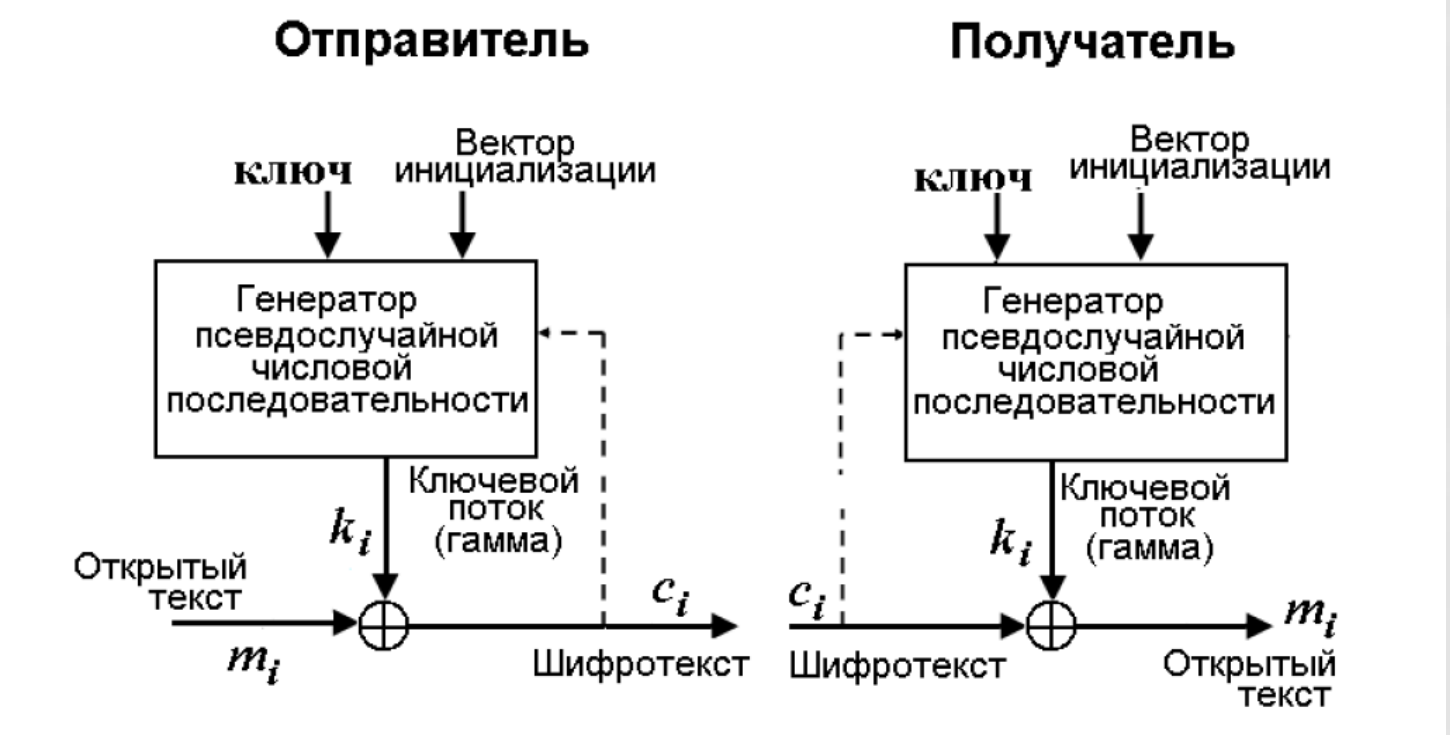
Поточные шифры и генераторы случайных последовательностей

