

Министерство образования Российской Федерации  
Ярославский государственный университет им. П.Г. Демидова  
Кафедра компьютерных сетей

**Математические методы защиты  
информации**

Методические указания

Ярославль 2013

ББК В 311  
У 68  
УДК 517.2

Составители **М.В. Краснов**

**Математические методы защиты информации (ч. 3.):** Метод. указания / Сост. М.В. Краснов; Яросл. гос. ун-т. Ярославль, 2013. .

Основное использование вычислительной техники связано с хранением информации. Естественно, возникает задача защиты информации от несанкционированного использования. В работе сформулированы основные идеи создания поточных алгоритмов. Наиболее известные из них подробно описаны. Рассмотрена также проблема формирования хеш-значения, которая возникает при криптографических методах защиты информации.

Указания предназначены для помощи студентам при изучении курса «Математические методы защиты информации» и могут быть использованы как справочный материал при выполнении домашних заданий, курсовых работ и при подготовке к экзаменам.

**Рецензент:** кафедра компьютерных сетей Ярославского государственного университета им. П.Г. Демидова.

© Ярославский государственный университет им. П.Г. Демидова, 2013

© Краснов М.В. 2013

---

### **Математические методы защиты информации (ч. 3.)**

Составители: Краснов Михаил Владимирович

Корректор А.А. Антонова

Лицензия ЛР № 020319 от 30.12.96.

Подписано в печать 22.05.2000. Формат 60×84/16. Бумага тип.  
Усл. печ. л. 1,6. Уч.-изд. л.1,73. Тираж 50 экз. Заказ

Оригинал-макет подготовлен в редакционно-издательском отделе  
Ярославского государственного университета.

Отпечатано на ризографе

Ярославский государственный университет.  
150000, Ярославль, ул. Советская, 14.

## Введение

В настоящее время использование электронной вычислительной техники в различных областях человеческой деятельности все более и более возрастает. Однако чаще всего вычислительная техника используется для хранения и передачи информации. Естественно, возникает задача защиты информации от несанкционированного использования. Среди способов защиты информации одним из наиболее распространенных методов является криптографический метод. Он предусматривает такое преобразование информации, при котором она становится доступной для прочтения лишь обладателю некоторого секретного параметра (ключа).

Опишем задачу защиты информации с помощью криптографического метода. Отправитель хочет послать получателю по каналу, который не является безопасным, текст  $T$ . Взломщик хочет перехватить передаваемую информацию. Отправителю нужно так послать сообщение, чтобы взломщик не смог прочитать исходный текст  $T$  из перехваченного сообщения, а получатель мог бы за приемлемое время восстановить исходный текст из полученного сообщения.

Чтобы решить поставленную задачу, отправитель шифрует исходный текст  $T$  с помощью некоторого преобразования  $E_K$ , где  $K$  - ключ шифрования. Шифр-текст  $C = E_K(T)$  передается по каналу связи.

Получатель должен уметь расшифровать шифр-текст, то есть восстановить исходный текст  $T$  с помощью некоторого преобразования  $D_{\tilde{K}}$ , где  $\tilde{K}$  - ключ расшифрования; другими словами  $T = D_{\tilde{K}}(C)$ .

Простейшие способы шифрования появились очень давно, однако научный подход к исследованию и разработке криптографических методов появился только в прошлом (двадцатом) веке. К настоящему времени криптография содержит множество результатов (теорем, алгоритмов), как фундаментальных, так и прикладных. Занятие криптографией невозможно без серьезной математической подготовки. Вместе с тем не следует забывать, что криптографические методы предназначены в первую очередь для практического применения, а теоретически стойкие алгоритмы могут оказаться незащищенными перед атаками, не предусмотренными математической моделью.

Алгоритмы, используемые в современных криптосистемах, можно разделить на два типа:

- симметричные, в которых ключ расшифрования легко находится по ключу шифрования;
- с открытым ключом, в которых ключ расшифрования трудно найти даже при известном ключе шифрования.

С другой стороны симметричные шифры можно разделить на два типа шифров: блочные и поточные.

Как правило, поточные шифры оперируют с битами (или с байтами) открытого текста и шифротекста, а блочные – блоками фиксированной дли-

ны. Приведем несколько существенных различий между этими типами шифров:

- блочный криптоалгоритм в своей работе требует полного блока данных и следовательно процесс шифрования происходит с временной задержкой, а в поточных алгоритмах стараются обеспечить шифрование в режиме реального времени;
- в блочных шифрах для шифрования всех блоков используется один ключ, а в поточных – для каждой порции исходных данных используется свой ключ.

Напомним, что абсолютная стойкость криптосистемы объясняется отсутствием каких-либо закономерностей в зашифрованных данных. Противник, перехвативший шифр, не может на основе его анализа получить какую-либо информацию об исходном тексте. Это свойство достигается при выполнении трех требований:

- равенство длин ключа и исходного текста;
- случайность ключа;
- однократное использование ключа.

Основной недостаток абсолютно стойкой криптосистемы – это равенство объема ключевой информации и суммарного объема передаваемых сообщений. Следовательно, построить эффективную систему можно, лишь отказавшись от абсолютной стойкости и использовать в качестве ключевой псевдослучайную последовательность.

Читатель не должен забывать, что простейшая модель поточного шифра может быть сформулирована следующим образом:

- процесс шифрования:
  - на источнике генерируется случайная ключевая последовательность (гамма) бит  $k = k_1 k_2 \dots k_m \dots$ ;
  - ключевая последовательность складывается по модулю два с битами исходного текста  $t = t_1 t_2 \dots t_m \dots$

$$c_i = t_i \oplus k_i.$$

- процесс расшифрования:
  - на приемнике повторно генерируется ключевая последовательность;
  - ключевая последовательность складывается по модулю два с зашифрованными данными.

$$t_i = c_i \oplus k_i.$$

Студенты должны обратить особое внимание на тот факт, что стойкость системы целиком зависит от внутренней структуры генератора ключевой последовательности. Заметим, что если каждый раз при создании ключевой последовательности будет выдаваться одна и та же последовательность, то взлом криптосистемы будет тривиальной задачей. Легко сформулировать

требования, которым должен удовлетворять генератор псевдослучайной последовательности, ориентированный на использование в системах поточного шифрования:

- криптографическая стойкость (вычисление числа  $k_{i+1}$  по известным предыдущим элементам последовательности  $k_i$  без знания ключа должно быть трудной задачей);
- статистическая безопасность (вероятности порождения различных значений должны быть в точности равны);
- большой период формируемой последовательности; □
- эффективная аппаратная и программная реализация.

Данные методические указания содержат начальные сведения по созданию поточных шифров.

## **Поточные шифры**

Поточные шифры можно разделить на два типа: синхронные и самосинхронизирующиеся. Опишем идеи указанных шифров:

- синхронные шифры – в этом случае значения ключевой последовательности не зависят от входного или зашифрованного текстов. Легко заметить, что у данного типа шифров отсутствует эффект размножения ошибки. Один бит шифра искаженный при передаче приведет к искажению только одного бита текста при расшифровании. Вставка или выпадения бит зашифрованной последовательности  $c_i$  недопустимы, так как нарушится синхронизация, что приведет к неправильному расшифрованию всех последующих бит.
- самосинхронизирующиеся шифры - в этом случае элементы входной последовательности зашифровываются с учетом  $n$  предшествующих битов зашифрованного текста, которые принимают участие в формировании ключевой последовательности. Легко заметить, что у данного типа шифров присутствует эффект размножения ошибки. Заметим, что восстановление синхронизации в случае необходимости произойдет автоматически через  $n$  элементов зашифрованного текста.

## **Генераторы псевдослучайных последовательностей**

Генераторы псевдослучайных последовательностей являются важными элементами систем защиты информации, например, они используются для решения следующих задач:

- генерации псевдослучайных последовательностей при построении синхронных и самосинхронизирующихся поточных шифров;
- хеширования информации.

Рассматриваемые в данной работе генераторы псевдослучайных последовательностей можно разделить на криптографические и некриптографические:

- некриптографические генераторы: конгруэнтные, на регистрах сдвига с линейной обратной связью, аддитивные;
- криптографические генераторы: с использованием односторонних функций, с использованием функции  $E_k$  блочных шифров, с использованием блоков стохастического преобразования, с использованием функции  $E_k$  поточных шифров.

Студенты не должны забывать, что при использовании криптостойкого генератора три следующие задачи для противника должны быть вычислительно неразрешимы:

- определение  $(i-1)$ -го элемента  $k_{i-1}$  последовательности на основе известного фрагмента гаммы  $k_i k_{i+1} \dots k_{i+b-1}$  конечной длины  $b$ ;
- определение  $(i+1)$ -го элемента  $k_{i+1}$  последовательности на основе известного фрагмента гаммы  $k_{i-b+1} \dots k_{i-1} k_i$  конечной длины  $b$ ;
- определение ключевой информации по известному фрагменту гаммы конечной длины.

### Конгруэнтные генераторы

Линейный конгруэнтный генератор является одним из простейших генераторов с хорошо известными свойствами. Формула, по которой вычисляется очередное значение последовательности, имеет вид

$$X_{i+1} = (aX_i + b) \bmod n,$$

где  $a, b, n$  — некоторые константы ( $n > 0$ ,  $0 < a < n$ ,  $0 \leq b < n$ ), а  $X_i$  — предыдущее псевдослучайное число. Ключом служит значение  $X_0$ . Легко заметить, что максимальный период такой последовательности равен  $n$ .

*Утверждение.* Линейная конгруэнтная последовательность, определенная числами  $a, b, n$  и  $X_0$  имеет период длины  $n$  тогда и только тогда, когда:

- числа  $b$  и  $n$  — взаимно простые;
- числа  $a - 1$  кратно  $p$  для каждого простого  $p$ , являющегося делителем  $n$ ;
- $a - 1$  кратно 4, если  $n$  кратно 4. ■

Приведем несколько констант при которых линейные конгруэнтные генераторы обеспечивают максимальный период [1].

Переполняется при	$a$	$b$	$n$
$2^{20}$	106	1283	6075
$2^{21}$	211	1663	7875
$2^{22}$	421	1663	7875

$2^{23}$	430	2531	11979
$2^{24}$	171	11213	53125

Обобщением линейного конгруэнтного генератора являются полиномиальные конгруэнтные генераторы, определяемые формулой вида:

$$X_{i+1} = f(X_i) \bmod n,$$

где  $f(x)$ - произвольный полином, например:

$$X_{i+1} = (aX_i^2 + bX_i + c) \bmod n - \text{квадратичный генератор};$$

$$X_{i+1} = (aX_i^3 + bX_i^2 + cX_i + d) \bmod n - \text{кубический генератор}.$$

К сожалению, конгруэнтные генераторы бесполезны для криптографии, поскольку не обладают достаточной криптостойкостью.

### Регистр сдвига с линейный обратной связью (LFSR).

**Определение.** Регистр сдвига с обратной связью - это регистр, который можно рассматривать как множество ячеек памяти, в каждой из которых записан один бит информации. На каждом шаге содержимое нескольких заранее определенных ячеек (отводов) пропускается через функцию обратной связи. Значение функции записывается в самую левую ячейку регистра, сдвигая все остальные его биты на одну позицию вправо. Выходом регистра на текущем шаге является «вытолкнутый» справа бит (рис. 1). ■

**Определение.**  $N$ -битовый сдвиговый регистр – регистр сдвига с длиной  $N$  битов. ■

**Определение.** Периодом сдвигового регистра называется длина получаемой последовательности до начала ее повторения. ■

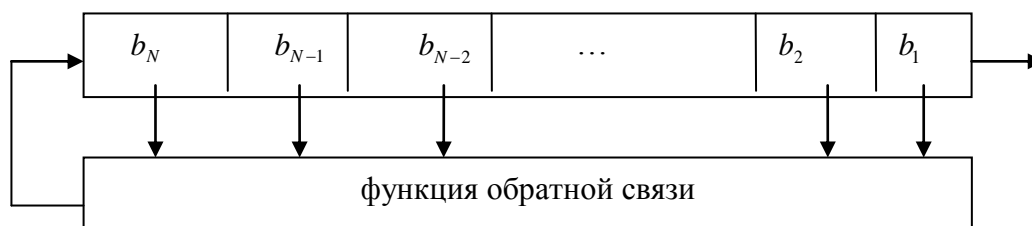


рисунок 1 Регистр сдвига с обратной связью

**Определение.** Регистр сдвига с линейный обратной связью - данный регистр устроен также как регистр сдвига с обратной связью, но в качестве функции берется логическая операция  $XOR$ . ■

Опишем конструкцию и работу LFSR:

- допустим, что в начальном состоянии регистра в его ячейках записана последовательность бит  $\{s_{N-1}, \dots, s_0\}$

- задается множество бит  $\{c_1, \dots, c_N\}$ , где

$$c_i = \begin{cases} 1 & \text{если соответствующая ячейка является отводом} \\ 0 & \text{в противном случае} \end{cases}$$

- на выходе регистра получается последовательность

$s_0, \dots, s_{N-1}, s_N, s_{N+1}, \dots$ , где

$$s_i = c_1 * s_{i-1} \oplus \dots \oplus c_N * s_{i-N} \text{ при } i \geq N.$$

Читатель легко может убедиться в том, что если в начальном состоянии регистра во всех его ячейках стоят нули, то на выходе мы получим последовательность из одних нулей. Нетрудно подсчитать, что для  $N$ -битового LFSR существует всего  $2^N - 1$  ненулевых начальных состояний регистра. Следовательно, максимальный период последовательности битов, которую будет генерировать регистр сдвига с линейной обратной связью, равен  $2^N - 1$ .

