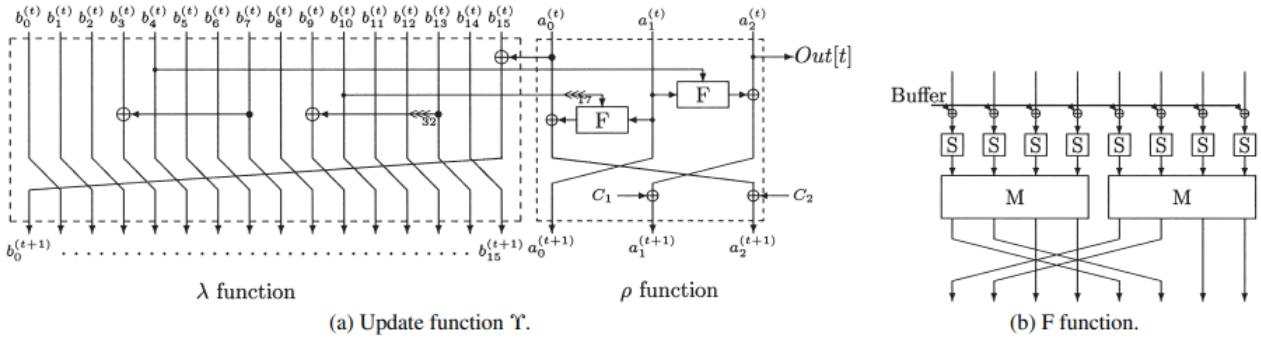
2) Сложность вычисления шифра должна быть меньше, чем у PANAMA.

6.0.1. Описание шифра MUGI

MUGI получается на вход 2 параметра: 128-битный секретный ключ и 128-битный инициализирующий вектор. Каждый раунд шифр производит 64-битную ключевую последовательность. Входные параметры формируют состояние **a** и буфер **b**. Во время раунда t состояние **a** и буфер **b** выглядят следующим образом:

$$\begin{cases} \mathrm{a}^{(t)} = \left(a_0^{(t)}, a_1^{(t)}, a_2^{(t)}\right) \left(a_i^{(t)} \in \mathrm{GF}(2)^{64}\right) \\ \mathrm{b}^{(t)} = \left(b_0^{(t)}, \cdots, b_{15}^{(t)}\right) \left(b_i^{(t)} \in \mathrm{GF}(2)^{64}\right) \end{cases} \tag{6.1}$$

Инициализация **a(0)** и **b(0)** происходит с помощью секретного ключа и инициализирующего вектор (описание этого этапа находится в [22]). Выходом генератора в момент t является: $Out[t] = a_2^(t)$, то есть второй юнит (вторые 64-бита) состояния **a(t)**.



(a) Update function ϒ. (b) F function.

Рис. 10. Схема функций шифра MUGI [24]

Каждый раунд происходит обновление **a** и **b** с помощью функции ϒ, λ, ρ (они представлены на схеме выше) следующим образом:

$$\left(\mathrm{a}^{(t+1)}, \mathrm{b}^{(t+1)}\right) = \Upsilon\left(\mathrm{a}^{(t)}, \mathrm{b}^{(t)}\right) = \left(\lambda\left(\mathrm{a}^{(t)}, \mathrm{b}^{(t)}\right), \rho\left(\left(\mathrm{a}^{(t)}, \mathrm{b}^{(t)}\right)\right)\right) \tag{6.2}$$

λ function.

$$
\begin{aligned}
b_j^{(t+1)} &= b_{j-1}^{(t)} \\
b_0^{(t+1)} &= b_{15}^{(t)} \oplus a_0^{(t)} \\
b_4^{(t+1)} &= b_3^{(t)} \oplus b_7^{(t)} \\
b_{10}^{(t+1)} &= b_9^{(t)} \oplus \left(b_{13}^{(t)} \lll 32\right)
\end{aligned}
\tag{6.3}
$$

λ **function**.

$$
\begin{aligned}
a_0^{(t+1)} &= a_1^{(t)} \\
a_1^{(t+1)} &= a_2^{(t)} \oplus \mathrm{F}\left(a_1^{(t)}, b_4^{(t)}\right) \oplus C_1 \\
a_2^{(t+1)} &= a_2^{(t)} \oplus \mathrm{F}\left(a_1^{(t)}, b_{10}^{(t)} \lll 17\right) \oplus C2 \\
C_1 &= 0x\mathrm{BB67AE8584CAA73B} \\
C_2 &= 0x\mathrm{3C6EF372FE94\ F82\ B}
\end{aligned}
\tag{6.4}
$$

S-box S и матрица M взяты из шифра AES.

6.0.2. Безопасность шифра

Специалисты отмечают [25], что у буфера, который обновляется линейно, низкая линейная сложность - 32 и низкий период последовательнсоти - 48. Это создает риски восстановления секретного ключа. Также были найдены возможные уязвимости нелинейной компоненты - состояния **a** [26].

7. Семейство шифров A5

Семейство поточных шифров A5: A5/1, A5/2 и A5/3 используется в шифрования данных между телефоном и базовой станцией в европейской системе мобильной цифровой связи GSM (Groupe Spécial Mobile).

7.1. A5/1

A5/1 представляет собой вторую по надежности версию алгоритма шифрования A5, применяемого в стандарте GSM для обеспечения конфиденциальности более чем 130 миллионов клиентов. Изначально шифр держался в секрете. Краткое описание дизайна A5/1 утекло в 1994 году, и устройство шифра было с получено с помощью обратной инженерии в 1999 году Брисеньо [3] на основе реального GSM-телефона.

A5/1 - потоковый шифр, основанный на трех линейных регистрах сдвига обратной связи, которые нерегулярно тактируются. Размер ключа составляет 64 бита, и последовательность ключей формируется путем выполнения операции XOR с выходами трех регистров.

