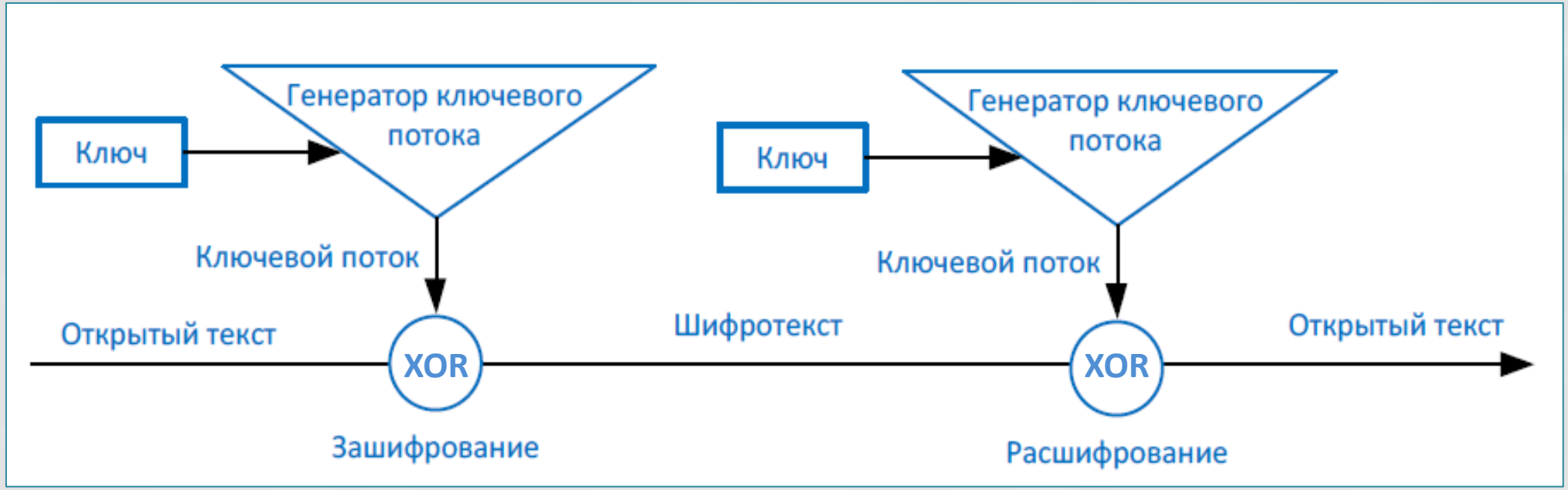
# Поточные шифры и генераторы случайных последовательностей

# Поточный шифр




- Поточные шифры (*stream cipher*) обрабатывают сообщение, как поток битов и выполняют математические функции над каждым битом отдельно
- Поточные шифры используют генератор ключевого потока (гаммы), который производит поток битов, объединяемых с помощью операции XOR с битами открытого текста
- Различают синхронные и асинхронные поточные шифры

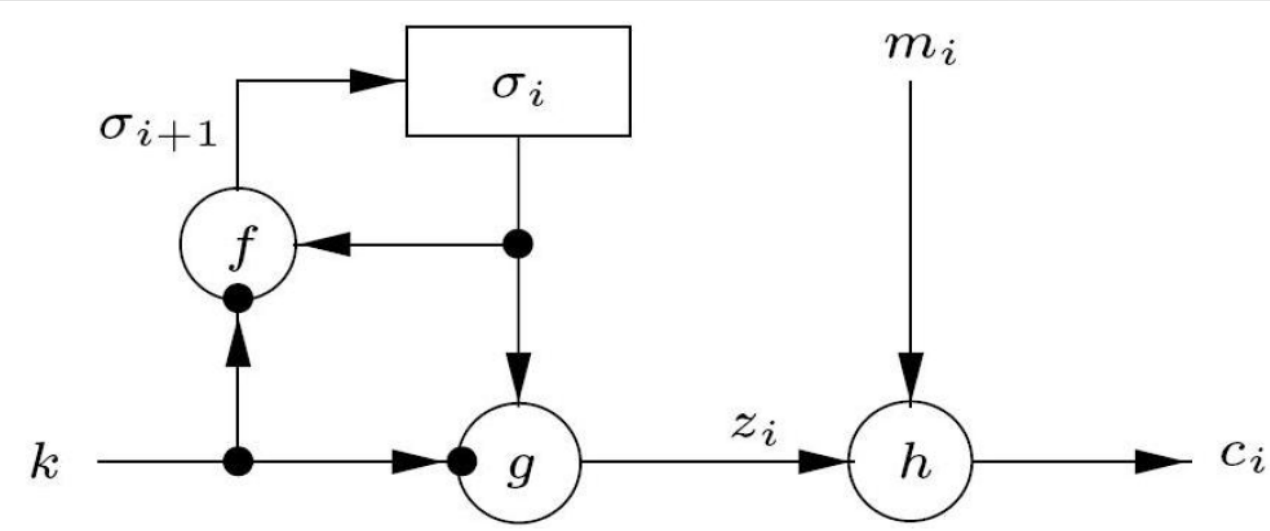
# Синхронные поточные шифры

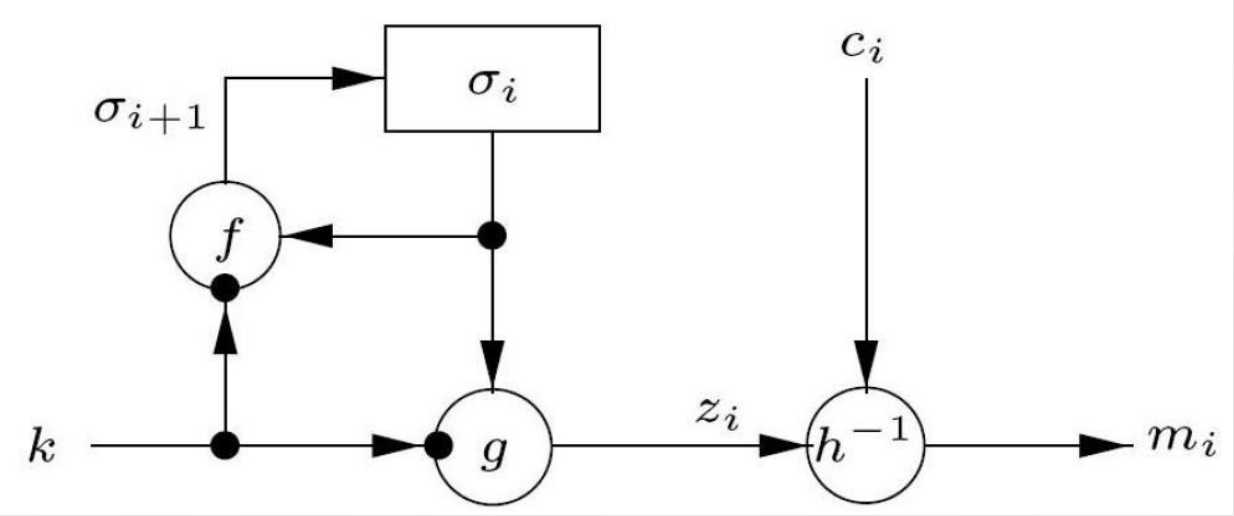
- В *синхронных* поточных шифрах поток ключа генерируется независимо от сообщения и кода.
- Кодирование синхронных шифров можно описать как

$$\begin{aligned} \sigma_{i+1} &= f(\sigma_i, k), \\ z_i &= g(\sigma_i, k), \\ c_i &= h_i(z_i, m_i). \end{aligned}$$

- $\sigma_0$ — начальное состояние, $k$ — ключ, $f$ — функция перехода между состояниями, $g$ производит поток ключей $z$, а код $c$ получается из этого потока и сообщения.

# Зашифрование и расшифрование



# Свойства синхронных шифров

- Нужно синхронизировать. Если вдруг в канале могут пропадать биты или появляться новые, нужны специальные методы синхронизации.
- Ошибка не распространяется дальше.
- Относительно атак:
  - из-за первого свойства атаки, связанные с новыми символами или удалением, легче заметить;
  - из-за второго свойства, если взломщик может менять код, он будет знать, как это повлияет на сообщение.