При изучении темы «Регистр сдвига с линейной обратной связью» студенты должны обратить особое внимание на то, что с LFSR можно ассоциировать двоичный многочлен  $C(x) = c_N x^N + c_{N-1} x^{N-1} + \dots + c_1 x + 1 \in F_2[x]$ . Степень многочлена является длиной сдвигового регистра, а его ненулевые коэффициенты являются отводами. Для того, чтобы конкретный LFSR имел максимальный период, ассоциированный многочлен должен быть примитивным<sup>1</sup>.

Приведем несколько примитивных многочленов[1].

степень	многочлен	степень	многочлен	степень	многочлен
1	$x + 1$	5	$x^5 + x^2 + 1$	9	$x^9 + x^4 + 1$
2	$x^2 + x + 1$	6	$x^6 + x + 1$	10	$x^{10} + x^3 + 1$
3	$x^3 + x + 1$	7	$x^7 + x^3 + 1$	11	$x^{11} + x^2 + 1$
4	$x^4 + x + 1$	8	$x^8 + x^4 + x^3 + x^2 + 1$	12	$x^{12} + x^6 + x^4 + x + 1$

В качестве иллюстрации работы LFSR рассмотрим следующий пример.

*Пример.* Построим регистр сдвига с линейной обратной связью с ассоциированным многочленом  $x^3 + x + 1$  и выпишем состояние регистра, если он был инициализирован вектором (111).

LFSR		Состояние регистра		Выход
		итерация	состояние	
		0	111	
		1	011	1
		2	101	1
		3	010	1
		4	001	0
		5	100	1
		6	110	0
		7	111	0

*Пример окончен.*

Основными достоинствами LFSR являются:

- простота аппаратной и программной реализации;
- максимальное быстродействие;

<sup>1</sup> Двоичный многочлен  $C(x)$  степени  $N$  называется примитивным, если он неприводим, а его корень  $\theta$  является образующей мультипликативной группы поля  $F_{2^N}$ .



- хорошие статистические свойства формируемых последовательностей.
- Определение.* Линейной сложностью бесконечной последовательности битов  $z = z_0, z_1, z_2, \dots$ , называется величина  $LC(z)$ , равная:
- 0, если  $z$  – последовательность нулей;
  - $\infty$ , если  $z$  нельзя получить с помощью какого-нибудь LFSR;
  - длине наименьшего LFSR, выдающего последовательность  $z$  в остальных случаях. ■

Студентам следует обратить внимание на тот факт, что хотя LFSR и является хорошим генератором псевдослучайных чисел, но использовать его для шифрования не рекомендуется. Легко заметить, что зная длину  $N$  регистра и  $2N$  последовательных бит, выходящих из LFSR, мы сможем найти ассоциированный многочлен. Предположим, что нам известны сгенерированные биты  $s_0, \dots, s_{N-1}, s_N, s_{N+1}, \dots, s_{2N-1}$ . Напомним, что функция обратной связи  $s_i = c_1 * s_{i-1} \oplus \dots \oplus c_N * s_{i-N}$  при  $i \geq N$ . Следовательно, можно выписать систему

$$\begin{cases} s_N = s_{N-1}c_1 \oplus s_{N-2}c_2 \oplus \dots \oplus s_0c_N \\ s_{N+1} = s_Nc_1 \oplus s_{N-1}c_2 \oplus \dots \oplus s_1c_N \\ \dots \\ s_{2N-1} = s_{2N-2}c_1 \oplus s_{2N-3}c_2 \oplus \dots \oplus s_{N-1}c_N \end{cases}$$

или в матричной форме

$$\begin{pmatrix} s_{N-1} & s_{N-2} & \dots & s_0 \\ s_N & s_{N-1} & \dots & s_1 \\ & & \dots & \\ s_{2N-2} & s_{2N-3} & \dots & s_{N-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_N \end{pmatrix} = \begin{pmatrix} s_N \\ s_{N+1} \\ \dots \\ s_{2N-1} \end{pmatrix} \pmod{2}.$$

*Пример.* Найдем 4-битовый регистр сдвига с линейной обратной связью, если нам известна последовательность битов, которая была им сгенерирована 01011110001.

Подставим в предыдущую систему уравнений элементы перехваченной последовательности.

$$\begin{cases} s_4 = s_3c_1 \oplus s_2c_2 \oplus s_1c_3 \oplus s_0c_4 \\ s_5 = s_4c_1 \oplus s_3c_2 \oplus s_2c_3 \oplus s_1c_4 \\ s_6 = s_5c_1 \oplus s_4c_2 \oplus s_3c_3 \oplus s_2c_4 \\ s_7 = s_6c_1 \oplus s_5c_2 \oplus s_4c_3 \oplus s_3c_4 \end{cases} \Rightarrow \begin{cases} 1 = c_1 \oplus c_3 \\ 1 = c_1 \oplus c_2 \oplus c_4 \\ 1 = c_1 \oplus c_2 \oplus c_3 \\ 0 = c_1 \oplus c_2 \oplus c_3 \oplus c_4 \end{cases} \Rightarrow \begin{cases} c_1 = 0 \\ c_2 = 0 \\ c_3 = 1 \\ c_4 = 1 \end{cases}$$

Ассоциированный с LFSR многочлен  $x^4 + x^3 + 1$ .

*Пример окончен.*

Тот факт, что на практике длина регистра  $N$  (предполагаем, что  $N = LC(z)$ ) не известна заранее, не намного усложняет задачу, так как можно по очереди проверять гипотезы  $LC(z) = 1, 2, \dots$  или воспользоваться алгоритмом Берлекэмпа-Мессии[5].

### Алгоритм Берлекэмпа-Месса.

Исходными данными алгоритма является: последовательность  $s$  длины  $N$ .

Алгоритм состоит из трех шагов:

шаг 1. Инициализация:

- Введем несколько переменных:  $L$  - длина регистра сдвига,  $r$  - номер итерации,  $\Delta_r$  - вспомогательная переменная.
- Введем несколько многочленов:  $C(x)$  - текущий регистр сдвига с линейной обратной связью,  $T(x)$  и  $B(x)$  - вспомогательные многочлены.
- $r = 0$ ,  $L = 0$ ,  $C(x) = 1$ ,  $B(x) = 1$ .

шаг 2. работа алгоритма, состоит из пяти пунктов:

1.  $r = r + 1$ ;  $\Delta_r = \sum_{j=0}^L C_j s_{r-j}$ ;
2. Если  $\Delta_r = 0$ , то переходим к пункту 4;  
в противном случае  $T(x) = C(x) - \Delta_r x B(x)$ ;
3. Если  $2L \leq r - 1$ , то  $B(x) = \Delta_r^{-1} C(x)$ ;  $C(x) = T(x)$ ;  $L = r - L$  и переходим к пункту 5;  
в противном случае  $C(x) = T(x)$ ;
4.  $B(x) = x B(x)$ ;
5. Если  $r = N$ , то завершаем шаг 2;  
в противном случае переходим к пункту 1.

шаг 3. Выход алгоритма:

- $L$  - длина регистра сдвига;
- $C(x)$  - регистр сдвига с линейной обратной связью.

В качестве иллюстрации работы алгоритма Берлекэмпа-Месса рассмотрим следующий пример.

*Пример.* Найдем регистр сдвига с линейной обратной связью, если нам известна последовательность битов, которая была им сгенерирована 01011110001<sup>2</sup>.

$r$	$\Delta_r$	$T(x)$	$B(x)$	$C(x)$	$L$	$r$	$\Delta_r$	$T(x)$	$B(x)$	$C(x)$	$L$
0			1	1	0	6	0	$1 + x^2 + x^3$	$(1 + x^2)x$	$1 + x^2 + x^3$	3
1	0	0	$x$	1	0	7	1	$1 + x^3 + x^4$	$1 + x^2 + x^3$	$1 + x^3 + x^4$	4
2	1	$1 + x^2$	1	$1 + x^2$	2	8	0	$1 + x^3 + x^4$	$(1 + x^2 + x^3)x$	$1 + x^3 + x^4$	4
3	0	$1 + x^2$	$x$	$1 + x^2$	2	9	0	$1 + x^3 + x^4$	$(1 + x^2 + x^3)x^2$	$1 + x^3 + x^4$	4
4	0	$1 + x^2$	$x^2$	$1 + x^2$	2	10	0	$1 + x^3 + x^4$	$(1 + x^2 + x^3)x^3$	$1 + x^3 + x^4$	4
5	1	$1 + x^2 + x^3$	$1 + x^2$	$1 + x^2 + x^3$	3	11	0	$1 + x^3 + x^4$	$(1 + x^2 + x^3)x^4$	$1 + x^3 + x^4$	4

<sup>2</sup> Вычисления будут проходить в поле  $GF(2)$ .

Результат  $L = 4$ ,  $C(x) = 1 + x^3 + x^4$

Пример окончен.

Заметим, что общий вид генератора двоичных последовательностей:

$$Q(t+1) = VQ(t),$$

где  $Q(t+1), Q(t)$  состояние регистра в соответствующие моменты времени;

$V$  квадратная матрица порядка  $N$  и  $V = T_i^k, i \in \{1, 2\}$

$$T_1 = \begin{pmatrix} c_1 & c_2 & \dots & c_{N-1} & c_N \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ & & \dots & & \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix} \text{ или } T_2 = \begin{pmatrix} 0 & 0 & \dots & 0 & c_N \\ 1 & 0 & \dots & 0 & c_{N-1} \\ & & \dots & & \\ 0 & \dots & 1 & 0 & c_2 \\ 0 & 0 & \dots & 1 & c_1 \end{pmatrix};$$

$N$  – степень образующего примитивного многочлена

$C(x) = c_N x^N + c_{N-1} x^{N-1} + \dots + c_1 x + 1$ , где  $c_N = 1, c_j \in \{0, 1\}, j \in \{1, \dots, N-1\}$ ;

$k$  – натуральное число (мы будем рассматривать только  $k = 1$ ).

**Утверждение.** Формируемая последовательность имеет максимальный период  $S = 2^N - 1$  тогда и только тогда, когда  $S$  и  $k$  взаимно просты. ■

Каждая матрица  $V$  имеет характеристический многочлен  $\phi(x)$ , которым является определитель матрицы  $V - xE$ , другими словами  $\phi(x) = |V - xE|$ , где  $E$  – единичная матрица. Многочлен  $C(x)$  определяет структуру генератора,  $\phi(x)$  определяет свойства генератора. Многочлены  $C(x)$  и  $\phi(x)$  связаны между собой следующим соотношением:  $\phi(x) = C(x^{-1})x^N$ . Примитивность  $\phi(x)$  означает примитивность  $C(x)$ , и наоборот. При  $k = 1$  генератор имеет вид, показанный на рис.2.