Поточный шифр

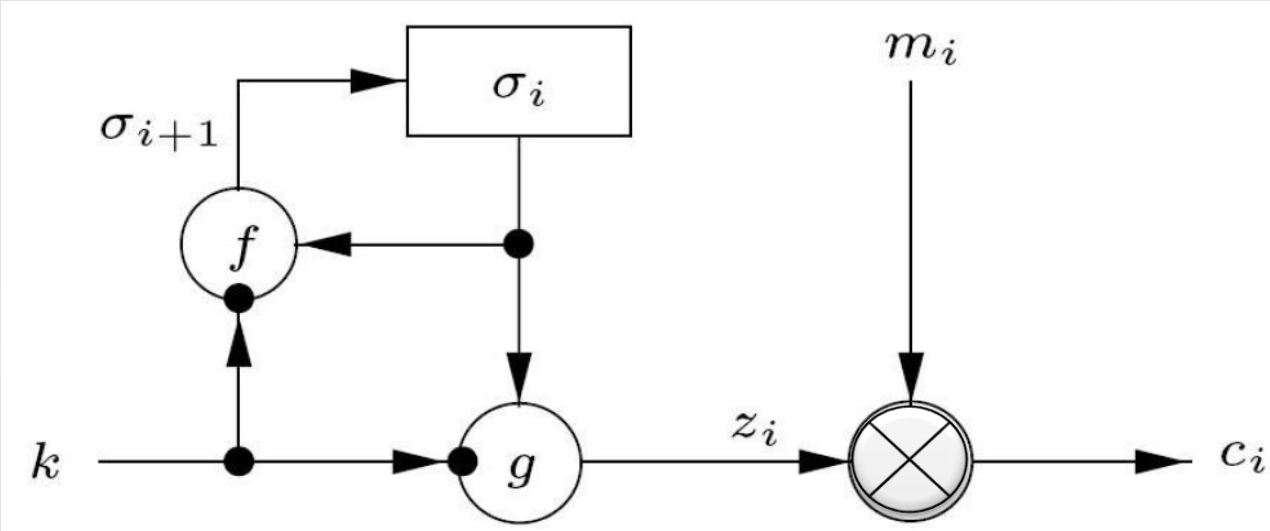


- Поточные шифры (*stream cipher*) обрабатывают сообщение, как поток битов и выполняют математические функции над каждым битом отдельно
- Поточные шифры используют генератор ключевого потока (гаммы), который производит поток битов, объединяемых с помощью операции XOR с битами открытого текста
- Различают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

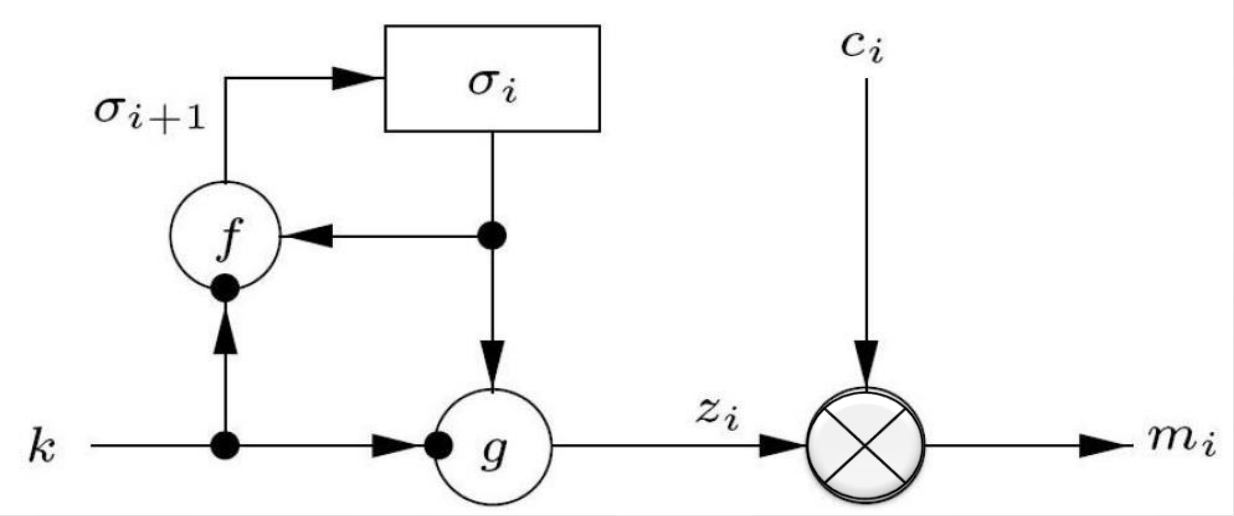
Синхронные поточные шифры

- В синхронных поточных шифрах гамма генерируется независимо от сообщения и шифровки
- Такое зашифрование формализуется следующим образом:
  - σ_{i+1}=f(σ_i, k)- функция перехода
  - z_i=g(σ_i, k)- генератор гаммы
  - c_i= $z_i \otimes m_i$- функция зашифрования
  - σ_i – состояние
  - k – секретный ключ

Синхронное зашифрование



и расшифрование



Свойства синхронных поточных шифров

- ***Требования по синхронизации:*** получатель и отправитель должны быть синхронизированы - т.е. вырабатывать одинаковые значения ключевого потока для соответствующих знаков передаваемого потока данных
- ***Отсутствие размножения ошибок:*** - изменение знака шифротекста при передаче не вызывает ошибок при расшифровании других знаков шифротекста
- ***Возможности активной атаки:*** любая вставка или удаление символа в шифротексте активным противником с высокой вероятностью будут замечены получателем