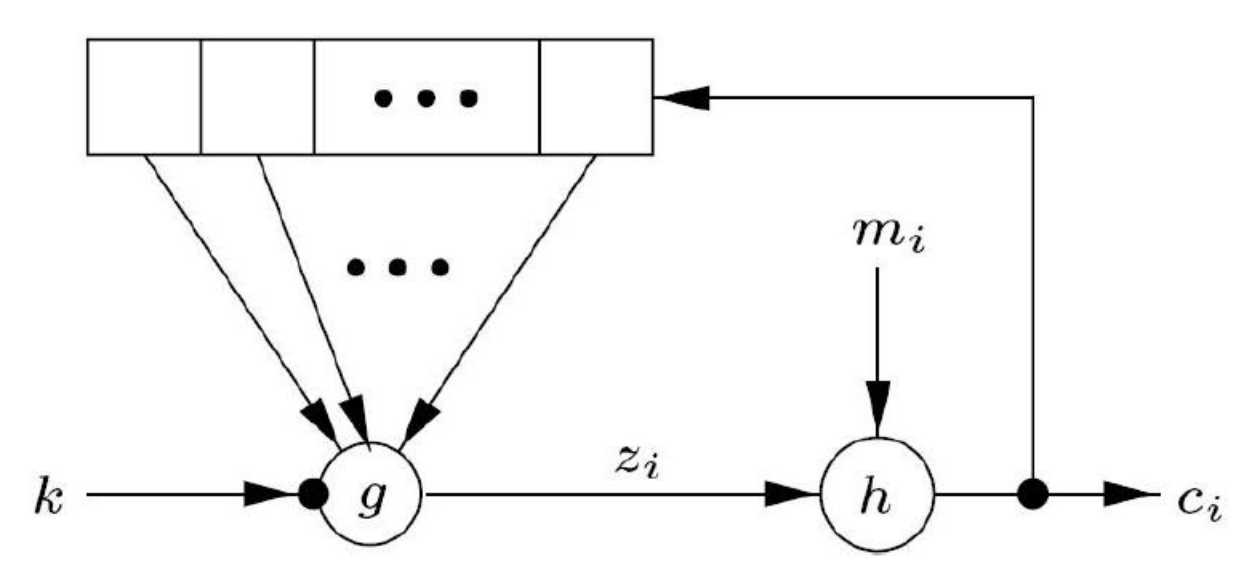
# Самосинхронизирующиеся шифры

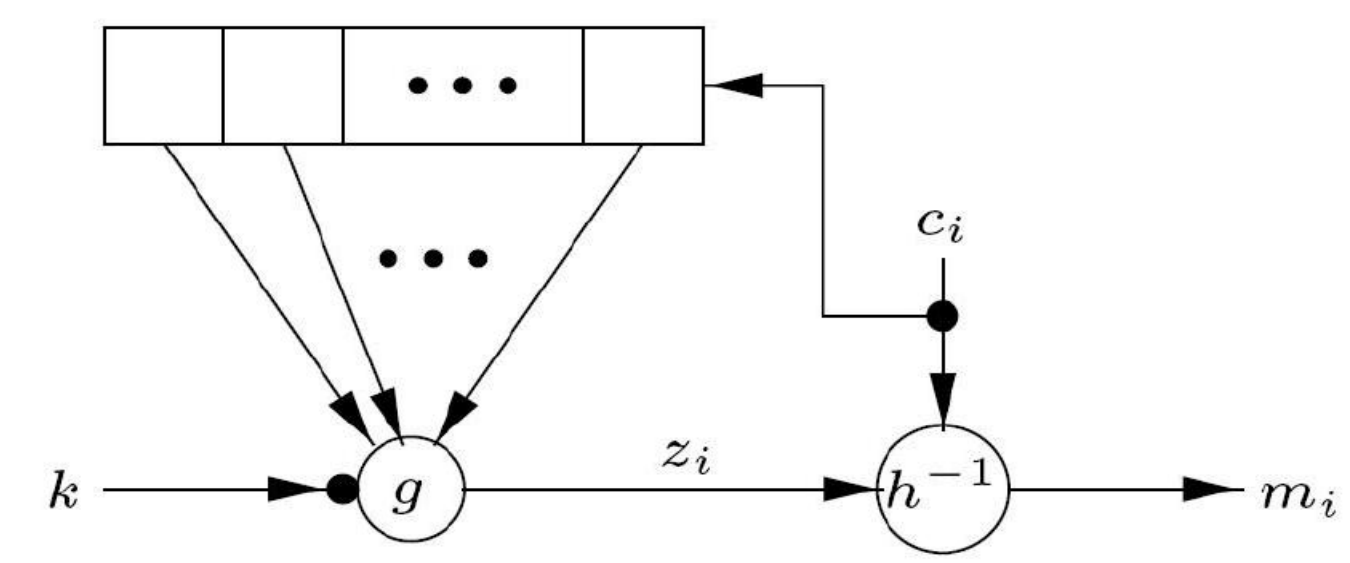
- В *самосинхронизирующихся* поточных шифрах поток ключа — функция сообщения и нескольких предшествующих битов кода.
- Кодирование самосинхронизирующихся шифров можно описать как

$$\begin{aligned}\sigma_i &= f(c_{i-t}, c_{i-t+1}, \ldots, c_{i-1}),\\ z_i &= g(\sigma_i, k),\\ c_i &= h_i(z_i, m_i).\end{aligned}$$

- $\sigma_0$ — начальное состояние, $k$ — ключ, $f$ — функция перехода между состояниями, $g$ производит поток ключей $z$, а код $c$ получается из этого потока и сообщения.

# Зашифрование и расшифрование



# Свойства самосинхронизирующихся шифров

- Синхронизируются сами: каждая ошибка влияет только на конечное число последующих битов.
- Ошибка распространяется, но ограниченно.
- Относительно атак:
  - из-за первого свойства атаки, связанные с новыми символами или удалением, труднее заметить;
  - из-за второго свойства, если взломщик изменил код, это повлияет на следующие биты, что может позволить заметить вторжение.

# Требования к криптостойким генераторам

- Генерируемая псевдослучайная последовательность должна иметь как можно больший период.
- Зная любой фрагмент последовательности, выдаваемой генератором, злоумышленник не должен иметь эффективной возможности найти начальное значение, загруженное в генератор
- Зная любой фрагмент последовательности, выдаваемой генератором, злоумышленник не должен иметь возможности получить достоверную информацию о предыдущих или последующих элементах последовательности
- Эффективная аппаратная и программная реализация

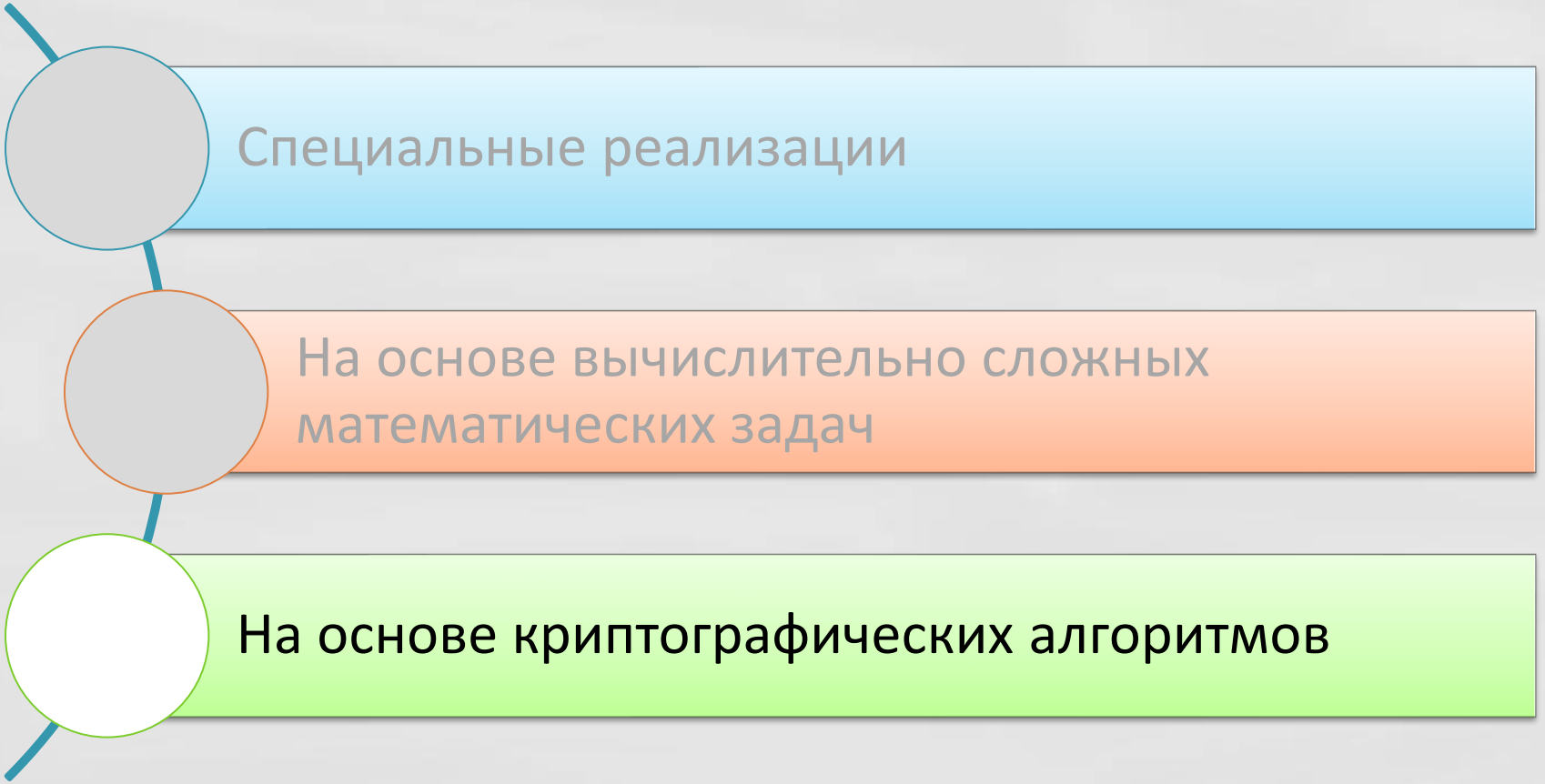
# Линейные конгруэнтные генераторы

- Это простейший генератор псевдослучайных чисел:
  - $X_i$=(a*$X_{i-1}$ +b) mod c, где $X_0$-начальное состояние, *a, b, c* – константы
- Последовательность определенная числами $X_0$ *a, b, c* имеет период длиной *c* тогда и только тогда, когда
  - числа *c и b* – взаимно простые;
  - *(a-1)* кратно каждому простому p, которое является делителем *c*
- При программной реализации значение *c* обычно устанавливается равным $2^{b-1}$, где *b* — длина слова в битах
- В качестве результата следует брать только старшие биты переменной состояния $X_i$
- Достоинство – высокая скорость генерации псевдослучайных чисел, недостаток простота восстановления последовательности по нескольким значениям