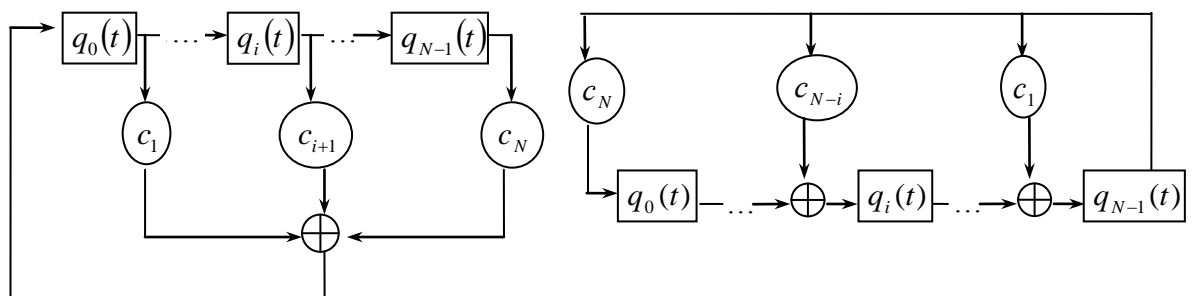


Схема Фибоначчи

$T=T_1$

Рисунок 2 Схема LFSR

Схема Галуа

$T=T_2$

### Применение LFSR в шифровании

Описанные выше регистры сдвига с линейной обратной связью несмотря на достаточно большой период и хорошие статистические качества имеют очень простое строение.

Студентам следует обратить внимание на тот факт, что в криптографических приложениях используют различные способы усложнения аналитического строения генераторов на основе LFSR.

Приведем несколько схем генераторов на основе LFSR:

1. Используются  $d$  регистров с линейной обратной связью (обычно берутся регистры с различными длинами и различными многочленами обратной связи), которые генерируют вектор  $(x_1^i, \dots, x_d^i)$ . Бит выхода схемы представляет собой применение функции, желательно нелинейной к вектору  $(x_1^i, \dots, x_d^i)$  (рис. 3). Эта функция называется *комбинирующей*, а генератор в целом – *комбинирующим генератором*.



Рисунок 3 комбинирующий генератор

Примером нелинейной комбинирующей функции для трех LFSR может служить  $f(x_1, x_2, x_3) = x_3 \oplus x_1 x_2 \oplus x_2 x_3$  (генератор Джиффи). Ключом комбинированного генератора служит начальное положение всех регистров LFSR[2].

2. Другой метод использования LFSR, представляет собой композицию линейных регистров сдвига. Так называется схема, в котором выход одного из регистров подается на вход другого (рис. 4).

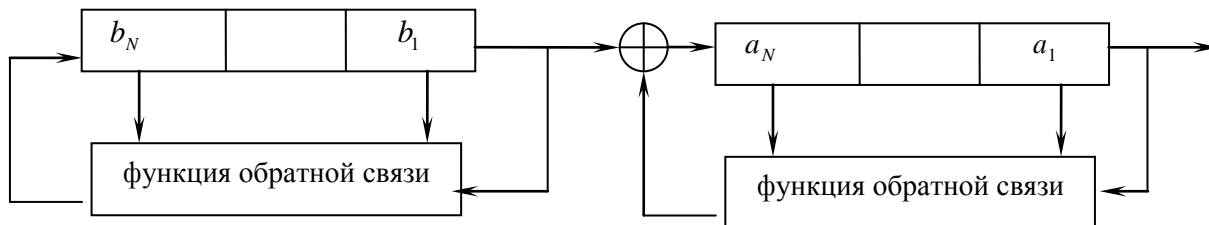


Рисунок 4. Композиция LFSR

3. Другой способ усложнения LFSR состоит в изменении закона рекурсии в процессе работы алгоритма генерации гаммы. Например, можно рассмотреть псевдослучайную последовательность, получаемую с помощью линейного регистра сдвига, закон функционирования которого меняется в зависимости от номера вырабатываемого бита. Рассмотрим два многочлена  $A(x) = x^m + \sum a_j x^j$ ,  $B(x) = x^m + \sum b_k x^k$  и последовательность  $v$ , которая имеет вид

$$v(2t + m) = -\sum_{j=0}^{m-1} a_j v(2t + j) \quad \text{и} \quad v(2t + 1 + m) = -\sum_{k=0}^{m-1} b_k v(2t + 1 + k).$$

Легко заметить, что в четных тактах закон рекурсии последовательности  $v$  определяется характеристическим многочленом  $A(x)$ , а в нечетных характеристическим многочленом  $B(x)$

4. Сжимающий генератор. Используется 2 регистра с линейной обратной связью. Тактовые импульсы поступают на оба LFSR. Предположим, что  $c = c_0 c_1 c_2 \dots$  - последовательность с выхода LFSR1,  $b = b_0 b_1 b_2 \dots$  последовательность с выхода LFSR2.

Тогда результирующая последовательность  $z = z_0 z_1 z_2 \dots$  включает в себя те биты  $b_i$ , для которых соответствующие биты  $c_i = 1$ . Остальные биты последовательности  $b$  игнорируются.

Студентам следует обратить внимание на тот факт, что при использовании в генераторах нелинейной функции можно выделить два подхода:

- использование нелинейной функции, как функцию обратной связи;
- использование двухступенчатой структуры. Задача первой ступени заключается в создании последовательности максимально большого периода. Задача второй ступени заключается в обеспечении криптостойкости (в качестве нелинейной функции можно использовать функцию шифрования  $E_k$  блочного шифра). Именно по такому принципу строятся генераторы комбинированного типа.

### Аддитивные генераторы

Аддитивные генераторы очень эффективны, их результатом являются случайные слова, а не биты.

Студентам следует обратить внимание на тот факт, что эти генераторы не обладают достаточной криптостойкостью, но они могут использоваться в качестве составных блоков для других генераторов.

Генератор состоит из  $m$  регистров разрядностью  $n$  – бит каждый и сумматора по модулю  $2^n$ . Начальное состояние генератора представляет собой массив  $n$  – битовых слов:  $X_0^0, X_1^0, \dots, X_m^0$ . Начальное заполнение выбирается таким образом, чтобы хотя бы в одном из регистров младший бит содержал "1".

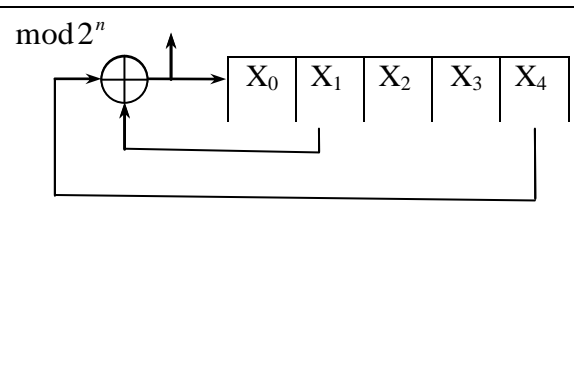
Уравнения работы генератора имеют вид:

$$X_0^{k+1} = \sum_{i=1}^m c_i X_{i-1}^k \mod 2^n;$$

$$X_i^{k+1} = X_{i-1}^k, \text{ для всех } i \in \{1, \dots, m-1\}.$$

где  $X_i^k$  - состояние  $i$  – го регистра в  $k$  – ый такт, а  $c_i$  - коэффициенты многочлена  $C(x)$  степени  $m$ , примитивного над  $GF(2)$ . Выходом генератора является  $X_0^{k+1}$ .

*Пример.* Построим аддитивный генератор, ассоциированный с многочленом  $x^5 + x^2 + 1$  и  $n = 4$ . Тогда генератор будет формировать рекуррентную последовательность в соответствии с формулой  $Y_i = (Y_{i-5} + Y_{i-2}) \mod 2^4$ . Предположим, что начальное состояние генератора задается вектором  $(0, 1, 2, 3, 4)$ .

			<table border="1"> <thead> <tr> <th>такт</th><th>выход</th><th>заполнение регистров</th><th>такт</th><th>выход</th><th>заполнение регистров</th></tr> </thead> <tbody> <tr><td>1</td><td>5</td><td>(5,0,1,2,3)</td><td>7</td><td>10</td><td>(10,9,7,4,7)</td></tr> <tr><td>2</td><td>3</td><td>(3,5,0,1,2)</td><td>8</td><td>0</td><td>(0,10,9,7,4)</td></tr> <tr><td>3</td><td>7</td><td>(7,3,5,0,1)</td><td>9</td><td>14</td><td>(14,0,10,9,7)</td></tr> <tr><td>4</td><td>4</td><td>(4,7,3,5,0)</td><td>10</td><td>7</td><td>(7,14,0,10,9)</td></tr> <tr><td>5</td><td>7</td><td>(7,4,7,3,5)</td><td>11</td><td>7</td><td>(7,7,14,0,10)</td></tr> <tr><td>6</td><td>9</td><td>(9,7,4,7,3)</td><td>12</td><td>1</td><td>(1,7,7,14,0)</td></tr> </tbody> </table>			такт	выход	заполнение регистров	такт	выход	заполнение регистров	1	5	(5,0,1,2,3)	7	10	(10,9,7,4,7)	2	3	(3,5,0,1,2)	8	0	(0,10,9,7,4)	3	7	(7,3,5,0,1)	9	14	(14,0,10,9,7)	4	4	(4,7,3,5,0)	10	7	(7,14,0,10,9)	5	7	(7,4,7,3,5)	11	7	(7,7,14,0,10)	6	9	(9,7,4,7,3)	12	1	(1,7,7,14,0)
такт	выход	заполнение регистров	такт	выход	заполнение регистров																																										
1	5	(5,0,1,2,3)	7	10	(10,9,7,4,7)																																										
2	3	(3,5,0,1,2)	8	0	(0,10,9,7,4)																																										
3	7	(7,3,5,0,1)	9	14	(14,0,10,9,7)																																										
4	4	(4,7,3,5,0)	10	7	(7,14,0,10,9)																																										
5	7	(7,4,7,3,5)	11	7	(7,7,14,0,10)																																										
6	9	(9,7,4,7,3)	12	1	(1,7,7,14,0)																																										
Схема аддитивного генератора			Пример работы аддитивного генератора																																												

При начальном состоянии (0,1,2,3,4) устройство на выходе формирует последовательность (5,3,7,4,7,9,10,0,14,7,7,1,...), которая удовлетворяет формуле  $Y_i = (Y_{i-5} + Y_{i-2}) \bmod 2^4$

*Пример окончен.*

Следовательно, можно дать другое описание аддитивного генератора:

- начальное состояние генератора представляет собой массив  $n$  –битовых слов:  $Y_1, Y_2, \dots, Y_m$ . Это первоначальное состояние и является ключом.
- формула, по которой вычисляется  $i$ -ое слово последовательности, имеет вид  $Y_i = (Y_{i-a} + Y_{i-b} + \dots + Y_{i-m}) \bmod 2^n$ .

## Генераторы, построенные с использованием односторонних функций.

Криптографические стойкие генераторы могут быть построены на основе так называемых односторонних функций.

*Определение.* Односторонней функцией называется функция  $F: X \rightarrow Y$ , обладающая двумя свойствами:

- существует полиномиальный алгоритм вычисления  $y = F(x)$ ;
- не существует полиномиальный алгоритма  $x = F^{-1}(y)$ . ■

До сих пор ни для одной функции, кандидата на звание односторонней, не доказано свойство 2.

Примером кандидата на звание односторонней функции может служить модульное возведение в степень, другими словами рассматривается функция  $F(x) = g^x \bmod p$ , где  $p$  – большое простое число,  $g$  – примитивный элемент поля  $Z_p^3$ .

Односторонняя функция в качестве функции шифрования неприемлема, так как, хотя  $F(x)$  – надежно зашифрованное сообщение для  $x$ , но получатель не сможет восстановить  $x$ . Обойти эту проблему можно с помощью односторонней функции с секретом.

<sup>3</sup> На этой идеи построена криптосистема с открытым ключом Эль-Гамаль

*Определение.* Односторонней функцией с секретом  $k$ , называется функция  $F_k : X \rightarrow Y$ , зависящая от параметра  $k$  и обладающая тремя свойствами:

- существует полиномиальный алгоритм вычисления  $y = F_k(x)$  при любом  $k$ ;
- при неизвестном  $k$  не существует полиномиального алгоритма  $x = F_k^{-1}(y)$ .
- при известном  $k$  существует полиномиального алгоритма  $x = F_k^{-1}(y)$ . ■

Студенты не должны забывать, что на идеях односторонних функций с секретом построено большинство асимметричных криптосистем (систем с открытым ключом).