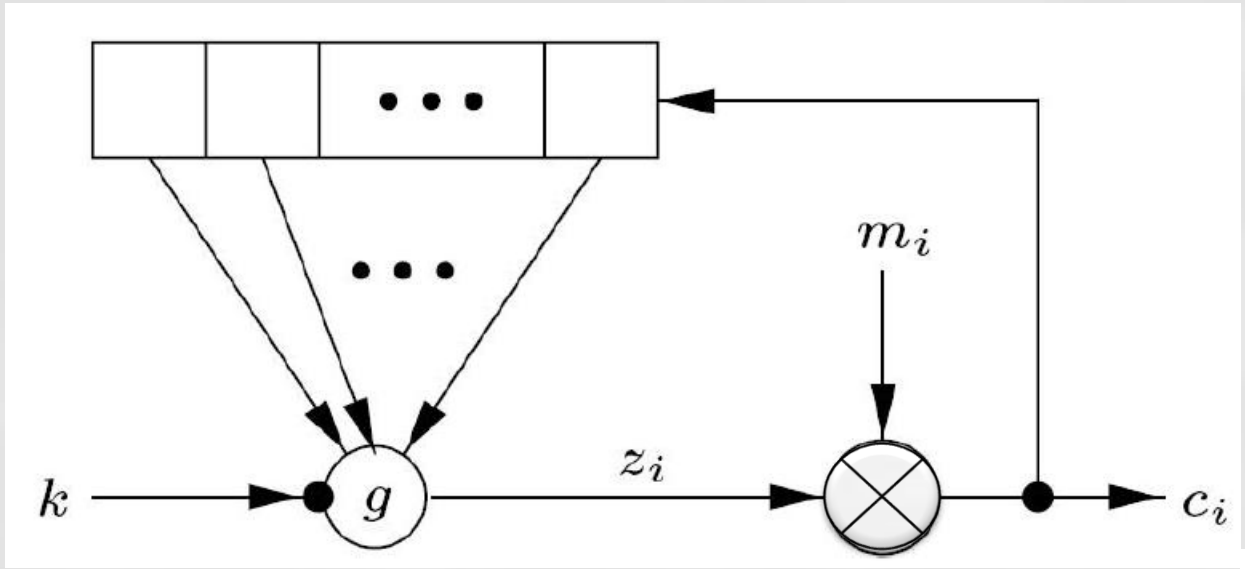
Атак с помощью вставки

- Пусть шифровка $c_i = z_i \otimes m_i$, i=1, N перехвачена нарушителем
- Нарушителю удалось вставить $m^{"}$ в сообщение после m_{t-1} и зашифровать на той же гамме $y_t^{"} = z_t \otimes m^{"}$
- Тогда:
  - $z_t = y_t^{"} \otimes m^{"}$ и находим $m_t = z_t \otimes y_t$
  - $z_{t+1} = y_{t+1}^{"} \otimes m_t$ и находим $m_{t+1} = z_{t+1} \otimes y_{t+1}$ и так далее

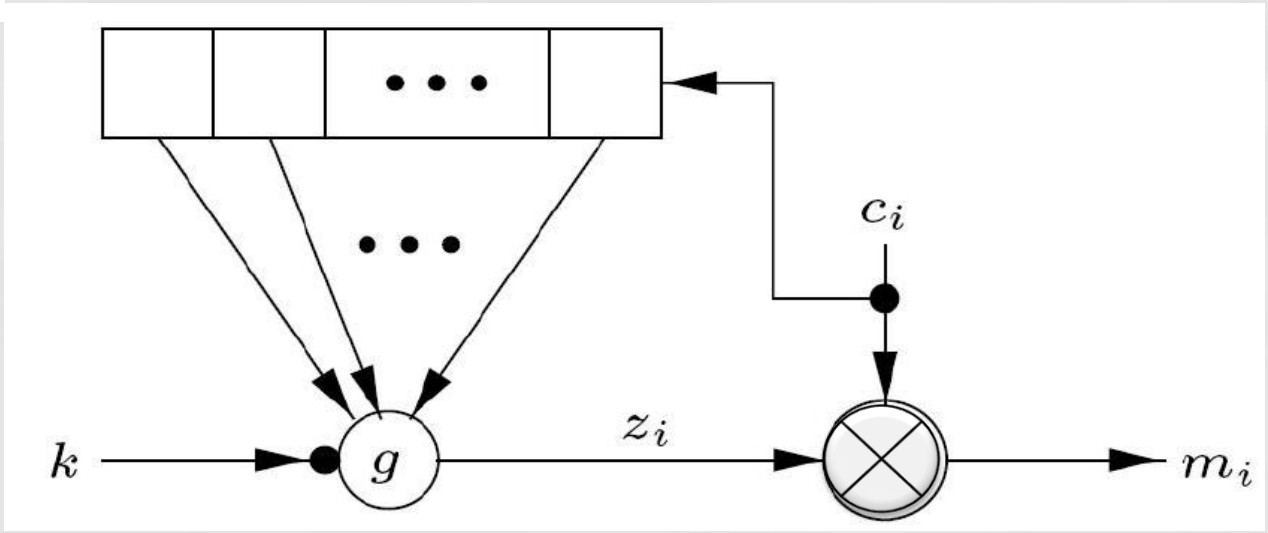
Самосинхронизирующиеся шифры

- В самосинхронизирующиеся (асинхронных) поточных шифрах гамма является функцией сообщения и нескольких предшествующих байтов (битов) шифровки
- Такое зашифрование формализуется следующим образом:
  - σ_i=f($c_{i-t}, c_{i-t+1}, \dots, c_{i-1}$)- функция перехода
  - z_i=g(σ_i, k)- генератор гаммы
  - c_i= $z_i \otimes m_i$- функция зашифрования
  - σ_i – состояние
  - k – секретный ключ

Самосинхронизирующееся зашифрование



и расшифрование



Свойства самосинхронизирующихся шифров

- ***Самосинхронизация:*** поскольку процесс расшифрования зависит от некоторого фиксированного числа предшествующих знаков шифротекста, то исчезновение при передаче знака из шифротекста сначала приведет к ошибкам, а затем все восстановится
- ***Ограниченное размножение ошибок:*** если во время передачи один знак шифротекста был изменен или удален/вставлен, то при расшифровке будет искажено не более t знаков

Свойства ... шифров (продолжение)

- ***Возможность активной атаки:*** любое изменение знаков шифротекста активным противником с большей (по сравнению с синхронными шифрами) вероятностью будет замечено со стороны получателя. Однако в случае вставки или удаления знаков шифротекста это намного труднее обнаружить (по сравнению с синхронными шифрами)
- ***Рассеивание статистики открытого текста:*** поскольку каждый знак открытого текста влияет на весь последующий шифротекст, статистические свойства открытого текста не сохраняются в шифротексте.