7.1.1. Устройство шифра A5/1

Устройство A5/1 достаточно просто, его можно увидеть на следующей схеме:

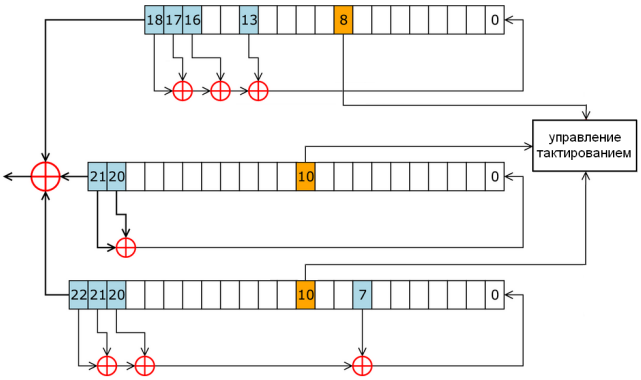


Рис. 11. Система регистров в алгоритме A5/1 [11]

У нас есть три $LSFR$ R_1, R_2, R_3 длиной 19, 22, и 23 бита соотвественно. Они имеют следующие функции обратной связи:

$R_1 : x^{19} + x^{18} + x^{17} + x^{14} + 1$

$R_2 : x^{22} + x^{21} + 1$

$R_3 : x^{23} + x^{22} + x^{21} + x^8 + 1$

Пусть регистры проинициализированы. Опишем 1 такт работы шифра:

1) Берутся 3 бита из каждого регистра, 8-ой бит из R_1, 10-ой бит из R_2, 10-ой бит из R_3/.
2) От этих 3 битов вычисляется функция $F = xy|xz|yz$.
3) Тактируются только те регистры, для которых полученное значение функции равно биту регистра, который подавался в фуннкцию F.
4) Затем выходные биты регистров складываются по модулю 2 и результат подается на выход генератора.

7.1.2. Функционирование шифра A5/1

.

Есть 64-битный регистр

Данные в сеансе GSM передаются в виде последовательности кадров, где один кадр отправляется каждые 4,6 миллисекунды. Каждый кадр содержит 114 битов, идущих от A к B, и еще 114 битов, идущих от B к A. Каждый сеанс шифруется новым сеансовым ключом. Затем идет инициализация генератора, описание которой будет ниже. Затем генерируется 228 бит ключевой последовательности, которая накладывается на 228 битов открытого текста.

Инициализация A5/1.

На вход алгоритма подаются сеансовый 64-битный ключ K и 22-битный номер кадра F_n

1) Обнуление регистров.
2) 64 такта с регулярным тактированием, при которых очередной бит ключа складывается по модулю 2 с младшим битом каждого регистра и регистры сдвигаются
3) аналогичные 22 такта, только с номером F_n
4) 100 тактов с нерегулярным тактированием

После инициализации генерируется ключевая последовательность длиной в 228 бит.

7.1.3. Безопасность шифра

Одной из первых атак была атака компромисса между временем/памятью/данными [12] с вычислительной сложностью около $2^{40} - 2^{45}$. В 2000 году была презентована новая атака компромисса между временем/памятью/данными [13], которая выполняется на персональном компьютере не дольше нескольких минут. Однако она требует 150-300 гигабайт памяти для своей таблицы.

7.2. A5/2

A5/2 - синхронный потоковый шифр, который является модификацией шифра A5/1. A5/2 был специально разработан как экспортный вариант для стран, не входивших в Евросоюз. При этом криптостойкость модификации шифра является более низкой.

7.2.1. Устройство шифра A5/2

Устройство A5/2 имеет некоторые отличия от устройства A5/1:

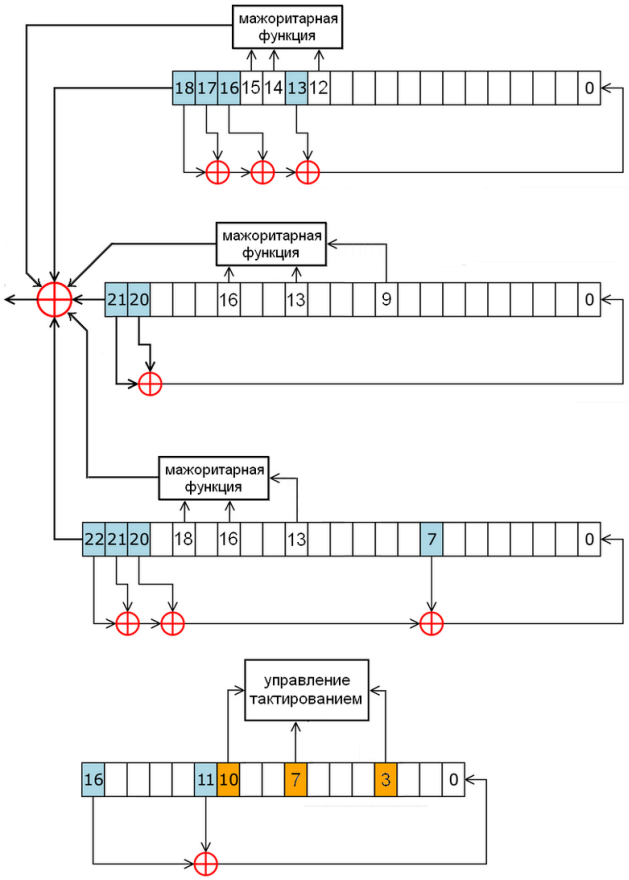


Рис. 12. Устройство A5/2 [14]

У нас есть 4 $LSFR$ R_1, R_2, R_3, R_4 длиной 19, 22, 23, 17 бита соотвественно. Они имеют следующие функции обратной связи:
$R_1 : x^{19} + x^{18} + x^{17} + x^{14} + 1$
$R_2 : x^{22} + x^{21} + 1$
$R_3 : x^{23} + x^{22} + x^{21} + x^8 + 1$
$R_4 : x^{17} + x^{12} + 1$
Пусть регистры проинициализированы. Опишем 1 такт работы шифра:

1) Берутся 3 бита из регистра R_4: 3, 7 и 10.
2) От этих 3 битов вычисляется функция голосования $F = xy|xz|yz$.
3) R_1 тактируется, если 10-ый бит $R_4 = F$, R_2 тактируется, если 3-ий бит $R_4 = F$, R_3 тактируется, если 7-ой бит $R_4 = F$.

4) Затем от каждого из первых трех регистров берется функция голосования для своих 3 битов: $R_1 - x_{12}, x_{14} \oplus 1, x_{15}$, $R_2 - x_9, x_{13}, x_{16} \oplus 1, R_3 - x_{13} \oplus 1, x_{16}, x_{18}$. Результаты этих функции суммируются по модулю 2 с выходными битами первых трех регистров. Полученное значение подается на выход генератора.

7.2.2. Функционирование шифра A5/2

Такое же как и в A5/1.

Инициализация A5/2.

Инициализация претерпела изменения. На вход алгоритма подаются сеансовый 64-битный ключ K и 22-битный номер кадра F_n

1) Обнуление регистров.
2) 64 такта с регулярным тактированием, при которых очередной бит ключа складывается по модулю 2 с младшим битом каждого регистра (R_1, R_2, R_3 и R_4) и регистры сдвигаются.
3) аналогичные 22 такта, только с номером F_n
4) 1 такт 3 бита R_4 (3, 7, 10) заполняются единицами.
5) 99 тактов с нерегулярным тактированием.

После инициализации генерируется ключевая последовательность длиной в 228 бит.

7.2.3. Безопасность шифра

Этот шифр не является безопасным. Например, есть атака, которая без дополнительных вычислений за секунды может взломать шифр [15].