Приведем в качестве примера два криптографических генератора: BBS-генератор (используется односторонняя функция с секретом  $F(x) = x^2 \bmod n$ ), RSA-генератор (используется односторонняя функция с секретом  $F(x) = x^e \bmod n$ ).

### **BBS-генератор**

BBS-генератор строится следующим образом:

- 1) вначале выбираются  $p$  и  $q$  - два больших простых числа примерно одинакового размера, причем  $p \equiv 3 \bmod 4$  и  $q \equiv 3 \bmod 4$ . Напомним, что  $p \equiv 3 \bmod 4$  означает, что существует целое число  $k$ , для которого выполняется равенство  $p = 3 + 4k$ ;
- 2) вычисляем число  $n = pq$ , которое называется целым числом Блюма;
- 3) выбираем случайное целое число  $x$ , такое что  $(x, n) = 1$ , то есть числа  $x$  и  $n$  являются взаимно простыми<sup>4</sup>;
- 4) вычисляем число  $x_0 = x^2 \bmod n$ , которое называется стартовым числом генератора;
- 5) искомой последовательностью бит длиной  $m$  будет являться последовательность

$$BBS_{n,m}(x_0) = b_0 b_1 b_2 \dots b_i \dots b_{m-1}, \quad i = 0, \dots, m-1,$$

где  $b_i$  - младший бит числа  $x_i$ ,

$$x_{i+1} = x_i^2 \bmod n.$$

*Пример.* Построим 10 битную псевдослучайную последовательность с помощью BBS-генератора.

- 1) Пусть  $p = 11$ ,  $q = 19$  (легко убедиться, что  $11 \equiv 3 \bmod 4$ ,  $19 \equiv 3 \bmod 4$ ).
- 2) Целое число Блюма  $n$  равно 209.

---

<sup>4</sup> Числа  $(a, b) = 1$ , если они не имеют общих делителей, кроме единицы.

- 3) Пусть  $x = 2$  (легко убедиться, что  $(2, 209) = 1$ ).
- 4) Стартовое число генератора  $x_0 = x^2 \bmod n \Rightarrow x_0 = 2^2 \bmod 209 \Rightarrow x_0 = 4$ .
- 5) В качестве элементов псевдослучайной последовательности будем брать младший бит в двоичной записи чисел  $x_{i+1} = x_i^2 \bmod n$ :

число	младший бит	число	младший бит
$x_0 = 4$	0	$x_5 = 158^2 \bmod 209 = 93$	1
$x_1 = 4^2 \bmod 209 = 16$	0	$x_6 = 93^2 \bmod 209 = 80$	0
$x_2 = 16^2 \bmod 209 = 47$	1	$x_7 = 80^2 \bmod 209 = 130$	0
$x_3 = 47^2 \bmod 209 = 119$	1	$x_8 = 130^2 \bmod 209 = 180$	0
$x_4 = 119^2 \bmod 209 = 158$	0	$x_9 = 180^2 \bmod 209 = 5$	1

В результате получили последовательность  $BBS_{209,10}(4) = 0011010001$ .

*Пример окончен.*

Студенты могут легко убедиться, что BBS-генератор допускает прямое определение отдельных бит, которые в нем вырабатываются  $x_i = x_0^{2^i} \bmod n$ . Поскольку  $p$  и  $q$  - простые числа и учитывая теорему Эйлера<sup>5</sup>, то

$$x_i = x_0^{2^i \bmod (p-1)(q-1)} \bmod n.$$

Безопасность этого генератора основана на сложности разложения  $n$  на множители. Отметим, что генератор BBS непредсказуем в левом и правом направлении. Это означает, что получив последовательность выданную генератором, криптоаналитик не сможет предсказать ни следующий, ни предыдущий бит последовательности.

### ***RSA-генератор.***

RSA-генератор строится следующим образом:

- 1) выбираем  $p$  и  $q$  - два больших простых числа примерно одинакового размера  $p \neq q$ ;
- 2) вычисляем число  $n = pq$  и число  $\varphi(n) = (p-1)(q-1)$ ;
- 3) выбираем случайное целое число  $e$ , которое являются взаимно простыми с  $\varphi(n)$ ;
- 4) выбираем в качестве стартового числа генератора случайное целое число  $x_0$  ( $1 < x_0 < n$ );
- 5) искомой последовательностью бит длиной  $m$  будет являться последовательность

$$RSA_{n,m}(x_0) = b_0 b_1 b_2 \dots b_i \dots b_{m-1}, \quad i = 0, \dots, m-1,$$

где  $b_i$  - младший бит числа  $x_i$ ,

$$x_{i+1} = x_i^e \bmod n.$$

<sup>5</sup> Если  $(a, n) = 1$ , то  $a^{\varphi(n)} \equiv 1 \bmod n$ ; где  $\varphi(n)$  - функция Эйлера.



*Пример.* Построим 10 битную псевдослучайную последовательность с помощью RSA-генератора.

- 1) Пусть  $p = 3$ , а  $q = 11$ .
- 2) Вычислим  $n = pq = 33$ . Вычислим функцию Эйлера  $\varphi(n) = (p-1)(q-1) = 20$ .
- 3) В качестве  $e$  возьмем число 3.
- 4) В качестве стартового числа генератора  $x_0 = 14$ .
- 5) В качестве элементов псевдослучайной последовательности будем брать младший бит в двоичной записи чисел  $x_{i+1} = x_i^e \bmod n$ :

число	младший бит	число	младший бит
$x_0 = 14$	0	$x_5 = 14^3 \bmod 33 = 5$	1
$x_1 = 14^3 \bmod 33 = 5$	1	$x_6 = 5^3 \bmod 33 = 26$	0
$x_2 = 5^3 \bmod 33 = 26$	0	$x_7 = 26^3 \bmod 33 = 20$	0
$x_3 = 26^3 \bmod 33 = 20$	0	$x_8 = 20^3 \bmod 33 = 14$	0
$x_4 = 20^3 \bmod 33 = 14$	0	$x_9 = 14^3 \bmod 33 = 5$	1

В результате получили последовательность  $RSA_{33,10}(14) = 0100010001$ .

*Пример окончен.*

### Генераторы, построенные с использованием блочных шифров

Для построения криптографически стойких генераторов можно использовать режимы OFB, CTR и CFB блочных шифров (например DES или AES). Будем считать, что размер блока шифра равен  $n$  бит.

- режим OFB (рис 5). В этом режиме блочный шифр на основе секретного ключа  $K$  и некоторого вектора инициализации  $Y_0$  формирует псевдослучайную последовательность  $r$  – битовых чисел  $z_1, z_2, \dots, z_k$ , которая может использоваться в качестве генератора. Псевдослучайная последовательность можно описать формулами:

$$Y_i = E_K(Y_{i-1}),$$

$$z_i = r \text{ старших бит } Y_i, \quad 1 \leq i \leq k$$

где  $E_K$  – алгоритм шифрования с ключом  $K$ , а  $1 \leq r \leq n$ .

Для шифрования каждого отдельного сообщения необходимо использовать различные  $K$  и  $Y_0$ . В противном случае будет получаться одна и та же псевдослучайная последовательность.

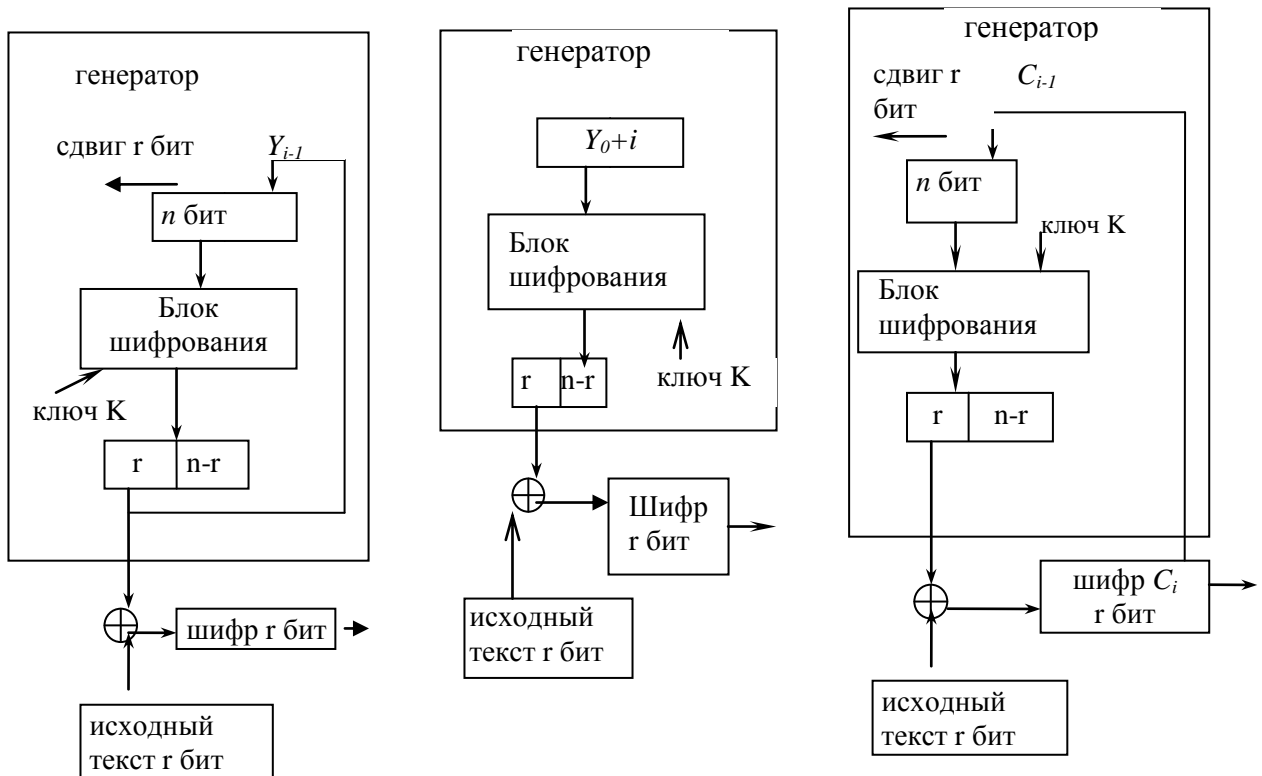
- режим CTR (рис. 5). Этот режим очень похож на OFB, но в нем для работы генератора используется просто счетчик увеличиваемый на каждом шаге на постоянное число. Псевдослучайная последовательность вычисляется по формуле:

$$z_i = r \text{ старших бит } E_K(Y_0 + i), \quad i = 1, 2, 3, \dots$$

где  $E_K$  – алгоритм шифрования с ключом  $K$ , а  $1 \leq r \leq n$ .

При использовании «идеального» блочного шифра режим CRT обеспечивает те же параметры стойкости, что и OFB. Преимущество режима CTR состоит в том, что любой элемент псевдослучайной последовательности может быть вычислен непосредственно.

- режим CFB(рис. 5). Этот режим очень похож на OFB, но в нем ключевой поток зависит от зашифрованного текста.



Использование блочного шифра в режиме OFB в качестве поточного шифра

Использование блочного шифра в режиме CTR в качестве поточного шифра

Использование блочного шифра в режиме CFB в качестве поточного шифра

**Рисунок 5 Режимы блочных шифров.**

## Генераторы, построенные с использованием блоков стохастического преобразования

Стохастическими методами защиты в широком смысле принято называть методы защиты информации, прямо или косвенно основанные на использовании генераторов псевдослучайных последовательностей (ПСП). Термин «стохастические методы защиты» применяется и в узком смысле, когда речь идет об алгоритмах, предполагающих использование стохастических сумматоров, то есть сумматоров с непредсказуемым результатом работы, зависящим от заполнения ключевой таблицы.

Схему одного из возможных вариантов построения  $R$ -блока стохастического преобразования рассмотрим по работе [6].