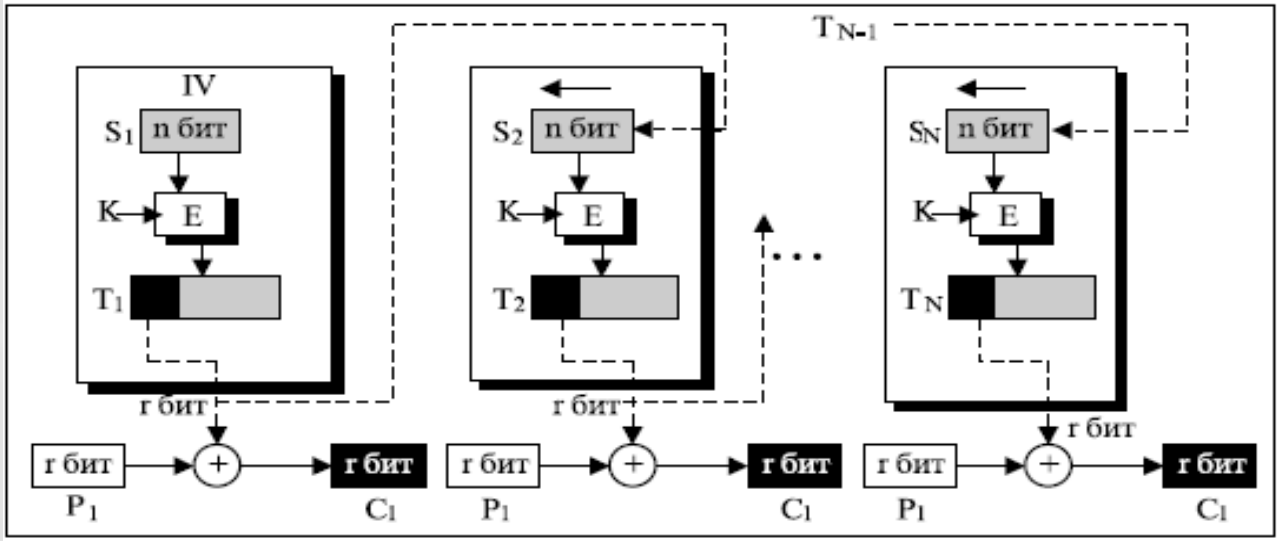
# Пример: генератор RANDU

- $V_{j+1} = (65539 \times V_j) \mathrm{mod} 2^{31}$, $V_0$- нечетное
- $X_j$=$V_j/2^{31}$- случайное число равномерно  распределенное на [0,1)
- Выбор значения 65539=$2^{16}$+3 был связан с эффективностью реализации операции умножения ($\times$)  по модулю $2^{31}$на 32 битом процессоре
- Вычислим $V_{j+2}$ (подстановка и выполнение вычислений по $\mathrm{mod} 2^{31}$)
  - Результат: $V_{j+2}$=$6V_{j+1} - 9V_j$ - очень просто вычислить очередное значение

# Способы реализации криптостойких генераторов



# Режим обратной связи по выходу (OFB)



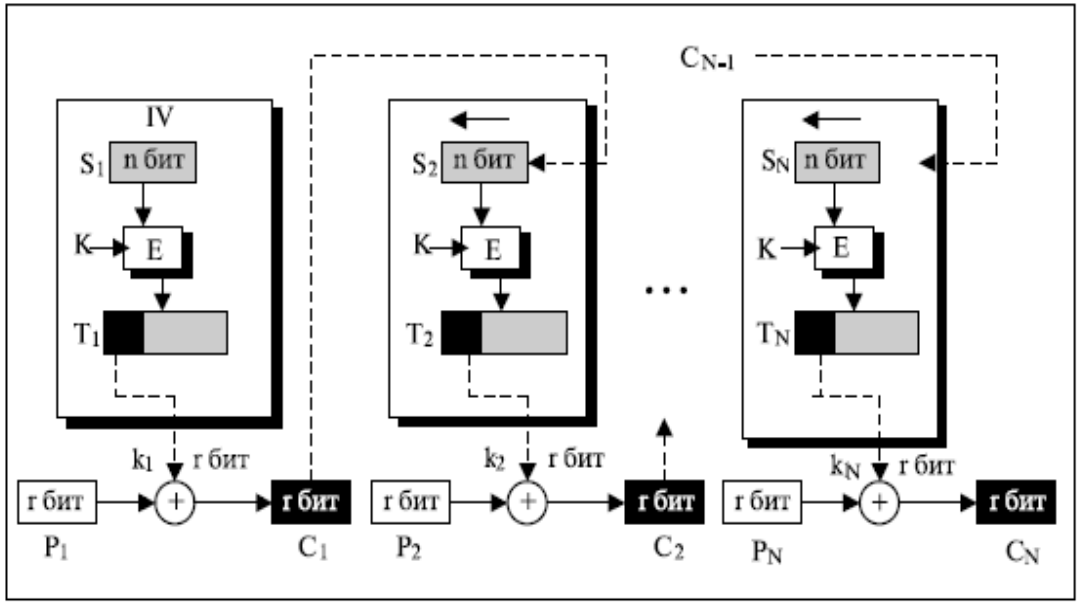

Шифрование

| | | | |
|---|---|---|---|
| E: | Шифрование | D: | Дешифрование |
| $P_i$: | Блок i исходного текста | $C_i$: | Блок i зашифрованного текста |
| K: | Секретный ключ | IV: | Начальный вектор (S1) |
| $S_i$: | Регистр сдвига | $T_i$: | Временный регистр |

- Каждый бит в зашифрованном тексте независим от предыдущего бита или битов. Это позволяет избежать распространения ошибок
- Это синхронный поточный шифр. Удаление бита из потока приводит к рассинхронизации между сторонами .

**OFB - Output Feedback**

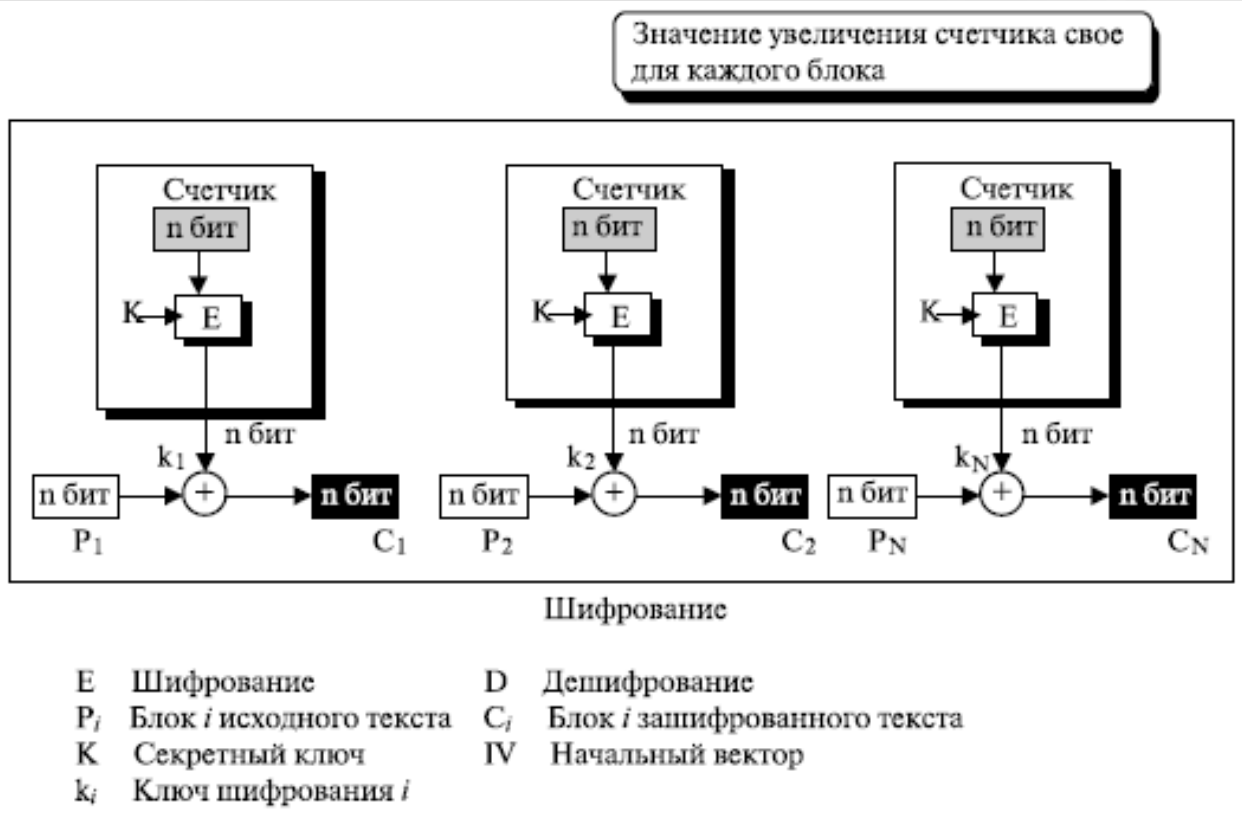
# Режим обратной связи по шифру ( CFB)