Линейные конгруэнтные генераторы гаммы

- Это простейший генератор псевдослучайных чисел:
  - Z_i=(a*Z_{i-1} +b) mod c, где Z_0-начальное состояние, *a, b, c* – константы
- Последовательность определенная числами Z_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 Z_i
- Достоинство – высокая скорость генерации псевдослучайных чисел, недостаток простота восстановления последовательности по нескольким значениям

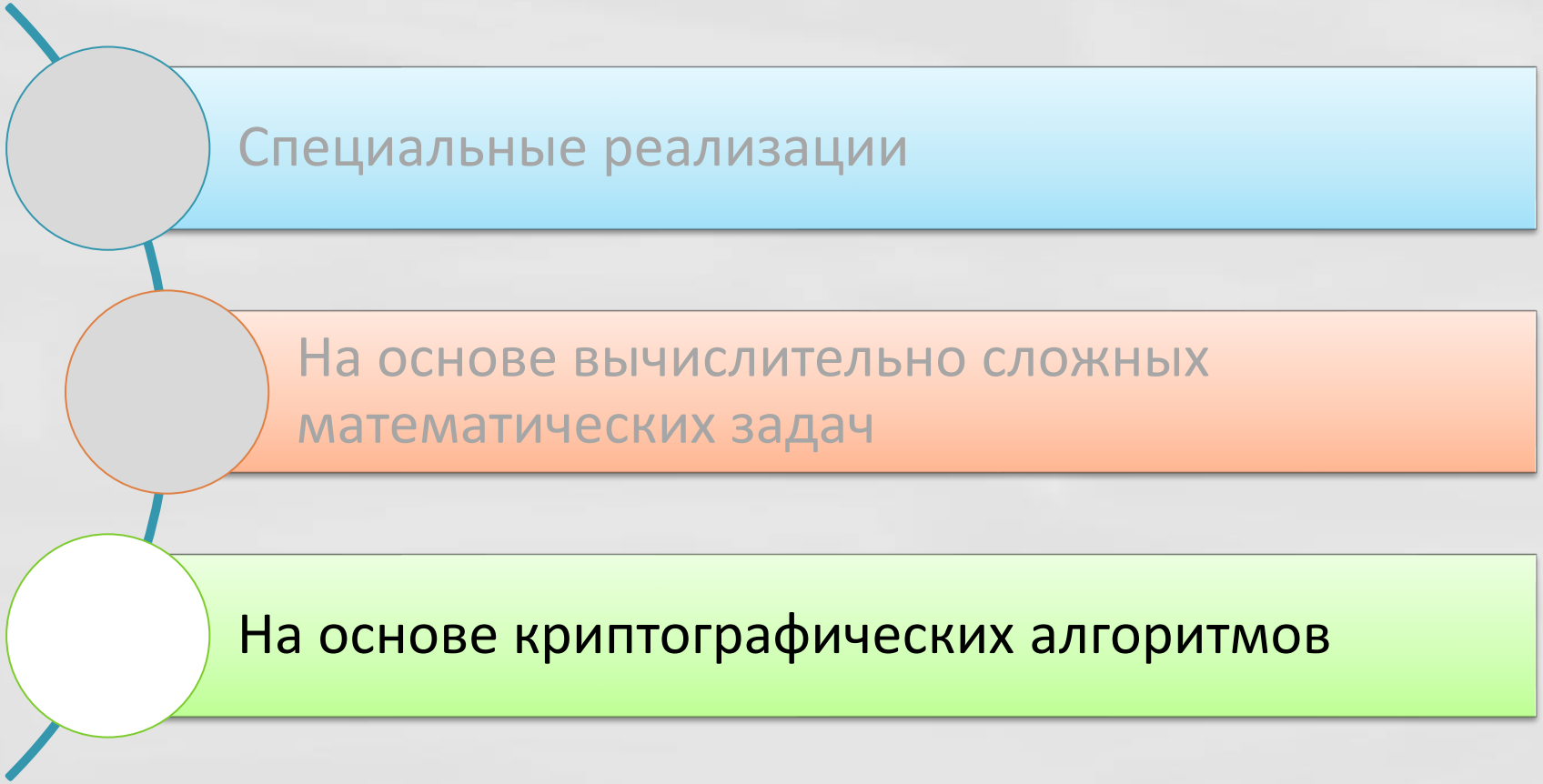
Пример: генератор RANDU

- $Z_{j+1} = (65539 \times Z_j) \mathrm{mod} 2^{31}$, Z_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 Z_{j+2} (подстановка и выполнение вычислений по $\mathrm{mod} 2^{31}$):
  - $Z_{j+2} = (2^{16}+3) \times Z_{j+1} == (2^{16}+3)^2 \times Z_j = (2^{32}+6 \times 2^{16}+9) \times Z_j = (2^{32}+6 \times (2^{16}+3)-9) \times Z_j$
  - Выполняем $Z_{j+2} \mathrm{mod} 2^{31}$ и получаем $Z_{j+2} = 6Z_{j+1} - 9Z_j$ - очень просто вычислить очередное значение по предыдущим