7.3. A5/3

Около 20 лет алгоритмы A5/1 и A5/2 стояли на страже "безопасности"телефонных разговоров, эти шифры были недостаточно стойкими. Поэтому на смену им приходит новый блочный шифр KASUMI, который ложится в основу A5/3. KASUMI берет свои истоки из криптосистемы MISTY.

После внедрения, A5/3 станет одной из наиболее широко используемых криптосистем в мире, и обеспечение её безопасности станет одним из наиболее важных практических вопросов в области криптографии.

7.3.1. Описание KASUMI

Этот алгоритм был опубликован в спецификации [16] в 2003 году. KASUMI - это сеть Фейстеля с 8 раундами. Он работает с блоком данных размером 64 бита и использует ключ размером 128 бит. Опишем функцию раунда f_i.

Функция раунда f_i.

Функция раунда f_i принимает 32-битный блок I, ключ раунда $RK_i = (KL_i, KO_i, KI_i)$. Функция f_i зависит от раунда. Внутрии ее используются функции FO, FL В 1,3,5,7 раундах она равна:

$$f_i(I, RK_i) = FO(FL(I, KL_i), KO_i, KI_i) \tag{7.1}$$

В 2,4,6,8 раундах она равна:

$$f_i(I, RK_i) = FL(FO(I, KO_i, KI_i), KL_i) \tag{7.2}$$

Функция FL.

Функция FL принимает 32-битный блок $I = (L||R)$ и 32-битный подключ $KL_i = (KL_{i,1}||KL_{i,2})$.

1) $R' = R \oplus ROL(LandKL_{i,1})$
2) $L' = L \oplus ROL(R'orKL_{i,1})$

ROL это циклический сдвиг влево. Выходом функции является $(L'||R')$.

Функция FO.

Функция FO принимает 32-битный блок $I = (L_0||R_0)$, 48-битный подключ $KO_i = KO_{i,1}||KO_{i,2}||KO_{i,3}$ и 48-битный подключ $KI_i = KI_{i,1}||KI_{i,2}||KI_{i,3}$.

Для $j = 1, 2, 3$:

1) $R_j = FI(L_{j-1} \oplus KO_{i,j}, KI_{i,j}) \oplus R_{j-1}$, функция FI, которая будет описана ниже
2) $L_j = R_{j-1}$

Выходом функции является $(L_3||R_3)$.

Функция FI.

Функция FI принимает 16-битный блок $I = (L_0||R_0)$ и 16-битный подключ $KI_{i,j} = (KI_{i,j,1}||KI_{i,j,2})$. L_0 - 9 битов, L_0 - 7 битов. $KI_{i,j,1}$ - 7 битов, $KI_{i,j,2}$ - 9 битов.

Также в функции FI используются две подстановки: 7-битная $S7$ и 9-битная $S9$, которые представлены в [16]; две фукнции $ZE(x)$ и $TR(x)$: $ZE(x)$ -> 00$||x$, $TR(x)$ -> $x[2\ :]$, то есть первая фукнция добавляет 2 нуля на место самых значимых битов, а вторая отрезает 2 самых значимых бита.

1) $L_1 = R_0$
2) $R_1 = S9[L_0] \oplus ZE(R_0)$
3) $L_2 = R_1 \oplus KI_{i,j,2}$
4) $R_2 = S7[L_1] \oplus TR(R_1) \oplus KI_{i,j,1}$
5) $L_3 = R_2$
6) $R_3 = S9[L_2] \oplus ZE(R_2)$
7) $L_4 = S7[L_3] \oplus TR(R_3)$
8) $R_4 = R_3$

Выходом функции является $(L_4||R_4)$.

Расписание ключей.

KASUMI имеет 128-битный ключ K. Каждый раунд используется 128-битный ключ, который высчитывается из K. Сначала считаются 2 16-битных массива $\{K_j\}$ и $\{K_{j'}\}(j = 1to8)$ следующим образом. Первый массив это:

$$K = K1||K2||K3||K4||K5||K6||K7||K8 \tag{7.3}$$

А второй массив равен изначально ключу плюс заданной маске $(K + mask$. Затем полученное значение также делится на 8 равных частей. Маску можно найти в [16]. И далее из этих массивов на каждом раунде считаются соответствующие ключи:

	1	2	3	4	5	6	7	8
$KL_{i,1}$	K1<<<1	K2<<<1	K3<<<1	K4<<<1	K5<<<1	K6<<<1	K7<<<1	K8<<<1
$KL_{i,2}$	K3′	K4′	K5′	K6′	K7′	K8′	K1′	K2′
$KO_{i,1}$	K2<<<5	K3<<<5	K4<<<5	K5<<<5	K6<<<5	K7<<<5	K8<<<5	K1<<<5
$KO_{i,2}$	K6<<<8	K7<<<8	K8<<<8	K1<<<8	K2<<<8	K3<<<8	K4<<<8	K5<<<8
$KO_{i,3}$	K7<<<13	K8<<<13	K1<<<13	K2<<<13	K3<<<13	K4<<<13	K5<<<13	K6<<<13
$KI_{i,1}$	K5′	K6′	K7′	K8′	K1′	K2′	K3′	K4′
$KI_{i,2}$	K4′	K5′	K6′	K7′	K8′	K1′	K2′	K3′
$KI_{i,3}$	K8′	K1′	K2′	K3′	K4′	K5′	K6′	K7′

Рис. 13. Расписание ключей [16]

7.3.2. Функционирование алгоритма

Инициализация A5/1. На вход подается 32-битный вектор CC, 5-битный вектор CB, 1-битный вектор CD, 8-битный вектор CA, 16-битный вектор CE. Эти вектора загружаются в 64-битный регистрс следующим образом:

$$A = CC||CB||CD||00||CA||CE \tag{7.4}$$

Установим $KM = 0x55555555555555555555555555555555$, $KSB_0 = 0$. Далее 1 раз вызывается KASUMI для регистра A(блок) с подданным на инициализацию секретным ключом CK:

$$A = KASUMI[A]_{CK \oplus KM} \tag{7.5}$$

Инициализация завершена.

Генерация ключевой последовательности. Далее генируется N 64-битных блоков следующм образом, где BLKCNT - счетчик блоков, а CK - секретный ключ:

$$KSB_n = KASUMI[A \oplus BLKCNT \oplus KSB_{n-1}]_{CK} \tag{7.6}$$

Количество блоков генерируется ровно столько, сколько надо для покрытия шифруемой последовательности (в GSM - 114 бит, в ECSD - 348 бит). Таким образом, мы получаем ключевую последовательность, составленную из блоков.