Ключевая информация  $R$ -блока – это заполненная таблица

$$H = \{H(m)\}, \quad m = \overline{0, (2^n - 1)},$$

размерности  $n \times 2^n$ , содержащей элементы  $GF(2^n)$ ,<sup>6</sup> перемешанные случайным образом, то есть  $H(m) \in GF(2^n)$ . Результат  $R_H(A, B)$  преобразования входного  $n$ -разрядного двоичного набора  $A$  зависит от заполнения таблицы  $H$ , параметра  $B$  и вычисляется по формуле

$$R_H(A, B) = H((m_A + B) \bmod 2^n),$$

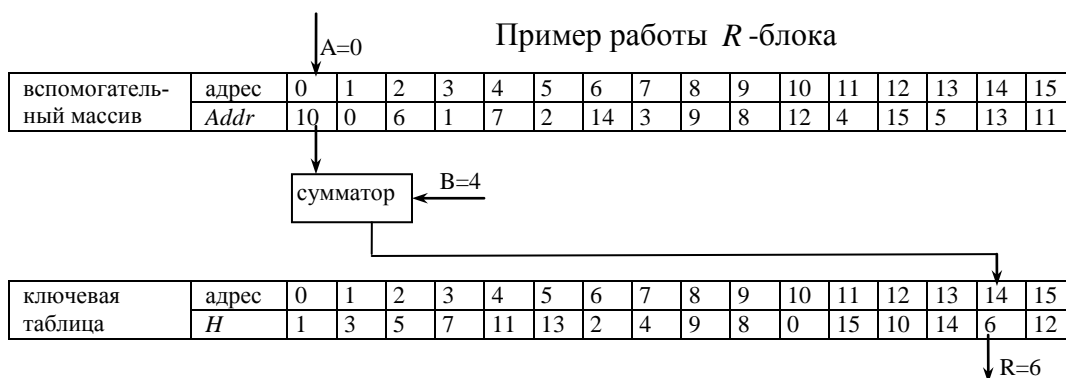
где  $m_A$  - адрес ячейки таблицы  $H$ , содержащей код  $A$ , то есть  $H(m_A) = A$ .

Другими словами результат работы  $R$ -блока содержимое ячейки таблицы  $H$ , которая получается при циклическом смещении на  $B$  позиций в сторону старших адресов относительно ячейки, содержащей код  $A$ . Для ускорения преобразования в состав  $R$ -блока можно ввести вспомогательный адресный массив

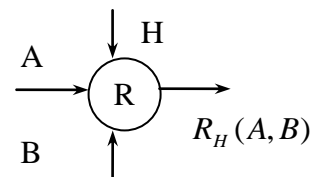
$$Addr = \{Addr(i)\}$$

размерности  $n \times 2^n$ , причем

$$\forall i = \overline{0, (2^n - 1)} \quad Addr(i) = m_i.$$



Условное графическое обозначение  $R$ -блока



Приведем несколько из возможных алгоритмов формирования таблицы  $H$ .

Первый вариант	Второй вариант
<p>Исходными данными алгоритма является инициализирующая последовательность <math>S_0, S_1, \dots</math>). Алгоритм состоит из нескольких шагов:</p> <ol style="list-style-type: none"> <li>1. инициализация индексов <math>i, j: j = 0; i = 0</math>;</li> <li>2. чтение элемента <math>S_i</math>;</li> </ol>	<p>Исходными данными алгоритма является инициализирующая последовательность <math>(x_0, x_1, \dots)</math>. Алгоритм состоит из нескольких шагов:</p> <ol style="list-style-type: none"> <li>1. запись в каждую ячейку таблицы <math>H</math> ее собственного адреса:  <math>H(0) = 0, H(1) = 1, \dots, H(2^n - 1) = 2^n - 1</math>;</li> </ol>

<sup>6</sup>  $GF(2^n)$  конечное поле содержащее  $2^n$  элементов.

3. если $S_i$ уже есть в таблице, то переходим на шаг 7; 4. записываем $S_i$ в таблицу $H$ : $H(j) = S_i$ ; 5. если $j$ равен $2^n - 1$ , то завершаем работу алгоритма; 6. увеличиваем индекс $j$ : $j = j + 1$ ; 7. увеличиваем индекс $i$ : $i = i + 1$ и переходим к шагу 2. На выходе получаем таблицу $H$ .	2. заполнение словами ключа массива $X$ из $2^n$ слов при этом ключ может повторяться необходимое число раз для заполнения всего массива: $X = \{x_i\}$ ; 3. инициализация индекса $j: j = 0$ ; 4. перемешиваем таблицу замен $H$ , для чего выполняем следующие действия for $i = 0$ to $2^n - 1$ $j = (j + X(i) + H(i)) \bmod 2^n$ меняются местами: $H(i) \leftrightarrow H(j)$ . На выходе получаем таблицу $H$ .
--	---

Студенты должны обратить внимание на то, что криптостойкость поточных шифров, использующих в своей работе блоки сложения по модулю  $2^n$ , можно легко значительно увеличить простой их заменой соответственно на стохастические сумматоры ( $R$ -блоки). В качестве иллюстрации рассмотрим аддитивный генератор ассоциированный с многочленом  $x^4 + x^3 + 1$ .

*Пример.* Построим аддитивный генератор, ассоциированный с многочленом  $x^4 + x^3 + 1$  и  $n = 4$ . Предположим, что начальным состоянием генератора является массив (12,2,0,11).

	такт	выход	заполнение регистров	такт	выход	заполнение регистров
	1	11	(11,12,2,0)	5	13	(13,7,14,2)
	2	2	(2,11,12,2)	6	0	(0,13,7,14)
	3	14	(14,2,11,12)	7	5	(5,0,13,7)
	4	7	(7,14,2,11)	8	4	(4,5,0,13)
Схема аддитивного генератора			Пример работы аддитивного генератора			

Получили на выходе генератора последовательность (11,2,14,7,13,0,5,4,...), которая удовлетворяет формуле  $Y_i = (Y_{i-4} + Y_{i-3}) \bmod 2^4$ .

Построим стохастический генератор, ассоциированный с многочленом  $x^4 + x^3 + 1$  и с ключевой таблицей  $H$ .

ключевая	адрес	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
таблица	$H$	2	1	0	12	5	3	13	10	5	14	8	7	11	9	15	4

Предположим, что начальным состоянием генератора является массив (12,2,0,11).

		такт	выход	заполнение регистров	такт	выход	заполнение регистров
		1	9	(9,12,2,0)	5	14	(14,14,3,2)
		2	2	(2,9,12,2)	6	10	(10,14,14,3)
		3	3	(3,2,9,12)	7	11	(11,10,14,14)
		4	14	(14,3,2,9)	8	10	(10,11,10,14)
Схема генератора		Пример работы генератора					

Получили на выходе генератора последовательность (9,2,3,14,14,10,11,10,...), которая удовлетворяет формуле  $Y_i = R_H(Y_{i-3}, Y_{i-4})$ .  
*Пример окончен.*

## Примеры поточных шифров

### Шифр RC4

RC4- это потоковый шифр с переменным размером ключа, разработанный Р. Риверстом. Описание алгоритма приведено по работам [2,3].

Алгоритм RC4 работает с  $n$  –битовыми словами (обычно  $n = 8$ ). Состояние генератора задается таблицей ( $S$  – блок) из  $2^n$  слов и двух переменных  $I$  и  $J$ . Первоначально переменные  $I$  и  $J$  равны нулю. Блок представляет собой зависящую от ключа  $K$  перестановку чисел от 0 до  $2^n - 1$ . Так как каждый элемент таблицы принимает значения в промежутке  $[0, 2^n - 1]$ , то его можно трактовать двояко: либо как число, либо как номер другого элемента в таблице. Криптоалгоритм формирует гамму в виде последовательности байт  $z_1 z_2 \dots z_m \dots$

Рассмотрим процедуру генерирования очередного псевдослучайного байта, которая состоит из пяти операций:

1. такт работы первого счетчика:  $I = (I + 1) \bmod 2^n$ ;
2. такт работы второго счетчика:  $J = (J + S_I) \bmod 2^n$ ;
3. ячейки блока замен с адресами  $I$  и  $J$  меняются местами:  
 $S_I \leftrightarrow S_J$ ;
4. вычисляется сумма:  $T = (S_I + S_J) \bmod 2^n$ ;
5. выдаем сгенерированный псевдослучайный байт:  $z = S_T$ .

Инициализация блока замены является очень простой процедурой, состоящей из четырех операций:

1. запись в каждую ячейку таблицы замен  $S$  – блока ее собственного адреса:  $S_0 = 0, S_1 = 1, \dots, S_{2^n-1} = 2^n - 1$ ;
2. заполнение словами ключа массив  $X$  из  $2^n$  слов при этом ключ может повторяться необходимое число раз для заполнения всего массива:  $X = \{x_i\}$ ;
3. инициализация индекса  $j$ :  $j = 0$ ;

4. перемешивание таблицы замен  $S$  – блока, для чего выполняем следующие действия

```
for  $i = 0$  to  $2^n - 1$  do
{
     $j = (j + x_i + S_i) \bmod 2^n$ 
     $S_i \leftrightarrow S_j$ 
}
```

В качестве иллюстрации работы шифра RC4 рассмотрим следующий пример.

*Пример.* Пусть  $n = 3$ ,  $K = 2,5$ .

Проведем инициализацию блока замены

$j = 0$	$S = (0,1,2,3,4,5,6,7)$	$X = (2,5,2,5,2,5,2,5)$
$i = 0$	$j = 0 + 0 + 2 = 2$	$S = (2,1,0,3,4,5,6,7)$
$i = 1$	$j = 2 + 1 + 5 = 0$	$S = (1,2,0,3,4,5,6,7)$
$i = 2$	$j = 0 + 2 + 0 = 2$	$S = (1,2,0,3,4,5,6,7)$
$i = 3$	$j = 2 + 3 + 5 = 2$	$S = (1,2,3,0,4,5,6,7)$
$i = 4$	$j = 2 + 4 + 2 = 0$	$S = (4,2,3,0,1,5,6,7)$
$i = 5$	$j = 0 + 5 + 5 = 2$	$S = (4,2,5,0,1,3,6,7)$
$i = 6$	$j = 2 + 6 + 2 = 2$	$S = (4,2,6,0,1,3,5,7)$
$i = 7$	$j = 2 + 7 + 5 = 6$	$S = (4,2,6,0,1,3,7,5)$

Сгенерируем несколько первых псевдослучайных байтов.

$I$	$J$	$S$	$T$	$z$
$I = 0 + 1 = 1$	$J = 0 + 2 = 2$	$S = (4,6,2,0,1,3,7,5)$	$T = 6 + 2 = 0$	$z_1 = 4 = 100_2$
$I = 1 + 1 = 2$	$J = 2 + 2 = 4$	$S = (4,6,1,0,2,3,7,5)$	$T = 1 + 2 = 3$	$z_2 = 0 = 000_2$
$I = 2 + 1 = 3$	$J = 4 + 0 = 4$	$S = (4,6,1,2,0,3,7,5)$	$T = 2 + 0 = 2$	$z_3 = 1 = 001_2$
$I = 3 + 1 = 4$	$J = 4 + 0 = 4$	$S = (4,6,1,2,0,3,7,5)$	$T = 0 + 0 = 0$	$z_4 = 4 = 100_2$

*Пример окончен.*

## Шифр A5

A5 - поточный шифр, применяемый для шифрования мобильной цифровой связи GSM (Group Special Mobile) между абонентом и базовой станцией.

Читатель может легко убедиться, что рассматриваемый шифр служит примером синхронного поточным шифра и основан на принципе комбинирования генератора.

Описание алгоритма приведено по работам [3,4]. Криптосистема A5/1 использует три LFSR, которые можно ассоциировать с двоичными многочленами:

- LFSR 1 -  $x^{19} + x^{18} + x^{17} + x^{14} + 1$ ,
- LFSR 2 -  $x^{22} + x^{21} + 1$ ,
- LFSR 3 -  $x^{23} + x^{22} + x^{21} + x^8 + 1$ .