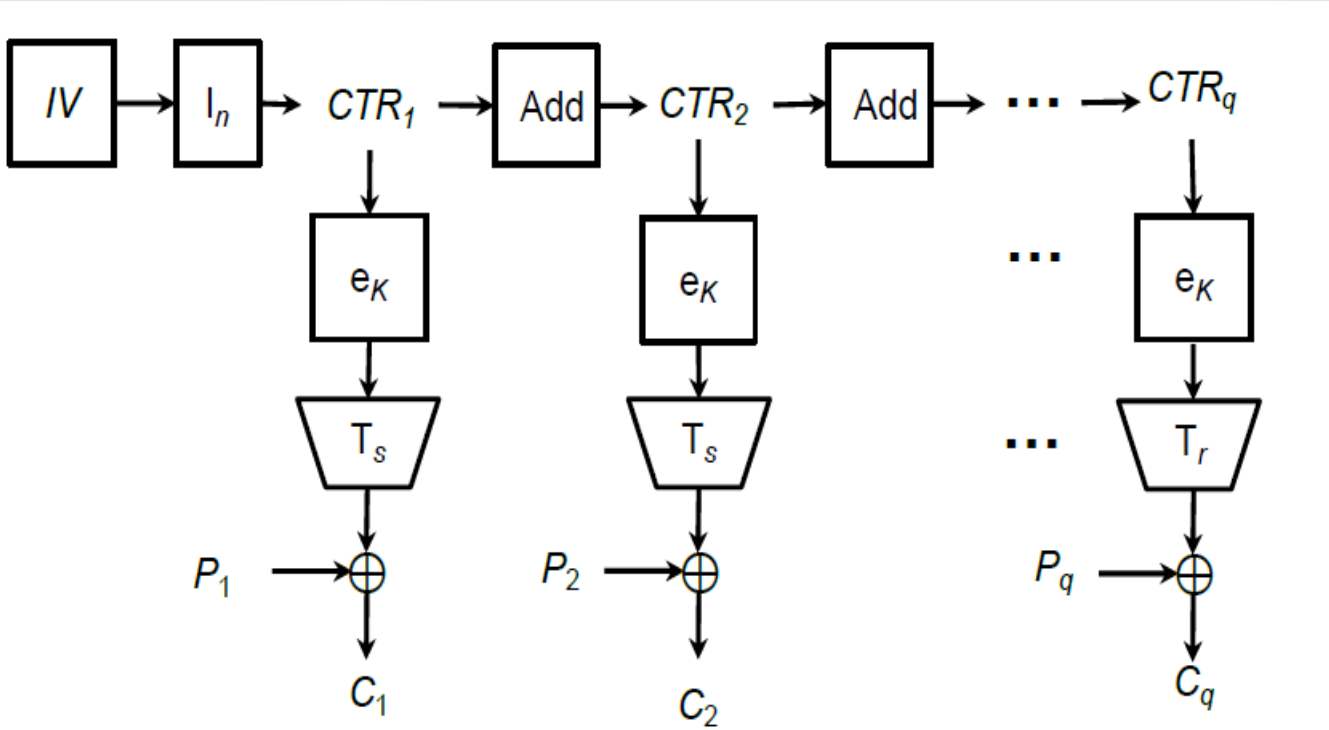
Требования к криптостойким генераторам гаммы