- Это шифр потока, в котором ключевой поток зависит от r-бит зашифрованного текста и , поэтому, допускает самосинхронизацию
- Ошибка в единственном бите в шифротекста создает ошибку в следующих блоках до тех пор, пока, ошибка находятся в регистре сдвига

**CFB - Cipher Feedback**

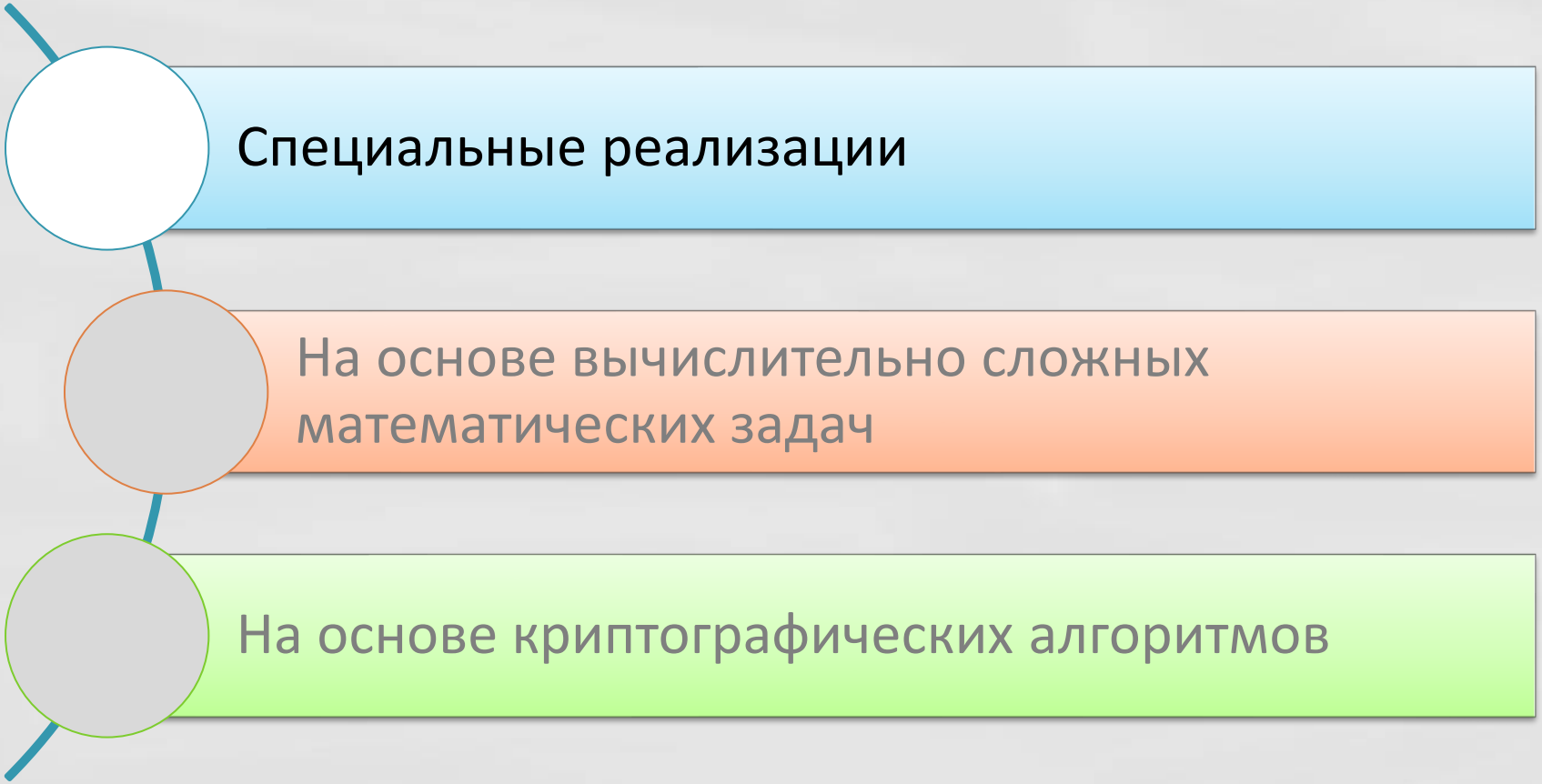
# Режим счетчика



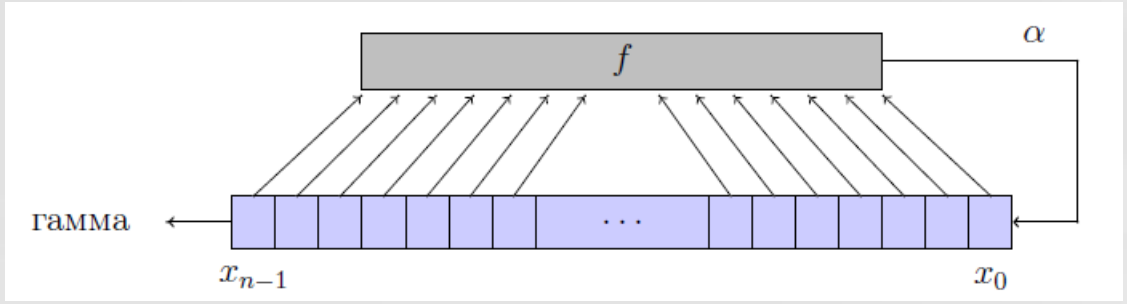

- Создает n -битовый зашифрованный текст, блоки которого независимы друг от друга — они зависят только от значений счетчика.
- Это синхронный  шифр потока
- Если значения счетчиков совпадает, то шифрование производится на одном ключе

**CTR - Counter**

# Способы реализации криптостойких генераторов

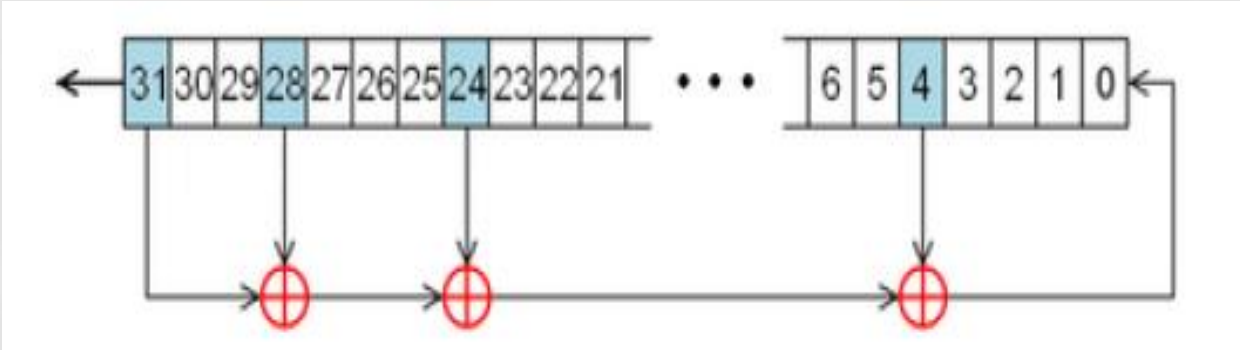


# Сдвиговые регистры с обратной связью




- Псевдослучайная последовательность (гамма) генерируется по одному биту за такт
- регистр изменять свое состояние во времени с помощью функции обратной связи $f$ вырабатывающей значение $\alpha$
- Период генератора составит $2^n - 1$, если номера отводов соответствуют примитивному полиному степени $n$