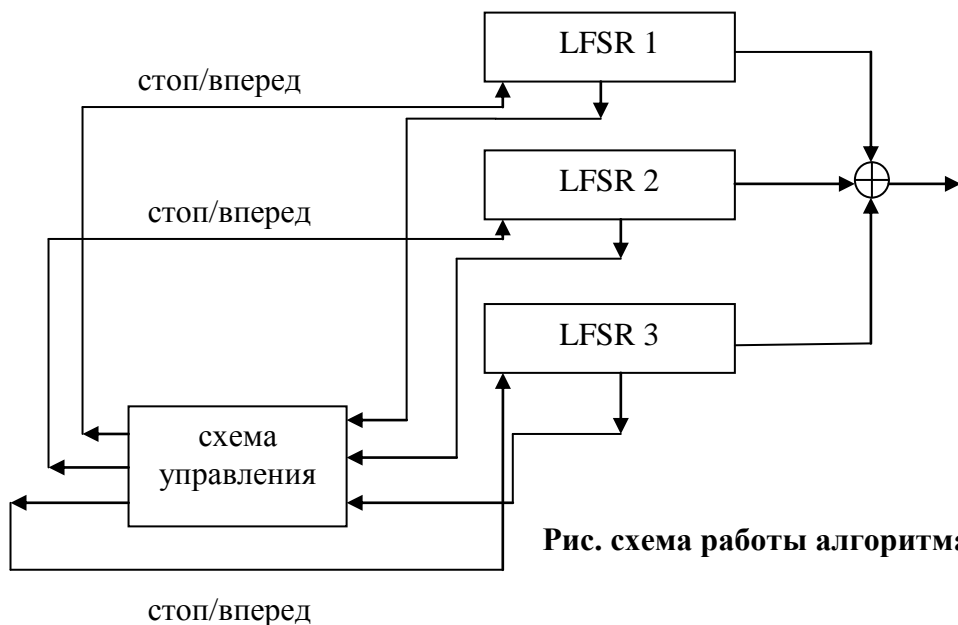


Рис. схема работы алгоритма A5/1

Выходной бит гаммы  $y(t)$  вычисляется по формуле

$$y(t) = s_1(t) \oplus s_2(t) \oplus s_3(t),$$

где  $s_i(t)$  соответствующий выходной бит(снимается с старшего разряда)  $i$  – го регистра. Регистры работают по принципу стоп/вперед, что обеспечивается с помощью функции  $majority(x_1, x_2, x_3)$ , результатом работы этой функции является бит, который определяет, какие регистры будут сдвигаться, а какие нет (если бит с выхода функции совпадает со значением бита регистра на определенной позиции, то регистр сдвигается, иначе нет).

Функция  $majority(x_1, x_2, x_3)$  вычисляется по формуле

$$majority(x_1, x_2, x_3) = x_1x_2 \vee x_1x_3 \vee x_2x_3,$$

где на вход функции подается значение битов регистров: восьмой разряд для LFSR 1, десятый разряд для LFSR 2, десятый разряд для LFSR 3. Работа регистров по правилу стоп/вперед обеспечивает нелинейность работы алгоритма в целом. Легко заметить, что на каждом такте сдвигаются два или три регистра (если  $x_1 = x_2 = x_3$ , сдвигаются все 3 регистра, в противном случае сдвигаются те 2 регистра  $i$  и  $j$ , для которых выполняется равенство  $x_i = x_j$ ).

Напомним, что начальное состояние всех LFSR является секретным сеансовым ключом.

## Шифр WAKE

WAKE - это потоковый шифр, который был предложен Д. Уилером. Шифр может служить примером самосинхронизирующего шифра.

В алгоритме используются четыре 32-разрядных регистра ( $A, B, C$  и  $D$ ) и блок замены  $S$ , состоящий из 256 32-разрядных значений. Заметим, что  $S$  – блок имеет особое свойство: старшие байты всех элементов таблицы являются перестановкой чисел от 0 до 255, а остальные три младших байта (каждого элемента таблицы) выбираются случайным образом. Шифр формирует гамму 32-разрядными словами.

Рассмотрим процедуру генерирования гаммы, которая состоит из трех операций:

1. на основе секретного ключа генерируются элементы  $S$  – блока;
2. используя секретный ключ, инициализируются регистры  $A, B, C$  и  $D$  соответственно значениями  $a_0, b_0, c_0, d_0$ ;
3. до тех пор пока нужны элементы гаммы, выполняем следующие действия:
  - получаем элемент гаммы  $k_i = d_i$ . Слово шифротекста получается поразрядным сложением по модулю два  $k_i$  и слова открытого текста  $p_i$ ;
  - состояния четырех регистров обновляются:

$$a_{i+1} = M(a_i, p_i \oplus d_i),$$

$$b_{i+1} = M(b_i, a_{i+1}),$$

$$c_{i+1} = M(c_i, b_{i+1}),$$

$$d_{i+1} = M(d_i, c_{i+1}).$$

Функция  $M$  действует на основе  $S$  – блока:

$$M(x, y) = (x + y) \gg 8 \oplus S_{(x+y) \bmod 256}.^7$$

Хотя автором шифра в общих чертах и дана процедура генерации блока замены  $S$ , но она не совсем завершенная. Считается, что здесь вполне подойдет любой другой алгоритм выбора перестановки и случайного заполнения.

## Шифр Fish

Fish - это потоковый шифр, который был предложен У. Блэхер и М. Дихтль.

Студенты должны обратить внимание на то, что рассматриваемый шифр служит примером сжимающей схемы примененной к аддитивным генераторам. Он выдает поток 32-битовых слов, который используется в качестве гаммы.

Алгоритм формирования гаммы состоит из трех шагов:

Шаг 1. используя аддитивные генераторы

$$a_i = (a_{i-55} + a_{i-24}) \bmod 2^{32}$$

$$b_i = (a_{i-52} + a_{i-19}) \bmod 2^{32}$$

генерируем две псевдослучайные последовательности  $a_0, a_1, \dots$  и  $b_0, b_1, \dots$ ;

---

<sup>7</sup> Операция  $\gg$  это обычный правый сдвиг. Младшие 8 бит суммы  $x + y$  являются входом  $S$  – блока.



Шаг 2. выполняется процедура сжатия. Она описывается следующим образом: если  $\gamma(b_i)=1^8$ , тогда  $a_i$  и  $b_i$  берутся, в противном случае они пропускаются. В результате получаются сжатые последовательности  $c_0, c_1, \dots$ , представляющая собой  $a_{i_1}, a_{i_2}, \dots$ ; и  $d_0, d_1, \dots$ , представляющая собой  $b_{i_1}, b_{i_2}, \dots$ . Для всех элементов  $\gamma(d_j)=1$ .

Шаг 3. завершающая процедура. Последовательности  $c_0, c_1, \dots$  и  $d_0, d_1, \dots$  разбиваются на пары  $(c_{2j}, c_{2j+1})$  и  $(d_{2j}, d_{2j+1})$ , на основе которых формируются два 32-битовых слова  $k_{2j}$  и  $k_{2j+1}$  следующим образом:

$$\begin{aligned} e_{2j} &= c_{2j} \oplus (d_{2j} \wedge d_{2j+1}) \\ f_{2j} &= d_{2j+1} \wedge (e_j \wedge c_{2j+1}) \\ k_{2j} &= e_{2j} \oplus f_{2j} \\ k_{2j+1} &= c_{2j+1} \oplus f_{2j}, \end{aligned}$$

где  $\oplus$  - побитовая логическая операция *Xor* и  $\wedge$  - побитовая логическая операция *And*.

## SEAL

SEAL- это потоковый шифр, разработанный Ф. Рогуэй и Д. Копперсмит.

Студенты должны обратить внимание на то, что SEAL описан как увеличивающая длину псевдослучайная функция, которая очевидным образом может быть использована для поточного шифрования.

Получая на вход 160-битовый ключ  $k$  и 32-битовый параметр  $n$  (индекс) шифр формирует последовательность  $SEAL_k(n)$  длиной  $L$ , где  $L$  не превосходит 64 Кбайт. Такой шифр будем обозначать  $SEAL(k, n, L)$ . Наиболее приемлемыми для реальных нужд считаются  $L$  с величинами от 512 до 4096 байт. Алгоритму для работы требуются восемь 32-битовых регистров и память объемом несколько Кбайт.

SEAL порождает последовательности переменной длины. Пусть  $L$  желаемое количество бит, алгоритм прекращает генерацию, как только сформировано  $L^1$  бит, где  $L^1$  - наименьшее кратное 128, большее или равное  $L$ .

Алгоритм состоит из нескольких шагов:

Введем обозначения:

$y \lll t$  - циклический сдвиг слова  $y$  на  $t$  разрядов влево;

$y \ggg t$  - циклический сдвиг слова  $y$  на  $t$  разрядов вправо;

$\wedge, \vee, \oplus, \neg$  - обозначают поразрядные операции соответственно *and, or, xor, not*;

символ  $\parallel$  обозначает операцию конкатенации;

---

<sup>8</sup> Отображение  $\gamma$ : отображает 32-битный вектор в его самый младший бит  $\gamma(w_{31}, w_{30}, \dots, w_0) = w_0$ .

$odd()$  - предикат, истинный тогда и только тогда, когда его аргумент – четное число;

$y + t$  сумма целых чисел  $y$  и  $t$  по модулю  $2^{32}$ .

Шаг 1. *Заполнение таблиц.*

Алгоритм SEAL использует зависимые от ключа таблицы  $R, S, T$ . Заполнение таблиц выполняется с помощью функции  $G$ , которая является функцией сжатия алгоритма хеширования SHA. Опишем функцию  $G_k(i)$  для 160-битовой последовательности  $k$  и 32-битового целого числа  $i$  ( $0 \leq i < 2^{32}$ ):

- Последовательность  $k$  разбивается на пять 32-битовых слов  $k = H_0 H_1 H_2 H_3 H_4$ .

- Строится 512-битовая последовательность  $Y$  по правилу  $i \parallel 0^{480}$ <sup>9</sup>.

- Блок  $Y$  рассматриваем как массив  $(w_0, \dots, w_{15})$  из 16 (32-разрядных) слов, так что  $w_0 = i$ ,  $w_1 = w_2 = \dots = w_{15} = 0^{32}$ . Блок  $Y$  расширяется до 80 слов по 32 разряда в каждом. Расширение происходит следующим образом. Пусть  $(w_0, \dots, w_{15})$  исходный блок,  $(W_0, \dots, W_{79})$  расширенный блок, при этом

$$W_t = \begin{cases} w_t & t = 0 \dots 15; \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1, & t = 16 \dots 79. \end{cases}$$

- Пусть  $a, b, c, d, e$  вспомогательные 32-битовые регистры и  $a = H_0$ ,  $b = H_1$ ,  $c = H_2$ ,  $d = H_3$ ,  $e = H_4$ .

- Выполняются 80 раундов алгоритма ( $t$  от 0 до 79), на каждом из которых происходит выполнение следующих операций:

$$\begin{aligned} temp &= (a \lll 5) + f_t(b, c, d) + e + W_t + K_t; & \text{обозначения} \\ e &= d; \quad d = c; & t - \text{номер раунда;} \\ c &= b \lll 30; & f_t - \text{раундовая функция;} \\ b &= a; \quad a = temp. & K_t - \text{раундовая константа.} \end{aligned}$$

Определим функцию  $f_t(x, y, z)$

$$f_t(x, y, z) = \begin{cases} (x \wedge y) \vee (\neg x \wedge z), & t = 0 \dots 19; \\ x \oplus y \oplus z, & t = 20 \dots 39, 60 \dots 79; \\ (x \wedge y) \vee (x \wedge z) \vee (y \wedge z), & t = 40 \dots 59. \end{cases}$$

Определим шестнадцатеричную константу  $K_t$

$$K_t = \begin{cases} 5A827999, & t = 0 \dots 19; \\ 6ED9EBA1, & t = 20 \dots 39; \\ 8F1BBCDC, & t = 40 \dots 59; \\ CA62C1D6, & t = 60 \dots 79. \end{cases}$$

<sup>9</sup>  $i \parallel 0^{480}$  - это конкатенация 32-битового числа  $i$  и 480-битовая последовательность нулей

- Выполнить сложением по модулю  $2^{32}$  полученных значений  $a, b, c, d, e$  соответственно с  $H_0 H_1 H_2 H_3 H_4$  :

$$H_0 = H_0 + a; \quad H_1 = H_1 + b; \quad H_2 = H_2 + c; \quad H_3 = H_3 + d; \quad H_4 = H_4 + e.$$

Описание функции  $G_k(i)$  завершено. После выполнения функции  $G_k(i)$  получим 160-битовое значение  $H_0 H_1 H_2 H_3 H_4$ .