7.3.3. Безопасность шифра

В работе [18] 2010 года описывается сендвич-атака на KASUMI, которая проводится за 2^{32} операции. При этом они используют знания о 4 ключах и сообщениях, что делает атаку неприменимой к A5/3. Авторы отмечают, что целью их работы было показать, что переход от MISTY к KASUMI ослабил криптографическую стойкость шифра, так как MISTY на момент публикации не имел атак эффективнее, чем полный перебор ключа.

В одной из последних работ [19], выпущенной в 2022 году, авторы представили атаку на основе выбора открытых текстов, которая требует 2^{63} битов исходного текста, и $2^{63.3}$ операции шифрования, и некоторые условия на ключ. Вероятность успеха этой атаки 2^{-18}.

Таким образом, один из самых используемых потоковых шифров в мире имеет свои недостатки, однако для взлома требуется достаточно специфичные условия.

8. Шифр LILI128

8.1. Описание

В 2000 году на NESSIE workshop был предложен шифр LILI128. LILI128 [1] это потоковый шифр, работающий в синхронном режиме и основанный на LSFR, с ключом в 128 бит. Разработчики утверждают, что у него большой период, большая линейная сложность, и что он устойчив к атакам, известным на 2002 год.

Рассмотрим его устройство подробнее на следующей схеме:

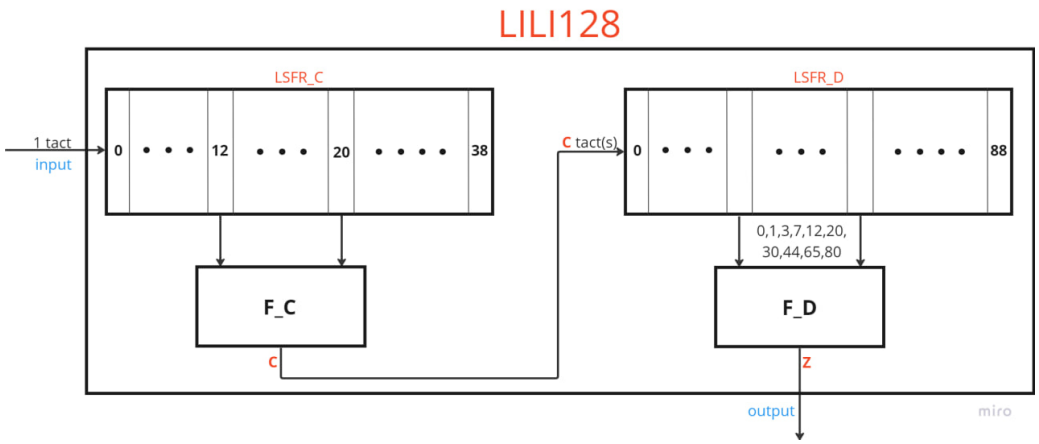


Рис. 14. Устройство LILI128

Шифр состоит из двух регистров $LSFR_C$ и $LSFR_D$, двух функции f_c и f_d. Пусть регистры проинициализированы. Опишем 1 такт работы шифра:

1) $LSFR_C$ производит 1 такт.
2) У регистра $LSFR_C$ берутся значения в 13 и 21 ячейках и от них берется функция f_c.
   $c = f_c(x_{12}, x_{20})$
3) Регистр $LSFR_D$ производит c тактов.
4) У регистра $LSFR_D$ берутся значения в 1,2,4,8,13,21,31,45,66,81 ячейках и от них считается функция f_d. Выходом шифра на 1 такте является результат этой функции f_d.

Функции шифра:

- Функция $f_c(x_{12}, x_{20}) = 2 * x_{12} + x_{20} + 1$.
- Функция f_d задана в виде таблицы, которая находится в приложении статьи [1].
- Функцией обратной связи для $LSFR_C$ является следующий примитивный полином:

$$x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1 \tag{8.1}$$

- Функцией обратной связи для $LSFR_D$ является следующий примитивный полином:

$$x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1 \tag{8.2}$$

Некоторые характеристики шифра:

- Период выходной последовательности $z(t)$ равен $(2^{39} - 1)(2^{89} - 1) \approx 2^{128}$.
- Выходная последовательность $z(t)$ сбалансирована, то есть ее количество нулей примерно равно ее количеству единиц (соотношение 1 к 0 равно $2^{88} : 2^{88} - 1$).
- Линейная сложность выходной последовательность $z(t)$ не меньше, чем 2^{68}
- Ненулевые ключи приводят к разным ключевым последовательностям. Хотя некоторые генераторы с нерегулярным тактированием при разных ключах выдают одну и ту же ключевую последовательность.

> **Замечание 8.1.** Нерегулярным тактированием регистра называется такое тактирование, при котором значение с регистра может сниматься как после 1 такта, так и после нескольких.

8.2. Особенности реализации

Программная реализация. Во время сдачи шифра на NESSIE workshop у авторов была реализация шифра LILI на языке C, которая работала со скоростью 4.8Мб/с (1200 циклов на байт) на 650МГц процессоре Pentium III с 128Мб ОЗУ. На момент написания статьи авторы добились скорости в 7.5Мб/с на той же машине. Первая реализация использовала конфигурацию Фиббоначи, а вторая - Галуа. При программной реализации конфигурация Галуа является более эффективной, так как состояние регистра складывается по модулю 2 с маской (которая содержит коэффициенты примитивного многочлена), а при Фиббоначи нужно складывать по модулю 2 все необходимые биты.

Авторы отмечают, что первоначальное вычисление маски Галуа приводит к дополнительным накладным расходам при инициализации, однако это происходит с незначительными затратами на скорость, так как это происходит только один раз.

Также приросту скорости второй реализации поспособствовал выбор размера слова процессора для виртуального регистра, типа unsigned (где это возможно), и побитового сдвига влево, вместо побитового сдвига вправо, так как он использовал меньше машинных инструкции.

Возможности аппаратной реализации. Для аппаратной реализации LSFR стоит использовать конфигурацию Фиббоначи. Для повышения скорости можно вместо нерегулярного тактирования поддерживать 4 копии функции обратной связи, с помощью которых можно перейти к регулярному тактированию. Авторы предпологают, что такая аппаратная реализация по производительности будет близка к скорости базового тактового генератора.

8.3. Атаки

В оригинальной статье [1] авторы утверждают, что для атаки "разделяй и властвуй"требуется минимум 2^{112} операции и 1700 битов ключевой последовательности. Также авторы утверждают, что на момент создания шифра не существовало быстрых корреляционных атак на нерегулярно тактируемый регистр. Меньше чем через год Fredrik Jönsson и Thomas Johansson опубликовали быструю корреляционную атаку [3], которая требовала операции меньше чем 2^{112}.