**Пример: РСЛОС соответствующий полиному $x^{32} + x^{29} + x^{25} + x^5 + 1$**

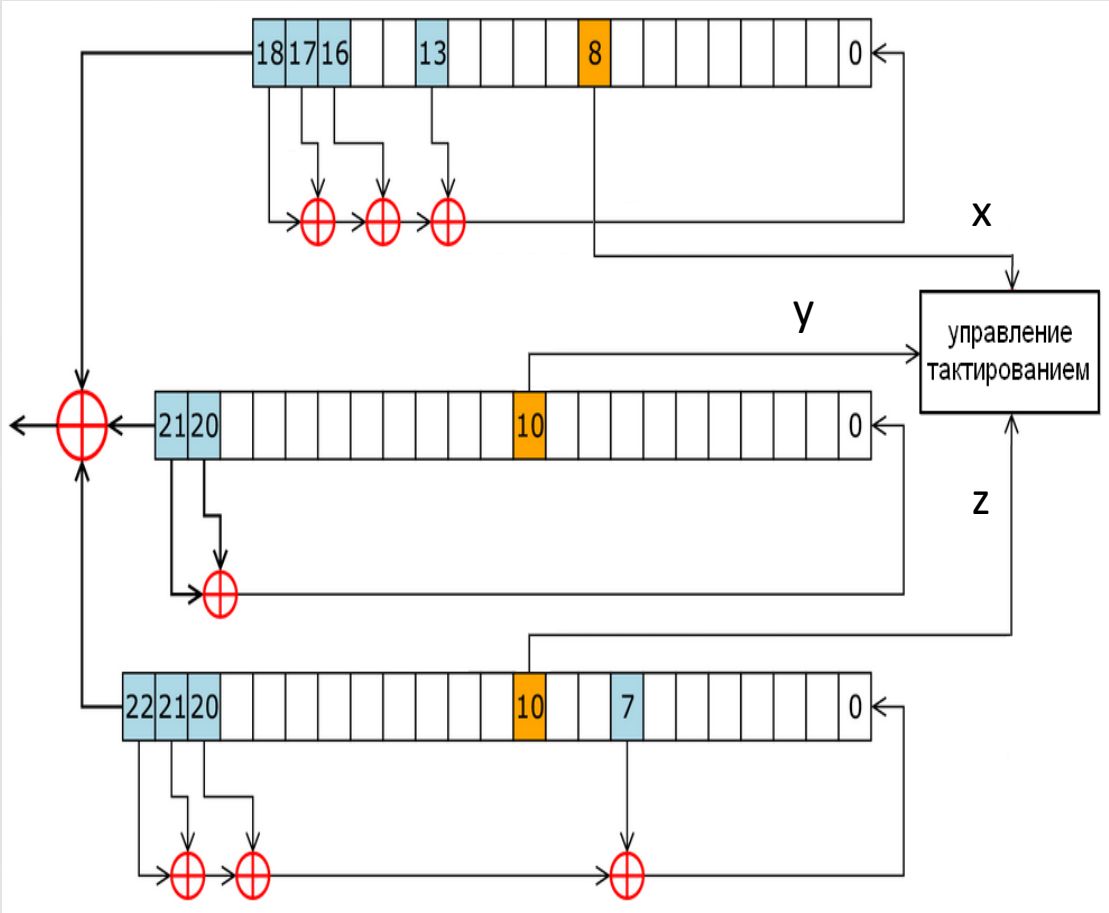



**Linear Feedback Shift Register**

# Криптоанализ РСЛОС

- Цель - найти начальное состояние регистра
- Если известен примитивный многочлен степени *n*:
  - Перехватываем n бит и загружаем в регистр не меняя порядок
  - Производим все действия в обратном порядке до получения начального состояния:
    - Вычисляем функцию обратной связи
    - Сдвигаем вправо и заменяем старший бит на значение обратной связи
- Знание состояния позволяет получить все сгенерированные в будущем последовательности

# Потоковый шифр A5/1




- Три LFSR регистра R1, R2, R3 имеют длины 19, 22 и 23 бита. Начальное состояние регистров – сеансовый ключ 64 бита

Управление тактированием осуществляется специальным механизмом:

- в каждом регистре есть биты синхронизации: 8 (R1), 10 (R2), 10 (R3),
- мажоритарная функция F = x&y V x&z V y&z,

  где x, y и z — биты синхронизации R1, R2 и R3 соответственно
- сдвигаются только те регистры, у которых бит синхронизации равен F (синхробит которых принадлежит большинству)

Выходной бит системы — результат операции XOR над выходными битами

# Криптоанализ A5/1

- **Атака «грубой силы».** Перебирается содержимое первых двух регистров, а содержимое третьего регистра восстанавливается по шифрующей гамме. Требуется перебрать $2^{40}$ вариантов ключей

- **Атака на основе открытого текста.** Базируется на специфики стандарта GSM, в котором помимо голосового трафика передаются различные известные системные сообщения. При оснащении атакующего компьютером с SSD-накопителем на 2 ТБ, минимум 3 ГБ памяти и двумя графическими процессорами (3D-ускорители для игр), есть возможность взламывать шифрование GSM-переговоров (A5/1) практически в режиме реального времени

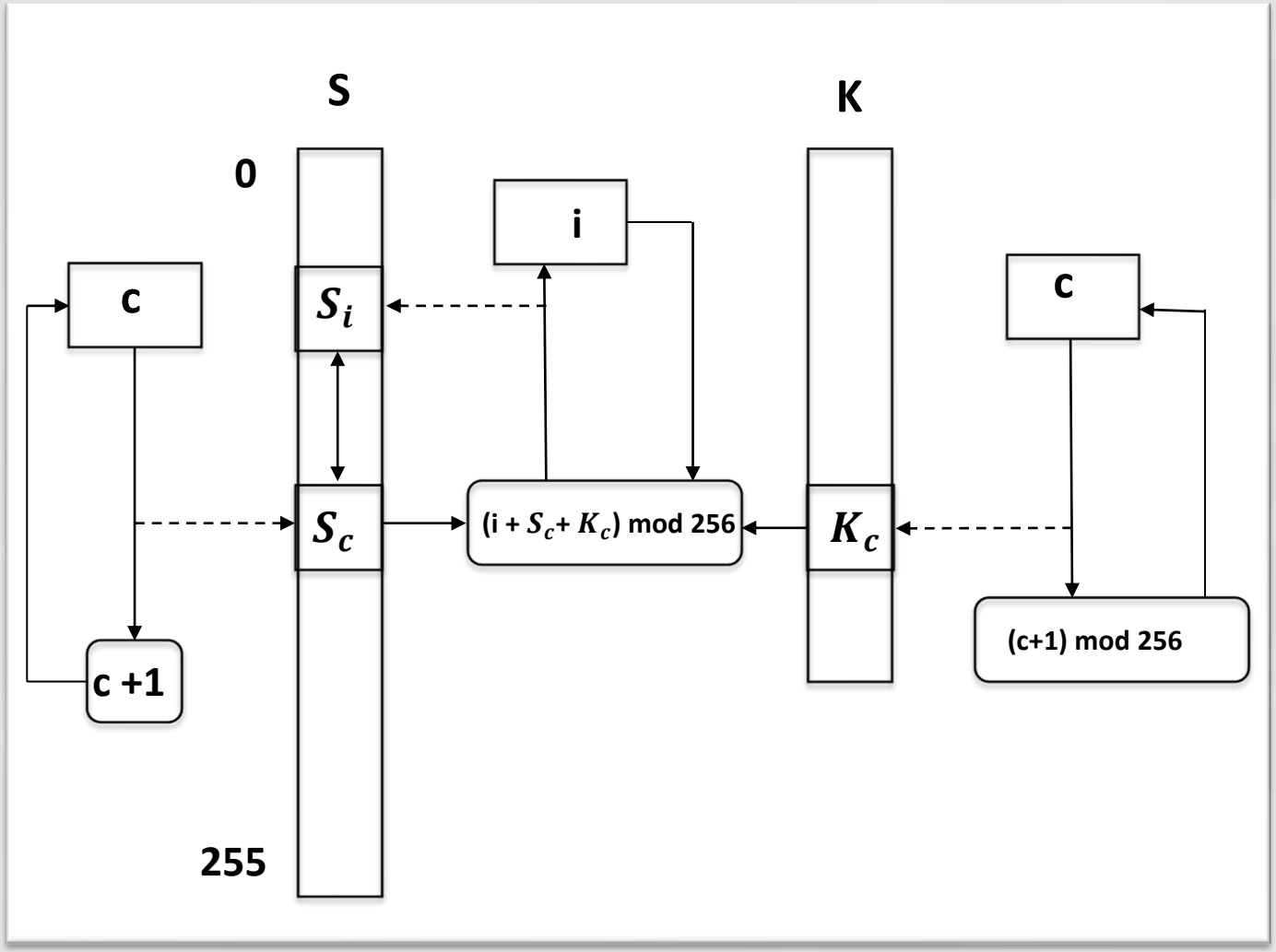
# Аддитивные генераторы

- Метод Фибоначчи с запаздыванием (*Lagged Fibonacci Generator*) – выдает случайные n-битовые слова :
  - $X_i=(X_{i-a} + X_{i-b}) \bmod 2^n$, где *a, b* – величины запаздываний
  - Для максимального качества рекомендованы пары (a, b): (17,5), (55,24) или (97,33) (соответствуют степеням примитивных много членов). Чем больше запаздывания, тем больше период последовательности
- Качественный (по мнению Б. Шнайера), но ресурсоемкий алгоритм. Требуется хранение массива чисел из предыстории случайной последовательности
- Ключом является массив *n*-битовых слов размера *max{a,b}*