Построим теперь функцию  $\Gamma$  с выходным значением длиной 32 бита, переиндексировав функцию  $G$ . Функция  $\Gamma$  определяется следующим образом:

$$\Gamma_k(i) = H_{i \bmod 5}^i,$$

где  $G_k(\lfloor i/5 \rfloor) = H_0^i H_1^i H_2^i H_3^i H_4^i$ .

Определим таблицы  $R, S, T$  следующим образом:

$$T[i] = \Gamma_k(i), \quad 0 \leq i < 512;$$

$$S[j] = \Gamma_k(0x1000 + j), \quad 0 \leq j < 256;$$

$$R[q] = \Gamma_k(0x2000 + q), \quad 0 \leq q < 4 \lceil (L-1)/8192 \rceil.$$

Читатель может легко убедиться, что  $q$  меньше 256 (напомним  $L$  не превосходит 64 Кбайт).

Шаг 1 завершен.

Шаг 2. *Генерация ключевой последовательности.*

Имея число  $L$ , таблицы  $R, S$  и  $T$ , заданные ключом  $k$  и параметр  $n$ , представленный ниже алгоритм растягивает  $n$  в  $L$ -битную последовательность  $y$ .

function SEAL(k; n; L)

{

$y = \emptyset$ ;

  for  $l = 0$  to  $\infty$  do {

    Initialize ( $n; l; A; B; C; D; n_1; n_2; n_3; n_4$ );

    for  $i = 1$  to 64 do {

$P = A \wedge 0x7fc$ ;  $B = B + T[P/4]$ ;  $A = A \gg \gg 9$ ;  $B = B \oplus A$ ;

$Q = B \wedge 0x7fc$ ;  $C = C \oplus T[Q/4]$ ;  $B = B \gg \gg 9$ ;  $C = C + B$ ;

$P = (P + C) \wedge 0x7fc$ ;  $D = D + T[P/4]$ ;  $C = C \gg \gg 9$ ;  $D = D \oplus C$ ;

$Q = (Q + D) \wedge 0x7fc$ ;  $A = A \oplus T[Q/4]$ ;  $D = D \gg \gg 9$ ;  $A = A + D$ ;

$P = (P + A) \wedge 0x7fc$ ;  $B = B \oplus T[P/4]$ ;  $A = A \gg \gg 9$ ;

$Q = (Q + B) \wedge 0x7fc$ ;  $C = C + T[Q/4]$ ;  $B = B \gg \gg 9$ ;

$P = (P + C) \wedge 0x7fc$ ;  $D = D \oplus T[P/4]$ ;  $C = C \gg \gg 9$ ;

$Q = (Q + D) \wedge 0x7fc$ ;  $A = A + T[Q/4]$ ;  $D = D \gg \gg 9$ ;

$y = y \parallel B + S[4i-4] \parallel C + S[4i-3] \parallel D + S[4i-2] \parallel A + S[4i-1]$ ;

      if  $|y| \geq L$  then return ( $y_0 y_1 \dots y_{L-1}$ );

      if odd( $i$ ) then ( $A; B; C; D$ ) = ( $A + n_1; B + n_2; C \oplus n_1; D \oplus n_2$ )

      else ( $A; B; C; D$ ) = ( $A + n_3; B + n_4; C \oplus n_3; D \oplus n_4$ );

```

    }
  }
}

```

Алгоритм использует подпрограмму Initialize для отображения  $n$  и  $l$  в значение внутренних переменных и регистров  $(A, B, C, D, n_1, n_2, n_3, n_4)$ .

*procedure Initialize* ( $A, B, C, D, n_1, n_2, n_3, n_4$ )

$A = n \oplus R[4l];$

$B = (n \ggg 8) \oplus R[4l + 1];$   $C = (n \ggg 16) \oplus R[4l + 2];$

$D = (n \ggg 24) \oplus R[4l + 3];$

for  $j=1$  to 2 do {

$P = A \wedge 0x7fc; B = B + T[P/4]; A = A \ggg 9;$

$P = B \wedge 0x7fc; C = C + T[P/4]; B = B \ggg 9;$

$P = C \wedge 0x7fc; D = D + T[P/4]; C = C \ggg 9;$

$P = D \wedge 0x7fc; A = A + T[P/4]; D = D \ggg 9;$

}

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

$P = A \wedge 0x7fc; B = B + T[P/4]; A = A \ggg 9;$

$P = B \wedge 0x7fc; C = C + T[P/4]; B = B \ggg 9;$

$P = C \wedge 0x7fc; D = D + T[P/4]; C = C \ggg 9;$

$P = D \wedge 0x7fc; A = A + T[P/4]; D = D \ggg 9;$

}

Шаг 2 завершен.

## Хеш-функция

*Определение.* Хеш-функция  $h$  принимает в качестве аргумента сообщение  $M$  произвольной длины и возвращает хеш-значение  $h(M) = H$  (свертку) фиксированной длины. ■

Примером хеш-функции может служить контрольная сумма для сообщения:  $h(x_1 x_2 \dots x_n) = (x_1 + x_2 + \dots + x_n) \bmod 2^w$ , где  $w$  размер машинного слова. Длина хеш-значения составит  $w$  бит независимо от длины сообщения. Однако рассмотренная хеш-функция не годится для криптографического применения.

Хеш-функция должна удовлетворять целому ряду условий:

- для любого заданного  $M$  вычисление  $h(M)$  должно выполняться относительно быстро;
- хеш-функция должна быть чувствительна к всевозможным изменениям в тексте  $M$ , таким как вставки, выбросы, перестановки;
- хеш-функция должна обладать свойством необратимости, то есть задача подбора документа  $M^1$ , который обладал бы требуемым значением хеш-функции, должна быть вычислительно неразрешима;

- вероятность того, что значения хеш-функций двух различных документов (вне зависимости от их длин) совпадут, должна быть ничтожно мала.

Функция хеширования может использоваться для обнаружения изменений сообщения, то есть она может служить для формирования криптографической контрольной суммы (также называемой кодом обнаружения изменений или кодом аутентификации сообщения). В этом качестве хеш-функция используется для контроля целостности сообщения, при формировании и проверке электронной цифровой подписи. Хеш-функции широко используются также в целях аутентификации пользователей.

Как правило, хеш-функции строят на основе одношаговых сжимающих функций  $y = f(x_1, x_2)$ , где  $x_i$  и  $y$  – двоичные векторы длины  $m$  и  $n$  соответственно, причём  $n$  – длина свёртки.

Для получения  $h(M)$  сообщение  $M$  сначала разбивается на блоки длины  $m$  (при этом если длина сообщения не кратна  $m$ , то последний блок неким специальным образом дополняется до полного), а затем к полученным блокам  $M_1, M_2, \dots, M_N$  применяют следующую последовательную процедуру вычисления свёртки:

$$H_0 = v;$$

$$H_i = f(M_i, H_{i-1}), \quad i = 1, \dots, N;$$

$$h(M) = H_N.$$

Здесь  $v$  некоторый фиксированный начальный вектор.

При изучении темы «хеш-функции» студентам необходимо обратить особое внимание на два типа криптографических хеш-функций:

- ключевые - они дают возможность без дополнительных средств гарантировать как правильность источника данных, так и целостность данных в системах с доверяющими друг другу пользователями (применяются в системах с симметричными ключами);
- бесключевые - они дают возможность с помощью дополнительных средств (например, шифрования или цифровой подписи) гарантировать целостность данных. Эти хеш-функции могут применяться в системах как с доверяющими, так и не доверяющими друг другу пользователями.

Приведем несколько примеров ключевой функции хеширования:

- Функция, построенная на основе одношаговой сжимающей функции вида

$$f_k(x, H) = E_k(x \oplus H), \quad \text{где } E_k - \text{алгоритм блочного шифрования (ключ } k \text{)}.$$

Алгоритм вычисления свертки имеет следующий вид:

$$H_0 = 0;$$

$$H_i = E_k(M_i \oplus H_{i-1}), \quad i = 1, \dots, N; \quad M_i - \text{текущий блок длины } m \text{ сообщения}$$

$$h(M) = H_N$$

$E_k$  – алгоритм блочного шифрования;

$M$  ;  
 $H_i$  – текущее хеш-значение.

- Основой для построения ключевых хеш-функций могут служить бесключевые хеш-функции. Пусть  $h(M)$  – бесключевая хеш-функция. Можно внедрить ключ  $k$  в процесс хеширования, и получить хеш-функцию с ключом  $h(k, M)$  (например  $h(k, M) = h(k \parallel M)$ <sup>10</sup>,  $h(k, M) = h(M \parallel k)$ ,  $h(k, M) = h(k \parallel M \parallel k)$ ). Заметим, что если ключ просто дописывается в начало или в конец исходного сообщения, то это может приводить к потенциальным слабостям. Таким образом, более предпочтительными являются способы введения ключа, при которых ключ вставляется в сообщение несколько раз.

Приведем несколько примеров бесключевой функции хеширования:

- Функция, построенная на основе одношаговой сжимающей функции вида  

$$f(x, H) = E_H(x) \oplus x$$
где  $E_H$  – алгоритм блочного шифрования (ключ  $H$ ).  
Следовательно алгоритм вычисления свертки имеет следующий вид:  

$$H_0 = v_0; \quad v_0 \text{ — стартовый вектор;}$$

$$H_i = E_{H_{i-1}}(M_i), \quad i = 1, \dots, N; \quad M_i \text{ — текущий блок длины } m \text{ сообщения}$$

$$h(M) = H_N \quad M;$$

$$H_i \text{ — текущее хеш-значение.}$$

- Функция, построенная на основе одношаговой сжимающей функции вида  

$$f(x, H) = E_x(H) \oplus H$$
где  $E_x$  – алгоритм блочного шифрования с ключом  $x$ .

Существует и ряд специально разработанных алгоритмов хеширования (например MD4, MD5, SHA1, SHA-256, SHA-512).

### **Хеш функция MD5**

Алгоритм получает на входе поток данных произвольной длины, а на выходе получаем хеш-значение длиной 128 бит. Сила этого алгоритма заключается в том, что практически очень сложно, почти невозможно, найти две строки, дающие одинаковые хеш-значения.

Процесс вычисления хеш-значения состоит из пяти шагов. Рассмотрим их подробнее[7].

Шаг 1. *Добавление недостающих битов.*

Сообщение дополняется так, что его длина (в битах) становится сравнимой с 448 по модулю 512. Расширение выполняется следующим образом: к сообщению добавляется один бит со значением 1, а оставшиеся биты заполняются нулевыми значениями.

Студенты не должны забывать, что добавление производится всегда, даже если длина сообщения уже сравнима с 448 по модулю 512. Таким обра-

<sup>10</sup> Символ  $\parallel$  обозначает конкатенацию, объединение строк аргументов

зом, количество добавленных битов может лежать в диапазоне от 1 до 512 включительно.

Шаг 2: *Добавление длины*.

Окончательно расширенное сообщение получается добавлением 64-битного представления длины исходного сообщения в битах присоединением к результату предыдущего шага. В том маловероятном случае, если первоначальная длина больше, чем  $2^{64}$  бит, то используются только младшие 64 бита двоичного представления длины. Эти биты добавляются в виде двух 32-разрядных слов, при этом младшее слово дописывается первым.

В результате первых двух шагов созданное расширенное сообщение можно представить как последовательность 512-битных блоков  $Y_0, Y_1, \dots$  или массив  $M[0 \dots N_1 - 1]$  (32-разрядных) слов. Заметим, что каждый блок  $Y_q$  во время вычисления будет рассматриваться как массив  $X[j]$  ( $0 \leq j < 16$ ) из 16 (32-разрядных) значений.

Шаг 3. *Инициализация буфера свертки*.

При вычислениях будет использоваться 128-битовый буфер для хранения результатов промежуточных вычислений хеш функции. Буфер можно представить как четыре 32-битных регистра ( $A, B, C, D$ ). Эти регистры инициализируются следующими числами:

$$\begin{aligned} A &= 01234567; & C &= FEDCBA98; \\ B &= 89ABCDEF; & D &= 76543210. \end{aligned}$$

Шаг 4. *Обработка сообщения блоками по 16 слов*.