8.3.1. Быстрая корреляционная атака от Fredrik Jönsson и Thomas Johansson

В своей статье [3] авторы демонстрируют быструю корреляционную атаку на LILI128, которая требует около 2^{71} битовых операции при условии, что известна ключевая последовательность длиной около 2^{30}, и проведена фаза предварительных вычислений, количество которых равно 2^{79}. Основная концепция атаки была взята из более общей работы [5].

Угроза и модель нарушителя. Данная атака направлена на получение k битов начального состояния регистра $LSFR_d$, где k - является параметром. После удачной атаки таким же образом происходит восстанавливание всего начального состояния $LSFR_d$.

Это атака на основе знания открытого текста (known-plaintext attack, KPA), так как нарушитель знает ключевую последовательность, что эквивалентно знанию пар открытого текста и шифротекста.

Описание атаки.

Постановка. Предположим, что у нас есть ключевая последовательность длины N $z = (z_1, z_2, \ldots, z_N)$. Пусть выходная последовательность f_d будет $d = (d_1, d_2, \ldots, d_M)$, где $M > N$, когда $LSFR_d$ регулярно тактируется. Обозначим номера тактов $LSFR_d$ $s = (s_1, s_2, \ldots, s_N)$, которые берутся как выходные биты.

$$z_k = d_{s_k}, k = 1, \ldots, N \tag{8.3}$$

Предполагается значение регистра $LSFR_c$ и по нему высчитывается $s = (s_1, s_2, \ldots, s_N)$. Если это значение верно то: $z = (d_{s_1}, d_{s_2}, \ldots, d_{s_N}$. Далее по этой последовательности ищется начальное состояние регистра $LSFR_d$. Обозначим последовательность, произведенную $LSFR_d$ при регулярном тактировании, $u = (u_1, u_2, \ldots, u_N)$. Из нее мы получаем d_i, следуя описанию шифра (Рисунок 3):

$$d_i = f_d(u_{i-89}, u_{i-88}, u_{i-86}, u_{i-79}, u_{i-77}, u_{i-69}, u_{i-59}, u_{i-45}, u_{i-24}, u_{i-9}) \tag{8.4}$$

Далее требуется аппроксимировать f_d линейной функцией.

Аппроксимация f_d. Быстрые корреляционные атаки основаны на теории кодов, исправляющих ошибки [6]. Фильтрующая или комбинирущая функция, которая используется в шифрах, основанных на LSFR, интерпретируется как бинарный симметричный канал, через который передается информация. Функцию f_d необходимо аппроксимировать линейной, для того чтобы составить уравнения проверки паритета (parity check equations) [6].

Для этого необходимо произвести преобразование Уолша (Walsh Transform):

$$F(\omega) = \sum_{x} (-1)^{f(x) \oplus \omega \cdot x} \tag{8.5}$$

Также через него выражается нелинейность булевой функции:

$$N_f = 2^{n-1} - \frac{1}{2} \max_{\omega} |F(\omega)| \tag{8.6}$$

В статье [1] написано, что $N_{f_d} = 480$. Следовательно мы можем найти линейную функцию $f_l(x_1, x_2, \ldots, x_{10} = a_1 * x_1 + a_2 * x_2 + \cdots + a_{10} * x_{10}$, такую что $d_H(f_d, f_l) = 480$. Если так аппроксимировать функцию, то мы получаем что:

$$p = P\left(f_d(x) = f_l(x)\right) = \frac{1024 - 480}{1024} = 0.53125 \tag{8.7}$$

Разложив функцию f_d на спектр Уолша, мы получаем, что существует 240 функции, таких что $d_H(f_d, f_l) = 480$. Для атаки берутся все функции.

Предварительные вычисления. Далее идет модификация алгоритма [5] с параметром t=3. Множество L всех состоянии $LSRF_d$ длиной 89 имеет мощность $|L| = 2^{89}$ и последовательность фиксированной длины N, полученная из состояния из L, является линейным-[N, 89] кодом C с матрицей генератором G_{LSFR_d}.

Заменив f_d на одну из f_l, мы можем записать выход f_l как линейную комбинацию начального состояния u_0. Таким образом, мы можем найти матрицу G^{89*N}, с помощью которой выходная последовательность v, полученную из f_l, записывается как $v = u_0 * G$.

Так как линейных функции 240 $f_{l_1}, f_{l_2}, \ldots, f_{l_240}$, то мы можем построить 240 матриц генераторов $G_1, G_2, \ldots, G_{240}$, которая конкетинируется в большую матрицу G' размером $89 * 240N$:

$$G' = (G_1, G_2, \ldots, G_240) = (g_1, g_2, \ldots, g_{240N}) \tag{8.8}$$

Следуя алгортирму [5], выбирается параметр k < 89. Находятся все тройки стобцов $g_{i_1}, g_{i_2}, g_{i_3}$, таких что:

$$\left(\boldsymbol{g}_{i_1} + \boldsymbol{g}_{i_2} + \boldsymbol{g}_{i_3}\right)^{\mathrm{T}} = (\underbrace{*, *, \ldots, *}_{k}, \underbrace{0, 0, \ldots, 0}_{89-k}) \tag{8.9}$$

, где $*$ означает любое значение (но не все нули).

Поиск первых k битов регистра $LSFR_d$ ***по ключевой последовательности***. Пусть таких троек будет m. Для каждой такой тройки считается $v_{i_1} + v_{i_2} + v_{i_3}$, которая является линейной комбинацией только первых k символов изначального состояния и не зависит от остальных 89 - k других символов. Таким образом, формируется новый линейный-[m, k] код:

$$(v_{i_1(1)} + v_{i_2(1)} + v_{i_3(1)}, v_{i_1(2)} + v_{i_2(2)} + v_{i_3(2)}, \ldots, v_{i_1(3)} + v_{i_2(3)} + v_{i_3(3)}) \tag{8.10}$$

По этим же тройкам из ключевой последовательности формируется новая, равная зашумленной последовательности, сгенерированной из последнего линейного-[m, k] кода:

$$Z = (Z_1, Z_2, \ldots, Z_m), Z_j = (z_{i_1(j)} + z_{i_2(j)} + z_{i_3(j)}), 1 <= j <= m \tag{8.11}$$

Здесь же можно посчитать вероятность ошибки e переданного кода. $z_i = v_i + e_i$, где e_i случайная двоичная величина с:

$$P(e_i = 1) = p = P(z_i \neq v_i) = 1 - 0.53125 => \varepsilon = 0.03125. \tag{8.12}$$

А для переданного кода (полученного из троек):

$$p_3 = P(e_{i_1(j)} + e_{i_2(j)} + e_{i_3(j)} = 1) = 3p(1-p) + p^3 = 1/2 - 4 * \varepsilon^2 \approx 0.49988 \tag{8.13}$$

Это означает, что зашумленная последовательность почти не коррелирует с изначальной. Это усложняет сложность но атаки, но оставляет ее возможной.