# Потоковый шифр RC4

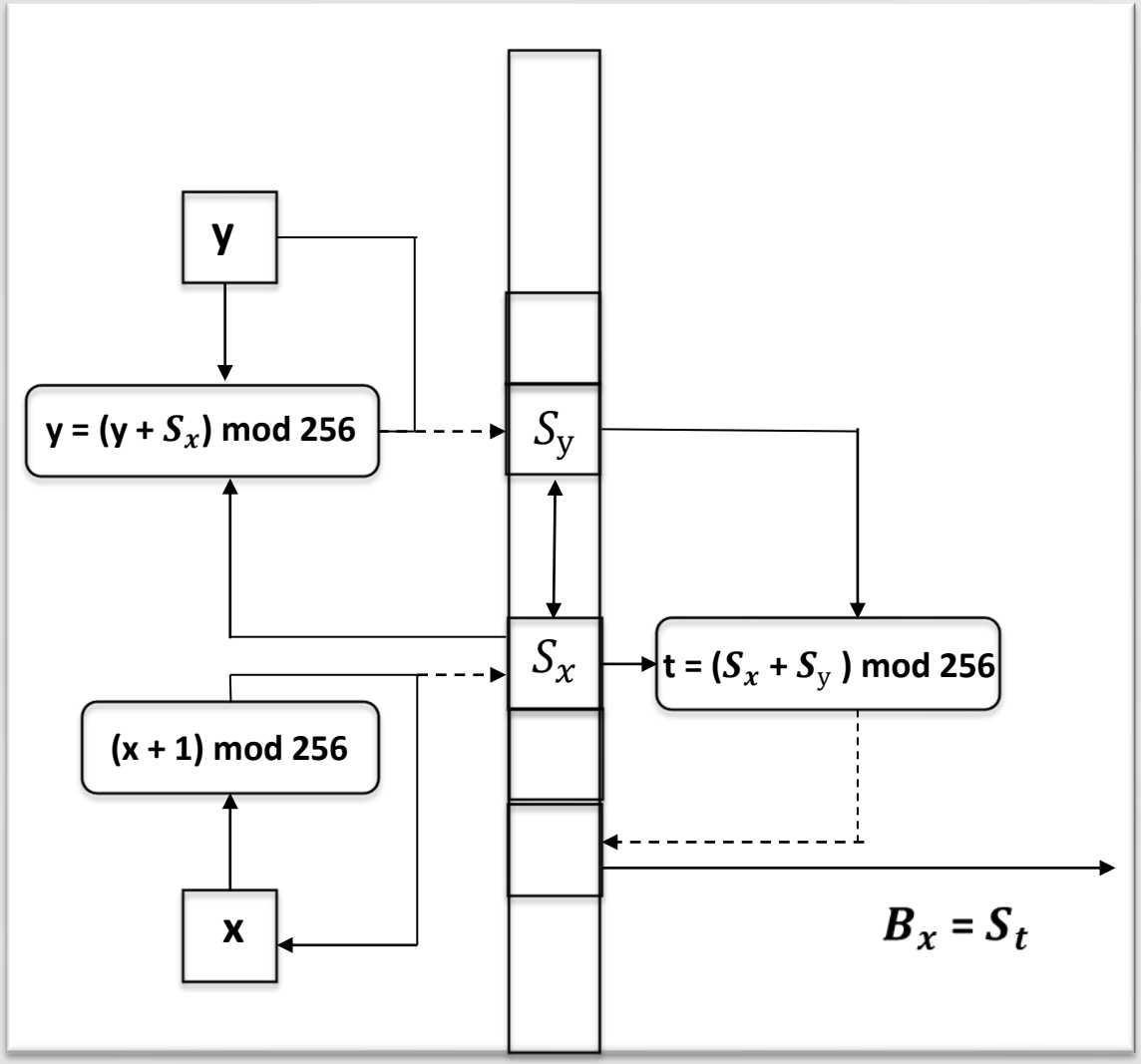
- RC4 — потоковый шифр,  разработан в 1984 г. Используется во многих сетевых протоколах, например, SSL/TLS и IEEE 802.11 (стандарт беспроводных сетей)
- Это байт-ориентированный шифр потока, в котором  каждый байт исходного текста складывается (XOR) с байтом ключа
- Секретный ключ, на основе  которого сгенерированы однобайтовые ключи в потоке ключей,  может быть переменной длины (от 1 до 256 байтов)

# Потоковый шифр RC4: инициализация




- 1. K (256 байт) заполняется ключом (40 байт при экспорте шифра, 56 байт для зарубежных компаний США, ключи повторяются в регистре )
- 2. S заполняется числами [0, 255]
- 3. c = 0; i =0;
- 4. i = (i + $S_c$+ $K_c$) mod 256;
- 5. поменять местами $S_c$ и $S_i$
- 6. c = c +1;
- 7. если c < 256, то перейти на п.4

# Потоковый шифр RC4: генерация




- 1. x = (x + 1) mod 256;
- 2. y = (y + $S_x$) mod 256;
- 3. поменять местами $S_x$ и $S_y$
- 4. t = ($S_x$ + $S_y$ ) mod 256;
- 5. $B_x = S_t$

# Пример криптоанализа RC4 (2013)

- Метод основан на уязвимости RC4 -генерация псевдослучайного потока с неравномерным распределением значений каждого из байтов. Тогда можно построить матрицу $\mathrm{P} = \{p_{i,j}\}$ генерации значения *j* в *i*-ом байте ключевого потока и применить статистический метод
- Атака позволяет полностью восстановить исходное сообщение, если имеется ~$2^{30}$ разных шифровок этого сообщения. Строим $E = \{e_{i,j}\}$- выборочных частот встречаемости символа j на i-ой позиции шифротекста. Каждому элементу из *E* соответствует элемент $p_{i,j\oplus m(i)}$ , где *m(i)* – *i*-ый символ исходного сообщения
- Строится функция правдоподобия того, что *i-ый* символ сообщения равен v

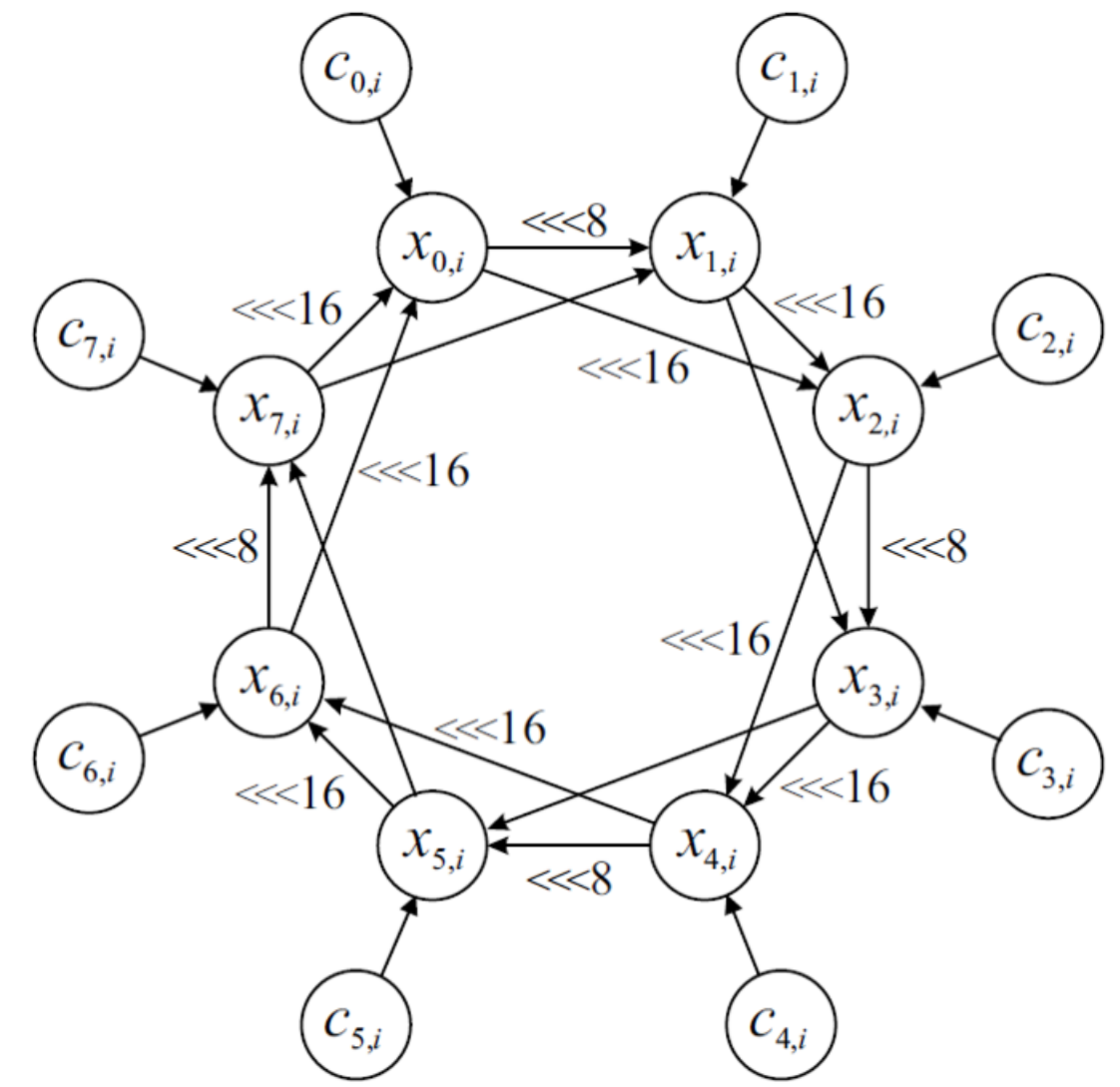
$$\log \mathrm{F(i,v)} = \sum_j e_{i,j} \log p_{i,j\oplus m(i)}$$