- 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

Способы реализации криптостойких генераторов



Режим гаммирования

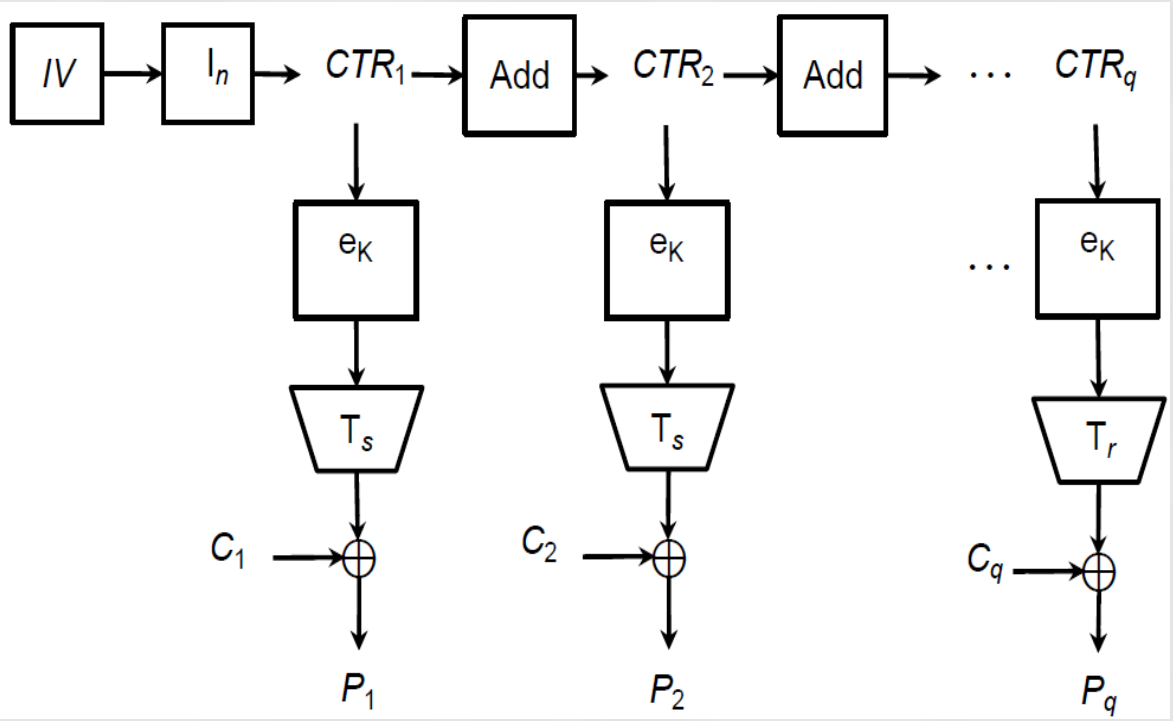


$$C_i = P_i \oplus T_s(e_K(CTR_i)), \quad i = 1, 2, \ldots, q-1,$$
$$C_q = P_q \oplus T_r(e_K(CTR_q)).$$

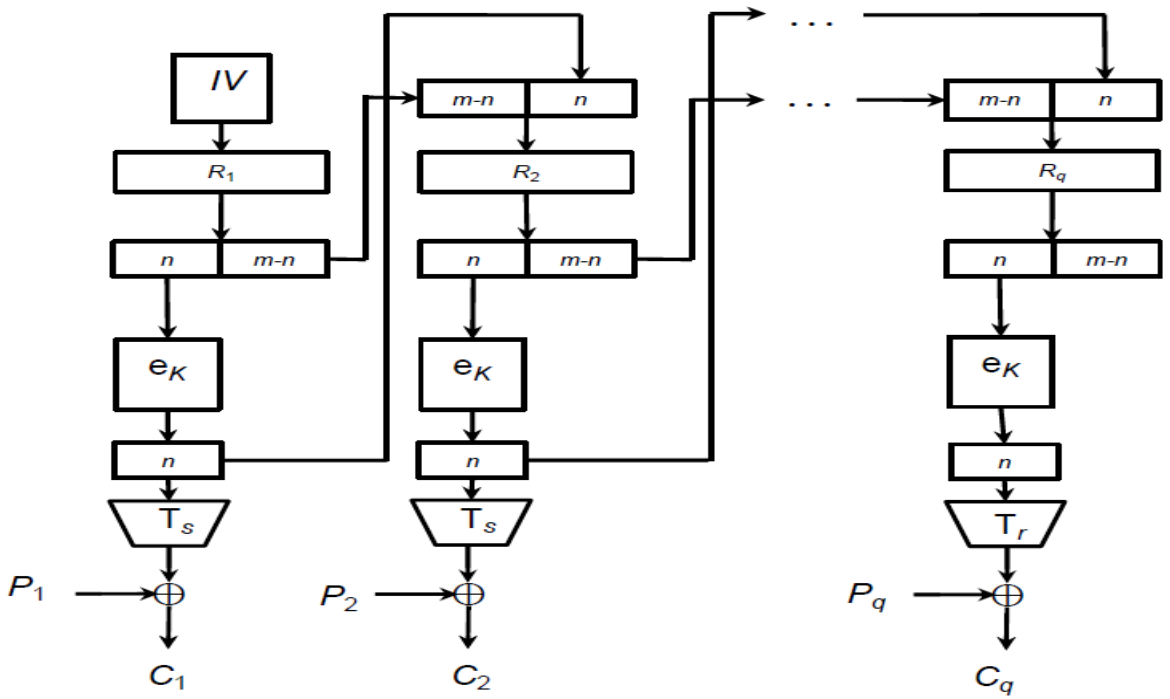
Counter, CTR

Для CRT свойства распространения ошибки такие же, как и синхронных шифров

$$P_i = C_i \oplus T_s(e_K(CTR_i)),$$
$$P_q = C_q \oplus T_r(e_K(CTR_q)).$$



Режим гаммирования с обратной связью по выходу



$R_1 = IV,$

$$\begin{cases} Y_i = e_K(\mathrm{MSB}_n(R_i)), \\ C_i = P_i \oplus T_s(Y_i), \\ R_{i+1} = \mathrm{LSB}_{m-n}(R_i) \| Y_i, \end{cases} \quad i = 1, 2, \ldots, q\text{-}1,$$