Последним шагом из последного полученного кода полным перебором среди векторов длины k находится кодовое слово ближайшее к $Z = (Z_1, Z_2, \ldots, Z_m)$. Заметим что кодовое слово получается из вектора длины k, умноженного на матрицу генератор. Таким образом, первые k битов ближайшого кодового слова являются искомыми битами начального состояния при должной длине ключевой последовательности. Остальные биты начального состояния ищутся также, например, можно искать следующие k битов. Последующий поиск будет намного быстрее, так как на самом деле в первой итерации мы также перебираем полное начальное состояне $LSFR_c$ и в следующей итерации этого делать не надо.

Описаный выше алгоритм описан для регулярно тактируемого $LSFR_d$, который в LILI128 нерегулярно тактируется. Поэтому сведем задачу к предыдущей. Перед выдачей бита с регистра $LSFR_d$ тактируется в среднем 2.5 раза, поэтому $M \approx 2.5N$. Поэтому в фазе предварительных вычислений мы рассматриваем все $240M$ возможнных символов при конструкции линейного-[m, k] кода, из которых затем выбираем $240N$ символов с помощью последовательности s, получившейся из $LSFR_c$ из фиксированного состяния. И наконец, этот алгоритм выполняется для каждого возможного состояния $LSFR_c$. Поэтому мы и получаем после первой итерации $39 + k$ битов общего начального состояния шифра LILI128.

Оценка требуемых ресурсов для атаки. Приведем основные результаты расчетов из [3]. Для t = 3 желательная длина N равна:

$$N \approx \frac{1}{960} \cdot (k \cdot 12 \cdot \ln 2)^{1/3} \cdot \varepsilon^{-2} \cdot 2^{(89-k)/3} \tag{8.14}$$

Количество требуемых операции при этом равно:

$$2^k * k * \frac{2 * ln2}{(2\varepsilon)^{2t}} \tag{8.15}$$

Для шифра LILI128 с $l = 89$, $\varepsilon = 0.03125$ и $t = 3$ была получена следующая таблица, в котором для некоторых k расчитаны N и количество требуемых операции для декодирования C_{dec}:

k	N	C_{dec}
1	$2^{30.5}$	$2^{25.5}$
3	$2^{30.3}$	$2^{29.1}$
5	$2^{29.9}$	$2^{31.8}$
7	$2^{29.4}$	$2^{34.3}$
10	$2^{28.6}$	$2^{37.8}$
15	$2^{27.1}$	$2^{43.4}$
20	$2^{25.6}$	$2^{48.8}$
25	$2^{24.0}$	$2^{54.1}$

Рис. 15. Желательная длина ключевой последовательности N и количество требуемых операции для декодирования C_{dec} для заданного k и начального состояния $LSFR_c$

[3]

Для k=5 нужно около 2^{32} операции. Учитывая перебор по начальным состяниям $LSFR_c$ получается то, что суммарная вычислительная сложность атаки равна $2^{39} * 2^{32} = 2^{71}$. При этом необходима ключевая последоватльность размера 128Мб. Фаза предварительных вычислений требует 2^{79} операции.

Возможности практической реализации атаки. В 2023 году на самом мощном суперкомпьютере с частотой в 1600 петафлопс атака проводится за 24 минуты. Очевидно, что в 2002 году это атака еще не представляла угрозы. На тот

момент она имела важный теоритический результат, так как авторы шифра LILI128 были не правы, оценив сложность корреляционной атаки минимум в 2^{112} операции.

Пути нейтрализации атаки. В данной атаке, одним из основных параметров, определяющих сложность является ε. Поэтому если мы выберем фильтрующую функцию с нелинейностью больше, чем 480, то мы существенно увеличим сложность атаки.

8.3.2. Time-memory tradeoff atack на LILI128

Через 8 месяцев после публикации [3] Saarinen M. J. O. публикуют новую атаку [8], которая также опровергает утверждения разработчиков LILI128 о его безопасности. Это атака компромисса между временем/памятью/данными, которая обходит один защитных методов шифра - нерегулярное тактирование.

Угроза и модель нарушителя. Данная атака направлена на получение всего начального состояния регистра $LSFR_d$.

Это атака на основе знания открытого текста (known-plaintext attack, KPA), так как нарушитель знает ключевую последовательность, что эквивалентно знанию пар открытого текста и шифротекста.

Ознакомимся с леммами из статьи [8] без доказательства:

Лемма 8.1. При совершении $\Delta_c = 2^{39} - 1$ тактов $LFSR_c$, происходит ровно $\Delta_d = 5 * 2^{38} - 1$ тактов $LFSR_d$.

Лемма 8.2. Существует матрица размера $89 * 89$, такая что при умножении на нее вектора начального состояния $LSFR_d$ мы получим состояние $LSFR_d$, тактированное на Δ_d шагов от изначального. Аналогично есть матрица, которая отматывает регистр назад на Δ_d шагов от текущего. Для расчета такой матрицы потребуется около 2^{38} битовых операции.

Описание атаки.

Предварительный этап. Строится таблица, содержащая 2^{45} пар вида (89-битное слово, 45-битное слово). Каждая пара считается следующим образом. Берется случайный вектор длиной 89 и загружается в $LSFR_d$, из него семплируются 45 битов, которые в выходной последовательности находятся на расстоянии Δ_d. Семплирование происходит с помощью умножения начального состояния на специальную матрицу (см. Лемма 4.1)

Индексом таблицы является 45-битное слово, которое может быть получено из разных начальных состоянии, поэтому могут быть коллизии. Ожидаемая заполненность таблицы равна $1 - \exp -2 = 0.8647$. Ее размер - $2^{51.48}$. Вычислительная сложность построения - 2^{48} операции алгоритма Data Encryption Standard (DES).

Фаза просмотра таблицы. Пусть у нас есть 2^{46} битов ключевой последовательности $z(0), z(1), \ldots, z(2^{46} - 1)$

We have 2^{46} bits of keystream $z(0), z(1), \ldots, z(2^{46} - 1)$.

1. For i $= 0$ to $2^{46} - 44\Delta_c - 1$ Do:
2. idx $= z(i) \mid z(i + \Delta_c) \mid \ldots \mid z(i + 44\Delta_c)$
3. Load Table[idx] to $LFSR_d$.
4. Rewind $LFSR_d$ back $\Delta_d \lfloor \frac{i}{\Delta_c} \rfloor$ positions.
5. For j $= 0$ to 127 Do:
6. If $(f_d(LFSR_d) \neq z(j\Delta_c + (i \bmod \Delta_c))$ break loop.
7. Advance $LFSR_d$ by Δ_d positions.
8. If previous loop was not broken, return $LFSR_d$.