- При расшифровке восстанавливаем *m(i)* символом *v,* на котором достигается максимум логарифма функции правдоподобия

# Поточной шифр Rabbit

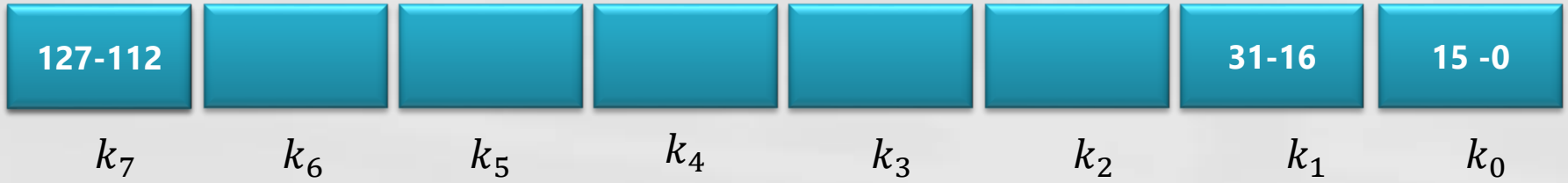
- Rabbit – участник конкурса eStream, организованного ЕС в 2004-2008 г. г., по выявлению новых поточных шифров, пригодных для широкого применения
- На конкурс принимались шифры, производительность которых превосходила производительность шифра AES-128, работающего в режиме счётчика (CTR)
- Rabbit стал победителем в категории 1 – «Поточные шифры для программной реализации» с основным требованием высокой производительности
- Rabbit используют 128-битный ключ для генерации битового потока, с помощью которого шифруется 128 битов сообщения за одну итерацию

# Схема работы шифра




- Внутреннее состояние шифра содержит 513 битов, содержащих:
  - 8 переменных состояния $x_{0,i}$, … $x_{7,i}$ по 32 бита каждая, где i – номер итерации
  - 8 счетчиков $c_{0,i}$, … $c_{7,i}$ по 32 бита каждый, где i – номер итерации
  - Бит переноса (513-ый) $\varphi_{7,i}$, передающийся между итерациями
- Внутреннее состояние шифра инициализируется 128-битным ключом

# Инициализация внутреннего состояния




$$x_{j,0} = \begin{cases} k_{(j+1 \mod 8)} \diamond k_j, & \text{for even } j, \\ k_{(j+5 \mod 8)} \diamond k_{(j+4 \mod 8)}, & \text{for odd } j, \end{cases}$$

$$c_{j,0} = \begin{cases} k_{(j+4 \mod 8)} \diamond k_{(j+5 \mod 8)}, & \text{for even } j, \\ k_j \diamond k_{(j+1 \mod 8)}, & \text{for odd } j, \end{cases}$$

- Инициализация выполняется за 4-ре итерации ( j=1,4)
- Окончательно счетчики реинициализируются так:

¤ - операция конкатенации 16-ти битовых строк

$$c_{j,4} = c_{j,4} \oplus x_{(j+4 mod 8),4}$$

# Функция перехода в следующее состояние

$$
\begin{aligned}
x_{0,i+1} &= g_{0,i} + (g_{7,i} \lll 16) + (g_{6,i} \lll 16) \\
x_{1,i+1} &= g_{1,i} + (g_{0,i} \lll 8) + g_{7,i} \\
x_{2,i+1} &= g_{2,i} + (g_{1,i} \lll 16) + (g_{0,i} \lll 16) \\
x_{3,i+1} &= g_{3,i} + (g_{2,i} \lll 8) + g_{1,i} \\
x_{4,i+1} &= g_{4,i} + (g_{3,i} \lll 16) + (g_{2,i} \lll 16) \\
x_{5,i+1} &= g_{5,i} + (g_{4,i} \lll 8) + g_{3,i} \\
x_{6,i+1} &= g_{6,i} + (g_{5,i} \lll 16) + (g_{4,i} \lll 16) \\
x_{7,i+1} &= g_{7,i} + (g_{6,i} \lll 8) + g_{5,i} \\
g_{j,i} &= \mathrm{LSW}((x_{j,i} + c_{j,i})^2) \oplus \mathrm{MSW}((x_{j,i} + c_{j,i})^2)
\end{aligned}
$$

Здесь все сложения по модулю $2^{32}$. Функции LSW(x) и MSW(x) возвращают, соответственно, младшие и старшие четыре байта 64-разрядного числа x, «— циклический сдвиг влево.

# Изменение системы счетчиков

$$
\begin{aligned}
c_{0,i+1} &= c_{0,i} + a_0 + \phi_{7,i} \mod 2^{32} \\
c_{1,i+1} &= c_{1,i} + a_1 + \phi_{0,i+1} \mod 2^{32} \\
c_{2,i+1} &= c_{2,i} + a_2 + \phi_{1,i+1} \mod 2^{32} \\
c_{3,i+1} &= c_{3,i} + a_3 + \phi_{2,i+1} \mod 2^{32} \\
c_{4,i+1} &= c_{4,i} + a_4 + \phi_{3,i+1} \mod 2^{32} \\
c_{5,i+1} &= c_{5,i} + a_5 + \phi_{4,i+1} \mod 2^{32} \\
c_{6,i+1} &= c_{6,i} + a_6 + \phi_{5,i+1} \mod 2^{32} \\
c_{7,i+1} &= c_{7,i} + a_7 + \phi_{6,i+1} \mod 2^{32}
\end{aligned}
$$

где счетчик бита переноса $\phi_{j,i+1}$ задаётся как

$$
\phi_{j,i+1} = \begin{cases} 1, & \text{if } c_{0,i} + a_0 + \phi_{7,i} \geqslant 2^{32} \wedge j = 0, \\ 1, & \text{if } c_{j,i} + a_j + \phi_{j-1,i+1} \geqslant 2^{32} \wedge j > 0, \\ 0, & \text{otherwise.} \end{cases}
$$

Кроме того, константы $a_j$ определяются как

$$
\begin{aligned}
a_0 &= \texttt{0x4D34D34D}, & a_1 &= \texttt{0xD34D34D3}, \\
a_2 &= \texttt{0x34D34D34}, & a_3 &= \texttt{0x4D34D34D}, \\
a_4 &= \texttt{0xD34D34D3}, & a_5 &= \texttt{0x34D34D34}, \\
a_6 &= \texttt{0x4D34D34D}, & a_7 &= \texttt{0xD34D34D3}.
\end{aligned}
$$

# Извлечение фрагмента гаммы

$$
\begin{aligned}
s_i^{[15..0]} &= x_{0,i}^{[15..0]} \oplus x_{5,i}^{[31..16]} \\
s_i^{[31..16]} &= x_{0,i}^{[31..16]} \oplus x_{3,i}^{[15..0]} \\
s_i^{[47..32]} &= x_{2,i}^{[15..0]} \oplus x_{7,i}^{[31..16]} \\
s_i^{[63..48]} &= x_{2,i}^{[31..16]} \oplus x_{5,i}^{[15..0]} \\
s_i^{[79..64]} &= x_{4,i}^{[15..0]} \oplus x_{1,i}^{[31..16]} \\
s_i^{[95..80]} &= x_{4,i}^{[31..16]} \oplus x_{7,i}^{[15..0]} \\
s_i^{[111..96]} &= x_{6,i}^{[15..0]} \oplus x_{3,i}^{[31..16]} \\
s_i^{[127..112]} &= x_{6,i}^{[31..16]} \oplus x_{1,i}^{[15..0]}
\end{aligned}
$$

где $s_i$ — 128-битный блок шифрующего потока на $i$-й итерации.

# Зашифрование и расшифрование

$$c_i = p_i \oplus s_i,$$

$$p_i = c_i \oplus s_i,$$

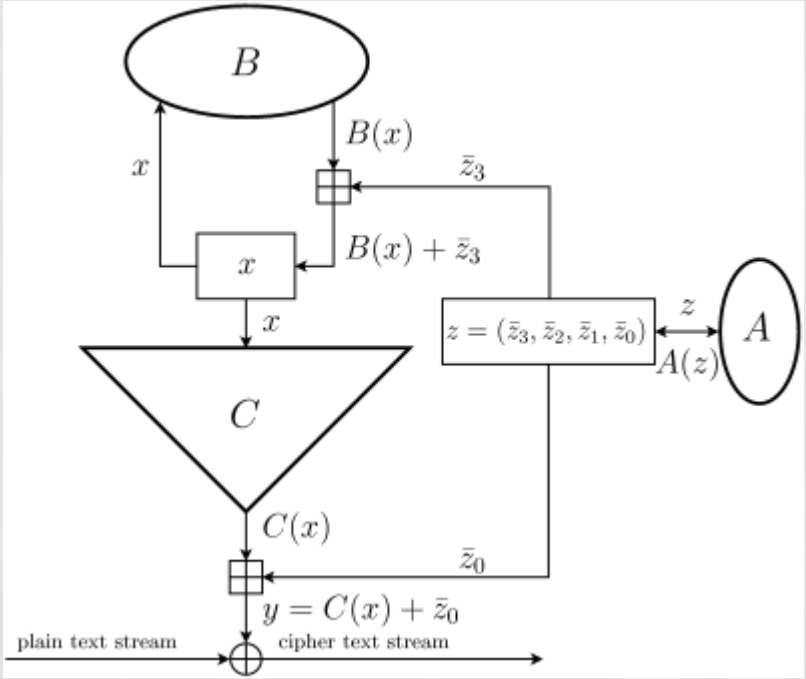
где $c_i$ и $p_i$ обозначают $i$-й блок шифротекста и текста соответственно.