Определим сначала несколько вспомогательных функций, аргументом и значением каждой из которых являются 32-битные слова:

$$\begin{aligned} f_F &= F(b, c, d) = (b \wedge c) \vee ((\neg b) \wedge d); \\ f_G &= G(b, c, d) = (b \wedge d) \vee (c \wedge (\neg d)); \\ f_H &= H(b, c, d) = b \oplus c \oplus d; \\ f_I &= I(b, c, d) = c \oplus (b \vee (\neg d)). \end{aligned}$$

На этом шаге используется массив из 64 (32-разрядных) слов  $T[0 \dots 63]$ . Пусть  $T[i]$  обозначает  $i$ -й элемент таблицы, который равен целой части от  $2^{32} * \text{abs}(\sin(i))$ , где  $i$  задается в радианах.

Для обработки выполняем следующие шаги:

```
for i = 0 to  $\frac{N_1}{16} - 1$  do
{
```

Копируем  $Y_i$  блок в массив  $X$ , то есть

for  $j = 0$  to 15 do  $X[j] = M[i * 16 + j]$

На время обработки очередного блока  $Y_i$ , значение хеш функции для предыдущего блока  $Y_{i-1}$  сохраняются в вспомогательных переменных:

$AA, BB, CC, DD$ .

$$AA = A; \quad BB = B; \quad CC = C; \quad DD = D.$$

Теперь выполняем четыре раунда.

**Раунд 1**

Пусть  $[abcd \ k \ s \ i]$  обозначает операцию

$$a = b + ((a + F(b, c, d) + X[k] + T[i]) \lll s)^{11}.$$

Выполняется 16 операций:

[ABCD 0 7 1]	[DABC 1 12 2]	[CDAB 2 17 3]	[BCDA 3 22 4]
[ABCD 4 7 5]	[DABC 5 12 6]	[CDAB 6 17 7]	[BCDA 7 22 8]
[ABCD 8 7 9]	[DABC 9 12 10]	[CDAB 10 17 11]	[BCDA 11 22 12]
[ABCD 12 7 13]	[DABC 13 12 14]	[CDAB 14 17 15]	[BCDA 15 22 16]

**Раунд 2**

Пусть  $[abcd \ k \ s \ i]$  обозначает операцию

$$a = b + ((a + G(b, c, d) + X[k] + T[i]) \lll s)$$

Выполняется 16 операций:

[ABCD 1 5 17]	[DABC 6 9 18]	[CDAB 11 14 19]	[BCDA 0 20 20]
[ABCD 5 5 21]	[DABC 10 9 22]	[CDAB 15 14 23]	[BCDA 4 20 24]
[ABCD 9 5 25]	[DABC 14 9 26]	[CDAB 3 14 27]	[BCDA 8 20 28]
[ABCD 13 5 29]	[DABC 2 9 30]	[CDAB 7 14 31]	[BCDA 12 20 32]

**Раунд 3**

Пусть  $[abcd \ k \ s \ i]$  обозначает операцию

$$a = b + ((a + H(b, c, d) + X[k] + T[i]) \lll s)$$

Выполняется 16 операций:

[ABCD 5 4 33]	[DABC 8 11 34]	[CDAB 11 16 35]	[BCDA 14 23 36]
[ABCD 1 4 37]	[DABC 4 11 38]	[CDAB 7 16 39]	[BCDA 10 23 40]
[ABCD 13 4 41]	[DABC 0 11 42]	[CDAB 3 16 43]	[BCDA 6 23 44]
[ABCD 9 4 45]	[DABC 12 11 46]	[CDAB 15 16 47]	[BCDA 2 23 48]

**Раунд 4**

Пусть  $[abcd \ k \ s \ i]$  обозначает операцию

$$a = b + ((a + I(b, c, d) + X[k] + T[i]) \lll s)$$

Выполняется 16 операций

[ABCD 0 6 49]	[DABC 7 10 50]	[CDAB 14 15 51]	[BCDA 5 21 52]
[ABCD 12 6 53]	[DABC 3 10 54]	[CDAB 10 15 55]	[BCDA 1 21 56]
[ABCD 8 6 57]	[DABC 15 10 58]	[CDAB 6 15 59]	[BCDA 13 21 60]
[ABCD 4 6 61]	[DABC 11 10 62]	[CDAB 2 15 63]	[BCDA 9 21 64]

Выполнить сложение

$$A = A + AA;$$

$$B = B + BB;$$

$$C = C + CC;$$

$$D = D + DD.$$

}

Шаг 5. Выход.

<sup>11</sup> Определим операцию побитового циклического сдвига влево  $X$  на  $Y$  бит:  $X \lll Y$ .



Окончательный результат находится в буфере  $(A, B, C, D)$  и . это и есть хеш-значение, но выводим начиная с младшего байта  $A$  и закончив старшим байтом  $D$ .

### **Хеш функция SHA-1**

Хеш-функция SHA-1 осуществляет преобразование информационной последовательности произвольной длины в хеш-образ разрядностью 160 бит.

Процесс вычисления хеш-значения состоит из пяти шагов.

Шаг 1. *Добавление недостающих битов.*

Формируется расширенное сообщение путем добавления к исходному сообщению битов таким образом, чтобы его длина стала равна 448 по модулю 512. Это означает, что длина расширенного сообщения на 64 бита меньше, чем число, кратное 512.

Студенты не должны забывать, что добавление битов производится всегда, даже если длина сообщения уже сравнима с указанной. Например, если длина сообщения 448 битов, оно дополняется 512 битами до 960 битов.

Добавление выполняется следующим образом: к концу сообщения приписывается единица, а затем необходимое количество нулей.

Шаг 2. *Добавление длины.*

Окончательно расширенное сообщение получается добавлением 64-битного представления длины исходного сообщения в битах присоединением к результату первого шага. Если длина исходного сообщения больше  $2^{64}$ , то используются только 64 младших разряда этого кода. В результате созданное расширенное сообщение можно представить как последовательность 512-битных блоков  $Y_1, Y_2, \dots, Y_N$ .

Шаг 3. *Инициализация буфера свертки.*

При вычислениях будет использоваться 160-битовый буфер для хранения результатов промежуточных вычислений хеш функции. Буфер может быть представлен как пять 32-битных регистра  $(A, B, C, D, E)$ . Эти регистры инициализируются 32-битными числами:

$$A = 67452301;$$

$$D = 10325476;$$

$$B = EFCDAB89;$$

$$E = C3D2E1F0.$$

$$C = 98BADCFE;$$

Следовательно  $SHA_0 = (A, B, C, D, E)$ .

Шаг 4. *Обработка текущего блока  $Y_i$ .*

Преобразование текущего блока  $Y_i$  можно задать формулой  $SHA_i = f(Y_i, SHA_{i-1})$ . Опишем преобразование более подробно:

- На время обработки очередного блока  $Y_i$ , значение хеш-функции для предыдущего блока  $Y_{i-1}$  сохраняются в вспомогательных переменных:

$$a, b, c, d, e.$$

$$a = A;$$

$$b = B;$$

$$c = C;$$

$$d = D;$$

$$e = E.$$

- Каждый исходный блок  $Y_i$  рассматриваем как массив  $(w_0, \dots, w_{15})$  из 16 (32-разрядных) слов. Блок  $Y_i$  расширяется до 80 слов по 32 разряда в каждом. Расширение происходит следующим образом. Пусть  $(w_0, \dots, w_{15})$  исходный блок,  $(W_0, \dots, W_{79})$  расширенный блок, при этом

$$W_t = \begin{cases} w_t & t = 0 \dots 15; \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1, & t = 16 \dots 79. \end{cases}$$

- Выполняются 80 раундов алгоритма ( $t$  от 0 до 79), на каждом из которых происходит выполнение следующих операций:

$$temp = (a \lll 5) + f_t(b, c, d) + e + W_t + K_t;$$

$$e = d; \quad d = c;$$

$$c = b \lll 30;$$

$$b = a; \quad a = temp,$$

где  $t$  - номер раунда;  $f_t$  - раундовая функция;  $K_t$  - раундовая константа.

Определим функцию  $f_t(x, y, z)$

$$f_t(x, y, z) = \begin{cases} x \wedge y \vee \neg x \wedge z, & t = 0 \dots 19; \\ x \oplus y \oplus z, & t = 20 \dots 39, 60 \dots 79; \\ x \wedge y \vee x \wedge z \vee y \wedge z, & t = 40 \dots 59. \end{cases}$$

Определим шестнадцатеричную константу  $K_t$

$$K_t = \begin{cases} 5A827999, & t = 0 \dots 19; \\ 6ED9EBA1, & t = 20 \dots 39; \\ 8F1BBCDC, & t = 40 \dots 59; \\ CA62C1D6, & t = 60 \dots 79. \end{cases}$$

Цикл завершается сложением по модулю  $2^{32}$  полученных значений  $a, b, c, d, e$  соответственно с  $A, B, C, D, E$ :

$$A = A + a; \quad B = B + b; \quad C = C + c; \quad D = D + d; \quad E = E + e.$$

Осуществляется переход к обработке следующего 512-битового блока расширенного сообщения.

Шаг 5. *Выход.*

Выходное значение хеш-функции является конкатенация значений  $A, B, C, D, E$ .

## Задачи

1. Построить генератор LFSR с ассоциированным двоичным многочленом  $x^8 + x^4 + x^3 + x^2 + 1$  схема Фибоначчи и схема Галуа.
2. Построить генератор LFSR с ассоциированным двоичным многочленом  $x^8 + x^7 + x^5 + x^3 + 1$  схема Фибоначчи и схема Галуа.

3. Найти LFSR, если нам известна последовательность битов, которая была им сгенерирована 100110101111000.
4. Построить генератор на основе композиционной схемы, если  
     LFSR1 ассоциирован с многочленом  $x^8 + x^4 + x^3 + x^2 + 1$   
     LFSR2 ассоциирован с многочленом  $x^8 + x^7 + x^5 + x^3 + 1$
5. Построить генератор на основе сжимающей схемы, если  
     LFSR1 ассоциирован с многочленом  $x^7 + x^3 + 1$   
     LFSR2 ассоциирован с многочленом  $x^9 + x^4 + 1$
6. Построить аддитивный генератор для случая, когда  $C(x) = x^7 + x^3 + 1$ ,  $n = 8$ . Показать, какая рекуррентная последовательность формируется.
7. Построить аддитивный генератор для случая, когда  $C(x) = x^9 + x^4 + 1$ ,  $n = 8$ . Показать, какая рекуррентная последовательность формируется.
8. Построить генератор на принципе сжимающей схемы примененного к аддитивным генераторам, построенных в задачах 7 и 8.
9. Построить 10 битную псевдослучайную последовательность с помощью BBS-генератора  $p = 7$ ,  $q = 23$ .
10. Построить 10 битную псевдослучайную последовательность с помощью RSA-генератора  $p = 17$ ,  $q = 11$ .
11. Модифицируйте алгоритм RC4 с помощью стохастического преобразования
12. Построить генератор на основе блочного шифра DES в режиме CTR.
13. Вычислить гамму в алгоритме SEAL:  
      $k = 67452301 \quad efcdab89 \quad 98badcfe \quad 10325476 \quad 00000001$  ;  
      $L = 32768$ ;  $n = 013577af$  .
14. Вычислить хеш-значение для текста “md4” на основе функции MD5.
15. Вычислите хеш-значение для текста “sha” на основе функции SHA-1.
16. Привести описание ключевой хеш-функции на основе алгоритма AES.

## Литература

1. Шнайер Б. Прикладная криптография. М.: Триумф, 2002.
2. Рябко Б.Я., Фионов А.Н. Криптографические методы защиты информации. М.: Горячая линия – Телеком, 2005
3. Асосков А.В., Иванов М.А., Мирский А.А. Поточные шифры. М.: Кудиц-образ, 2003
4. Киселёв С.А., Токарева Н.Н. О сокращении ключевого пространства шифра А5/1 и обратимости функции следующего состояния в поточном генераторе // Дискретный анализ и исследование операций Том 18, № 2, 2011
5. Блейхут Р. Теория и практика кодов, контролирующих ошибки. М.: Мир, 1986

6. Бурдаев О.В., Иванов М.А., Тетерин И.И. Ассемблер в задачах защиты информации. М.: Кудиц-образ, 2004
7. Баричев С. Г, Серов Р. Е. Основы современной криптографии М.: Горячая линия — Телеком, 2001
8. Иванов М. А., Чугунков И. В. Теория, применение и оценка качества генераторов псевдослучайных последовательностей. М.: Кудиц – Образ, 2003.