Рис. 16. Алгоритм атаки [8]

В первой строчке мы пробегаемся по всем подпоследовательностям z длины 45, элементы которых находятся на расстоянии Δ_c друг от друга. В 2ой строчке берем очередную подпоследовательность. Далее ищем ее в нашей таблице, при успехе вытаскиваем соотвествующее начальное значение регистра $LSFR_d$, иначе следущая итерация. Затем откатываем состояние регистра на число кратное Δ_d, если i перескочило через Δ_c. В строчках 5-7 мы проверяем совпадает ли сгенерированная из $LSFR_d$ 128-битная последовательность (с шагом в Δ_d) с наблюдаемой z (с шагом Δ_c). И если она совпала, то возвращаем текущее начальное состояние регистра.

Вероятность того, что состояние будет верным равно:

$$1 - \left(1 - \frac{0.8647 * 2^{45}}{2^{89}}\right)^{2^{46-44\Delta_c}} \approx 90\% \tag{8.16}$$

Оценка требуемых ресурсов для атаки. Для атаки требуется 2^{46} битов ключевой последовательности. Таблица размером размером - $2^{51.48}$ бит, вычислительная сложность ее построения - 2^{48} операции алгоритма DES.

Возможности практической реализации атаки. Вычислительная сложность в 2^{48} операции алгоритма DES более чем позволяет ее провести при условии, что есть необходимый объем ключевой последовательности. Но проблема в том, что собрать 2^{46} битов ключевой последовательности в большинстве случаев невозможно. Поэтому если эта атака и применима, то только в таких специфичных случаях, когда доступен "неограниченный"объем ключевой последовательности.

Пути нейтрализации атаки. С помощью увеличения длины атакуемого регистра можно существенно затруднить данную атаку, так как вырастет объем таблицы и количество требуемых опереаций.

8.3.3. Атака на фильтрующую функцию

В 2007 году китайские исследователи [9] смогли получить явный вид функции f_d шифра LILI128. Для этого им понадобилось около $2^{12} - 2^{13}$ битов ключевой последовательности. Реконструкция функции была проведена на IBM ноутбуке с помощью MATLAB за 1.6 часов.

Угроза и модель нарушителя. Данная атака направлена на фильтрующую функцию f_d шифра LILI128. Злоумышленник проводит реконструкцию функции f_d, что упрощает дальнейший криптоанализ.

Нарушитель имеет доступ к оракулу шифрования. Он сам его инициализирует, использует и далее анализирует полученную ключевую последовательность. Также он знает, что выход шифра - это выход какой-то фильтрующей функции f_d. То есть в их атаке неизвестной переменной является только функция f_d, хотя обычно целью атаки является начальное состояние. Поэтому модель нарушителя не совсем вписывается в общепринятые модели.

Описание атаки. Авторы не приводят точного алгоритма атаки, но описывают основные приемы, которые были применены. Первым шагом они взяли программную имплементацию шифра и проинициализировали ее значением "yyyyyyyyyyyyyyyy". Затем сгенерировали ключевую последовательность. И взяли из нее такой отрезок последовательности (начиная с первого бита), который бы выражал функцию f_d и был бы минимален по сложности (вероятно имеется в виду линейная сложность). Это они могли сделать с помощью знания того, от каких конкретно позиции состояния $LSFR_d$ берется функция f_d. Затем с помощью кластеризации и нелинейного прогноза они смогли восстановить диаграмму фазового пространства, состоящей из 10 измерений: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10), из ключевой последовательности, полученной на предыдущем шаге. Реконструированная в АНФ f_d содержит 46 слагаемых, как линейные так и нелинейные:

$$\begin{aligned}
&x_2 + x_3 + x_4 + x_5 + x_6x_7 + x_1x_8 + x_2x_8 + x_1x_9 + x_3x_9 \\
&\quad + x_4x_{10} + x_6x_{10} + x_3x_7x_9 + x_4x_7x_9 + x_6x_7x_9 + x_3x_8x_9 \\
&\quad + x_6x_8x_9 + x_4x_7x_{10} + x_5x_7x_{10} + x_6x_7x_{10} + x_3x_8x_{10} \\
&\quad + x_4x_8x_{10} + x_2x_9x_{10} + x_3x_9x_{10} + x_4x_9x_{10} + x_5x_9x_{10} \\
&\quad + x_3x_7x_8x_{10} + x_5x_7x_8x_{10} + x_2x_7x_9x_{10} + x_4x_7x_9x_{10} \\
&\quad + x_6x_7x_9x_{10} + x_1x_8x_9x_{10} + x_3x_8x_9x_{10} + x_4x_8x_9x_{10} \\
&\quad + x_6x_8x_9x_{10} + x_4x_6x_7x_9 + x_5x_6x_7x_9 + x_2x_7x_8x_9 \\
&\quad + x_4x_7x_8x_9 + x_4x_6x_7x_9x_{10} + x_5x_6x_7x_9x_{10} \\
&\quad + x_3x_7x_8x_9x_{10} + x_4x_7x_8x_9x_{10} + x_4x_6x_7x_8x_9 \\
&\quad + x_5x_6x_7x_8x_9 + x_4x_6x_7x_8x_9x_{10} + x_5x_6x_7x_8x_9x_{10}
\end{aligned} \tag{8.17}$$

Непосредственной проверкой можно проверить, что полученная функция и функция f_d, заданная таблицей в оригинальной статье совпадают.

Также авторы утверждают, что если видоизменить функцию f_d, задать начальное 128-битное состояние, сгенерировать ключевую последовательность длиной 2^{13} и отравить им, то они смогут ее реконструировать.

Это атака не дает возможности узнать начальное состояние. Но явный вид функции открывает новые вектора атаки на функцию с меньшей вычислительной сложностью, что позже и сделали. В 2012 году была придумана атака [10], которая существенно эффективнее, чем предыдущие. Она требует 2^{47} предварительных вычислений, которые создают таблицу размером 2^{47} 89-битных слов, ключевую последовательность длиной 2^{46} битов и 2^{35} активных вычислений.

Оценка требуемых ресурсов для атаки. Для атаки требуется 2^{13} битов ключевой последовательности. Количество требуемых операции не приведено, но реконструкция функции была проведена на IBM ноутбуке с помощью MATLAB за 1.6 часов.

Возможности практической реализации атаки. Атака реализована.

Пути нейтрализации атаки. Так как нет точного алгоритма атаки, то труднее понять, как от нее защититься. Авторы ищут явный функции, поэтому можно предположить, что если увеличить количество переменных, от которых зависит функция, то мы существенно увеличим количество требуемых операции для реконструкции.