$Y_q = e_K(\mathrm{MSB}_n(R_q)),$

$C_q = P_q \oplus T_r(Y_q).$

Output Feedback, OFB

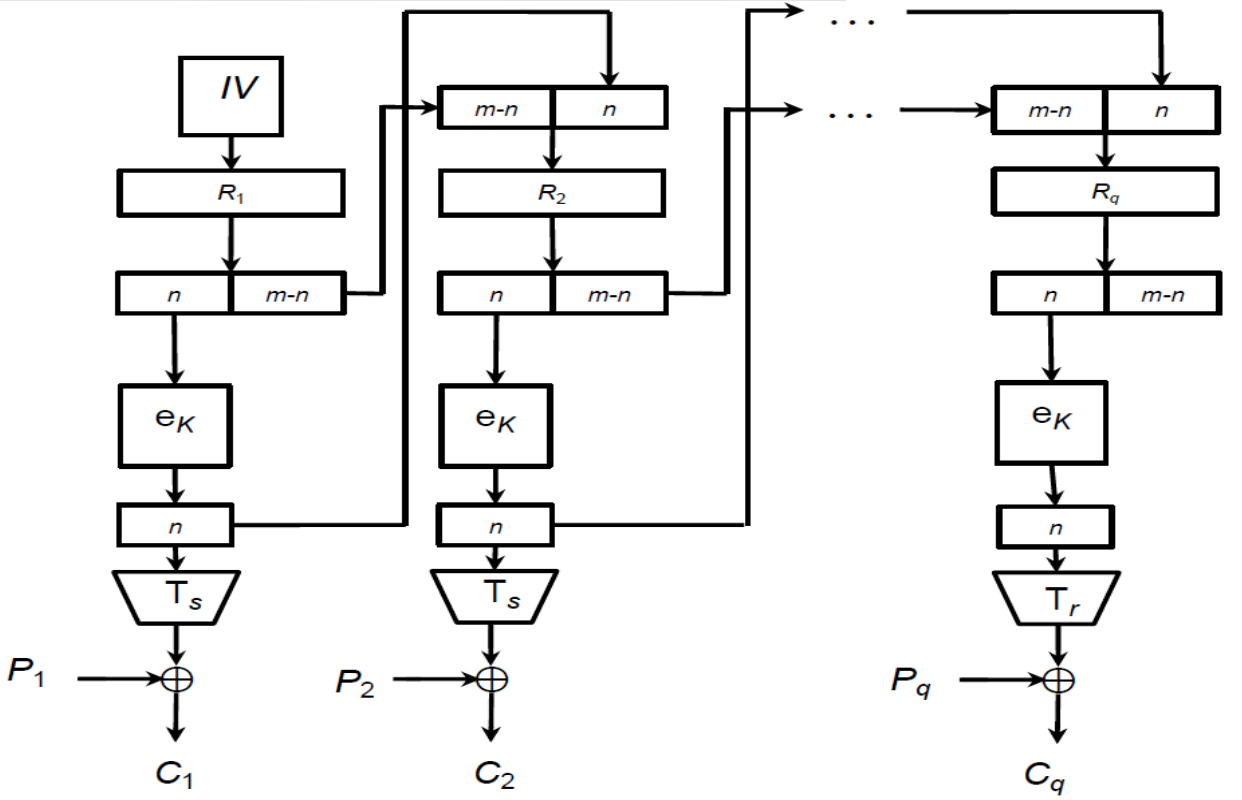
OFB чаще всего используется в высокоскоростных синхронных системах, где недопустимо распространение ошибки

$R_1 = IV,$

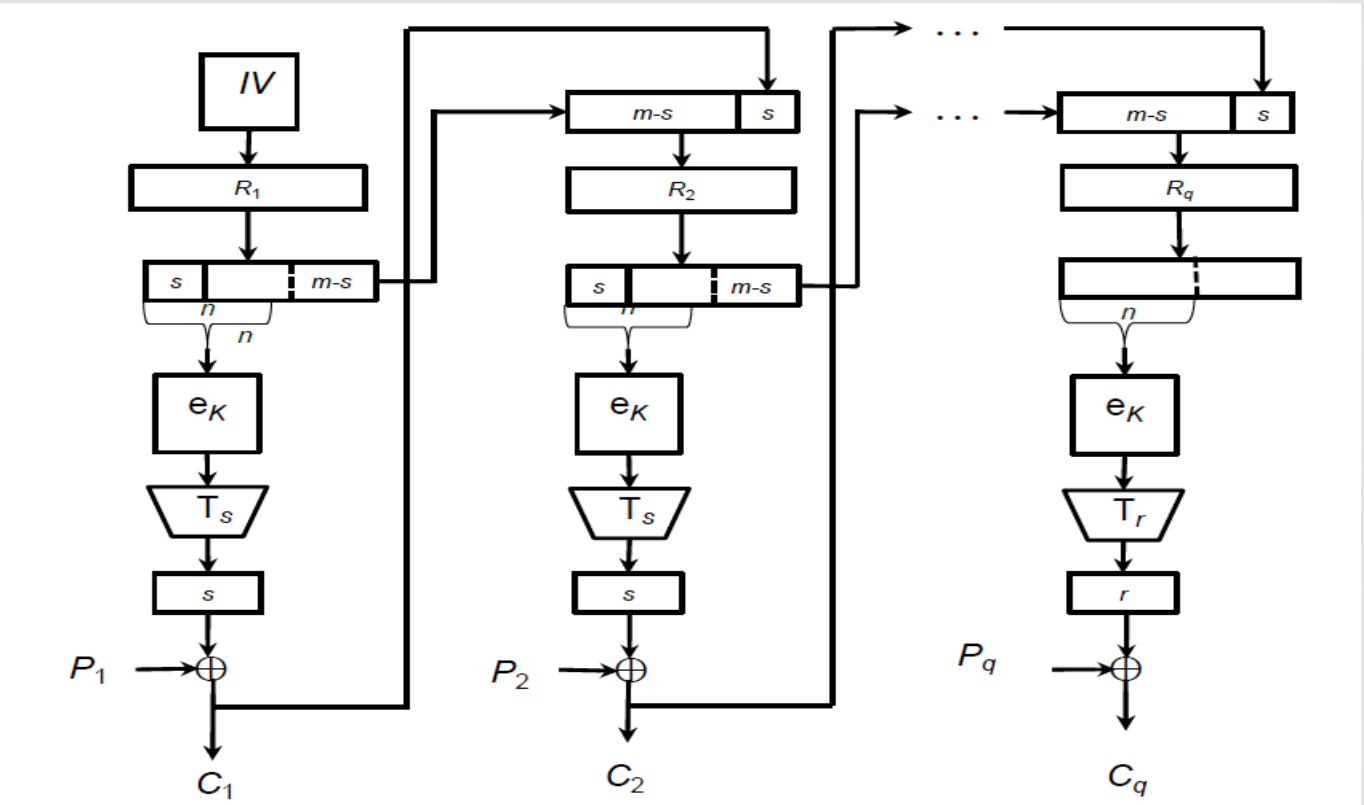
$$\begin{cases} Y_i = e_K(\mathrm{MSB}_n(R_i)), \\ P_i = C_i \oplus T_s(Y_i), \\ R_{i+1} = \mathrm{LSB}_{m-n}(R_i) \| Y_i, \end{cases}$$