# Свойства шифра

- Этап расширения ключа гарантирует взаимно однозначное соответствие между ключом состояния и ключом счётчика, который предотвращает избыток ключей. Он также распределяет биты ключа оптимальным образом для итераций.
- Система итераций гарантирует, что после одной итерации функции следующего состояния каждый бит ключа повлияет на все восемь переменных состояния. Это также гарантирует, что после второй итерации функции следующего состояния все биты ключа повлияют на все биты состояния с вероятностью 0,5. Для надёжности шифрования итерацию проделывают четыре раза.
- Даже если счетчики будут известны злоумышленнику, модификация счётчика сильно усложняет восстановление ключа путём инвертирования счётчика системы, так как потребуются сведения о переменных состояния, также это нарушает взаимно однозначное соотношение между ключом и счетчиком.
- Достоинство шифра в тщательном перемешивании его внутренних состояний между двумя последовательными итерациями. Функция перемешивания полностью основана на арифметических операциях, доступных на современных процессорах, то есть S-блоки подстановок и поисковые таблицы не нужны для реализации шифра.

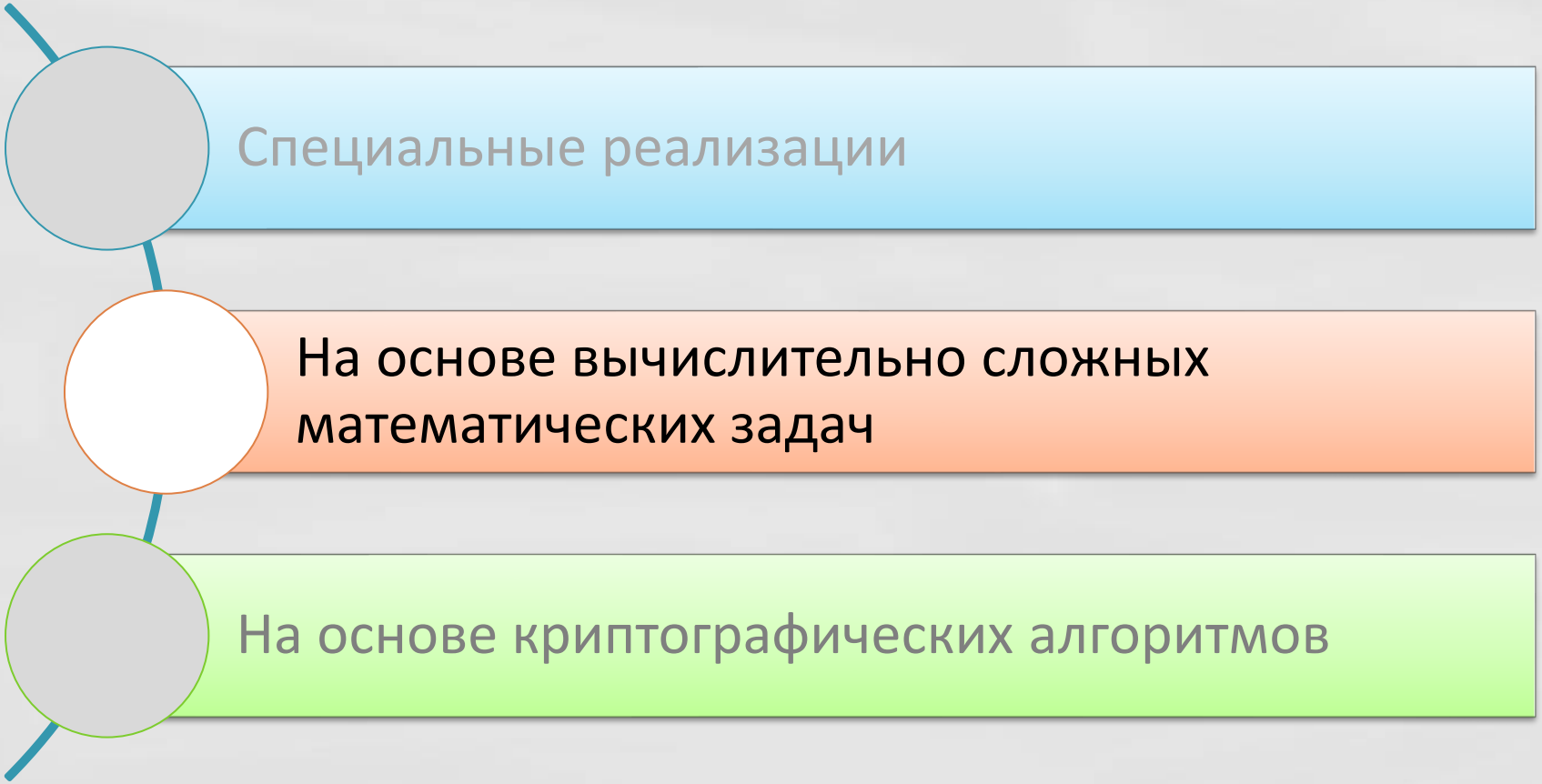
# Поточный шифр ABC

- Разработчики - В. С. Анашин, А. Ю. Богданов, И. С. Кижватов, Российский государственный гуманитарный университет, г. Москва
- Синхронный поточный шифр, оптимизированный для программных приложений
- Версия шифра ABC со 128-битным ключом и 32-битными внутренними переменными принимала участие в конкурса eSTREAM (2004-2008)
- Шифр гарантируют рекордную длину периода гаммы $2^{32}(2^{127}\text{-}1)$ бит




«ABC: новый быстрый гибкий поточный шифр» - https://www.ecrypt.eu.org/stream/ciphers/abc/abc.pdf НЕДОВ. САЙТ !!!

# Способы реализации криптостойких генераторов



# Генератор RSA

- $q$ и $p$ – большие простые числа, $N=p*q$
- $e$ – целое число взаимно простое с $(p-1)*(q-1)$
- $x_0$- ключ
- $x_i=x_{i-1}^e mod\ N$
- Выход: младший бит $x_i$
- Безопасность алгоритма основана на сложности разложения большого числа $N$ на множители

# Генератор с квадратичным остатком

- Алгоритм BBS (авторы — L. Blum, M. Blum, M. Shub):
  - $X_{i+1} = X_i^2 \bmod M$
  - $X_0 = X^2 \bmod M$, где X случайное целое число, взаимно простое с M
  - M=p*q, где p и q большие простые числа $p \equiv q \equiv 3 \bmod 4$
- Результатом i-го шага является один (обычно младший) бит числа $X_{i+1}$.
- Безопасность алгоритма BBS основана на сложности разложения большого числа *M* на множители

# Генератор Blum-Micali

- $g$ и $p$ – простые числа
- $x_0$- ключ
- $x_i = g^{x_{i-1}} mod\ p$
- Выход: $k_i = \left\{1, \text{если } x_i < {}^{p-1}/_2 \middle| 0, \text{если } x_i \geq {}^{p-1}/_2\right\}$
- Безопасность этого генератора определяется трудностью вычисления дискретных логарифмов

# Тестирование псевдослучайных последовательностей

- Цель: проверки случайного характера генерируемой бинарной последовательности
- Реализация: набор статистических тестов, построенных по следующим принципам:
  - Выбирается критерий и подходящая статистика для проверки
  - Задается ошибка 1-го рода (вероятность ложного отклонения «случайности») и выбирается длина последовательности из соображения минимизации ошибки 2-го рода (вероятность пропуска «неслучайности» )
  - Вычисляется значение статистики для идеальной и сгенерированной последовательности и вероятность P-value того, что исследуемый генератор создал последовательность не менее случайную, чем идеальный Принимается решение: если P-value больше ошибки 1-го рода, то исследуемая последовательность считается случайной и наоборот в противном случае

# Пример: NIST Statistical Test Suite (1-2 тест)

- Частотный тест (монобитный тест на частоту, Frequency (Monobits) Test). В этом тесте исследуется доля 0 и 1 в последовательности и насколько она близка к идеальному варианту – равновероятной последовательности. Для теста надо иметь не менее 100 бит данных
- Блочный тест на частоту (Test for Frequency within a Block). Последовательность разбивается на блоки длиной M бит, и для каждого рассчитывается частота появления единиц и насколько она близка к эталонному значению – M/2. Длина тестовой последовательности не менее 100 бит, длина блока больше 20 бит

# Пример: NIST Statistical Test Suite (3-4 тест)

- Тест на серийность (Runs Test). В тесте находятся все серии битов – непрерывные последовательности одинаковых битов – и их распределение сравнивается с ожидаемым распределением таких серий для случайной последовательности. Длина последовательности 100 и более бит
- Тест на максимальный размер серии единиц (Tests for the Longest-Run-of-Ones in a Block). Исследуется длина наибольшей непрерывной последовательности единиц и сравнивается с длиной такой цепочки для случайной последовательности. Длина последовательности 100 и более бит