8.4. Заключение

Шифр LILI128 имеет достаточно простую конструкцию, его можно эффективно реализовать как программно, так и аппаратно. Он имеет большой период, сбалансированную выходную ключевую последовательность и высокую линейную сложность. Но авторы явно ошиблись в оценке защищенности шифра, так как почти сразу после публикации шифра вышла быстрая корреляционная атака, которая имела сложность ниже, чем нижняя оценка сложности авторов. Также специалисты в ряде атак (например [8]) смогли обойти нерегуляруемое тактирование, которое тоже было добавлено как элемент усиливающий безопаность шифра. Наконец, китайские исследователи [9] смогли получить явный вид функции f_d, что открыло новые векторы атак и существенно понизило безопаность шифра. Учитывая количество уязвимостей генератора LILI128, тяжело назвать его безопасным. Он не является CPA-стойким, однако объем ключевой последовательности, которую необходимо собрать для атаки является достаточно большим.

9. Список литературы

[1] Clark A. et al. The LILI-II keystream generator // Information Security and Privacy: 7th Australasian Conference, ACISP 2002 Melbourne, Australia, July 3–5, 2002 Proceedings 7. – Springer Berlin Heidelberg, 2002. – С. 25-39.

[2] Dubrova E. How to speed-up your NLFSR-based stream cipher //2009 Design, Automation and Test in Europe Conference and Exhibition. – IEEE, 2009. – С. 878-881.

[3] Jönsson F., Johansson T. A fast correlation attack on LILI-128 //Information Processing Letters. – 2002. – Т. 81. – №. 3. – С. 127-132

[4] Siegenthaler. Decrypting a class of stream ciphers using ciphertext only //IEEE Transactions on computers. – 1985. – Т. 100. – №. 1. – С. 81-85.

[5] Chepyzhov V. V., Johansson T., Smeets B. A simple algorithm for fast correlation attacks on stream ciphers //Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10–12, 2000 Proceedings 7. – Springer Berlin Heidelberg, 2001. – С. 181-195.

[6] MacWilliams F. J., Sloane N. J. A. The theory of error-correcting codes. – Elsevier, 1977. – Т. 16.

[7] Babbage S. H. Improved"exhaustive search"attacks on stream ciphers //European Convention on Security and Detection, 1995. – IET, 1995. – С. 161-166.

[8] Saarinen M. J. O. A time-memory tradeoff attack against LILI-128 //Fast Software Encryption: 9th International Workshop, FSE 2002 Leuven, Belgium, February 4–6, 2002 Revised Papers 9. – Springer Berlin Heidelberg, 2002. – С. 231-236.

[9] Huang X. et al. Reconstructing the nonlinear filter function of LILI-128 stream cipher based on complexity //arXiv preprint cs/0702128. – 2007.

[10] Mihaljević M. J. et al. Internal state recovery of keystream generator LILI-128 based on a novel weakness of the employed Boolean function //Information Processing Letters. – 2012. – Т. 112. – №. 21. – С. 805-810.

[11] Баданин М.Ф. . - URL: https://commons.wikimedia.org/wiki/File:РСЛОС_в_А5.png (дата обращения: 28.12.2023)

[12] Golić J. D. Cryptanalysis of alleged A5 stream cipher //International Conference on the Theory and Applications of Cryptographic Techniques. – Berlin, Heidelberg : Springer Berlin Heidelberg, 1997. – С. 239-255.

[13] Biryukov A., Shamir A., Wagner D. Real time cryptanalysis of A5/1 on a PC //Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10–12, 2000 Proceedings 7. – Springer Berlin Heidelberg, 2001. – С. 1-18.

[14] Баданин М.Ф. . - URL: https://commons.wikimedia.org/wiki/File:РСЛОСвА52.png (дата обращения: 28.12.2023)

[15] Bogdanov A., Eisenbarth T., Rupp A. A hardware-assisted realtime attack on A5/2 without precomputations //Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9. – Springer Berlin Heidelberg, 2007. – С. 394-412.

[16] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, 3G Security, Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification, V3.1.1, 2001.

[17] Amirki . - URL: https://commons.wikimedia.org/wiki/File:Feistel_cipher_diagram_en.svg (дата обращения: 29.12.2023)

[18] Dunkelman O., Keller N., Shamir A. A practical-time attack on the A5/3 cryptosystem used in third generation GSM telephony //Cryptology ePrint Archive. – 2010.

[19] Sugio N., Igarashi Y., Hongo S. Integral Cryptanalysis on Reduced-Round KASUMI //IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences. – 2022. – Т. 105. – №. 9. – С. 1309-1316.

[20] Rogaway P., Coppersmith D. A software-optimized encryption algorithm //Fast Software Encryption: Cambridge Security Workshop Cambridge, UK, December 9–11, 1993 Proceedings 1. – Springer Berlin Heidelberg, 1994. – С. 56-63.

[21] Handschuh H., Gilbert H. 2 cryptanalysis of the SEAL encryption algorithm //International Workshop on Fast Software Encryption. – Berlin, Heidelberg : Springer Berlin Heidelberg, 1997. – С. 1-12.

[22] Watanabe D. et al. A new keystream generator MUGI //Fast Software Encryption: 9th International Workshop, FSE 2002 Leuven, Belgium, February 4–6, 2002 Revised Papers 9. – Springer Berlin Heidelberg, 2002. – С. 179-194.

[23] Daemen J., Clapp C. Fast hashing and stream encryption with PANAMA //International Workshop on Fast Software Encryption. – Berlin, Heidelberg : Springer Berlin Heidelberg, 1998. – С. 60-74.

[24] Sekine H. et al. A strength evaluation of a pseudorandom number generator MUGI against linear cryptanalysis //IEICE transactions on fundamentals of electronics, communications and computer sciences. – 2005. – Т. 88. – №. 1. – С. 16-24.

[25] Golić J. D. A weakness of the linear part of stream cipher MUGI //Fast Software Encryption: 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004. Revised Papers 11. – Springer Berlin Heidelberg, 2004. – С. 178-192.

[26] Biryukov A., Shamir A. Analysis of the non-linear part of MUGI //Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers 12. – Springer Berlin Heidelberg, 2005. – С. 320-329.

[27] ETSI/SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 and UIA2. Document 2: SNOW 3G Specification, version 1.1 (September 2006)

[28] ETSI/SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 UIA2. Document 5: Design and Evaluation Report, version 1.1 (September 2006)

[29] Klein A. Attacks on the RC4 stream cipher //Designs, codes and cryptography. – 2008. – Т. 48. – С. 269-286.

[30] Paul G., Rathi S., Maitra S. On non-negligible bias of the first output byte of RC4 towards the first three bytes of the secret key //Designs, Codes and Cryptography. – 2008. – Т. 49. – С. 123-134.

[31] Vanhoef M., Piessens F. All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS //24th USENIX Security Symposium (USENIX Security 15). – 2015. – С. 97-112.