$Y_q = e_K(\mathrm{MSB}_n(R_q)),$

$P_q = C_q \oplus T_r(Y_q)$



Режим гаммирования с обратной связью по шифротексту



$R_1 = IV,$

$$\begin{cases} C_i = P_i \oplus T_s(e_K(MSB_n(R_i))), \\ R_{i+1} = LSB_{m-s}(R_i) \| C_i, \end{cases} \quad i = 1, 2, \ldots, q-1,$$

$$C_q = P_q \oplus T_r(e_K(MSB_n(R_q))).$$

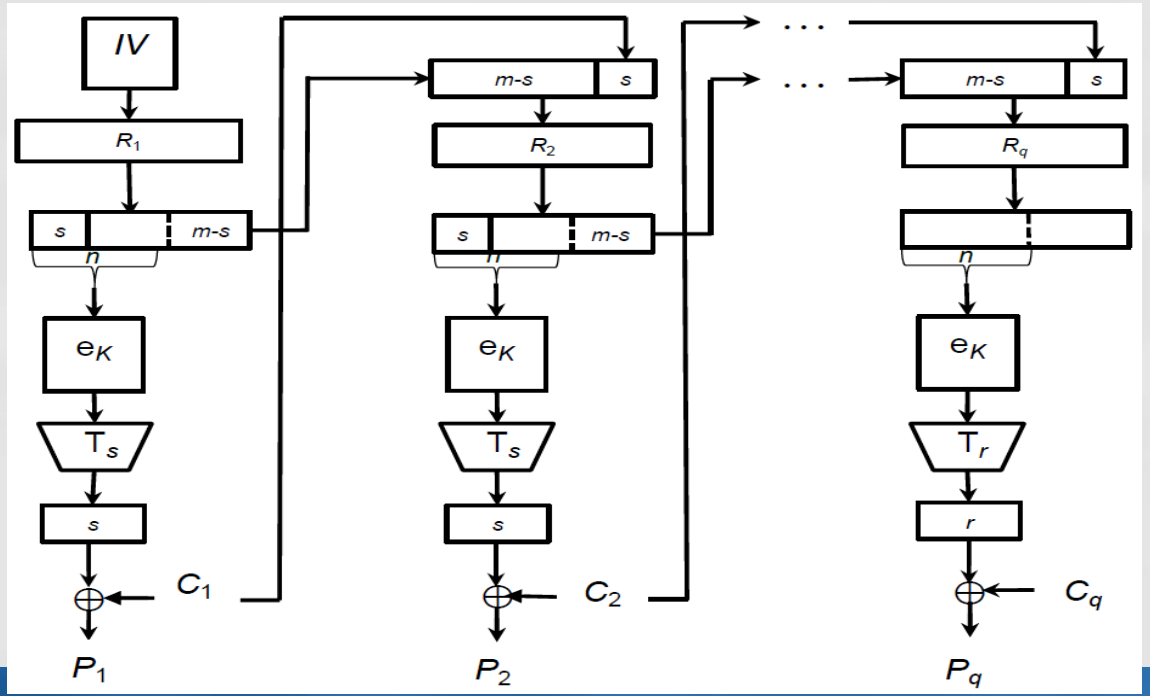
Cipher Feedback, CFB

CFB обычно выбирается для помехоустойчивого шифрования потока символов, когда каждый символ может рассматриваться отдельно, как в линии связи между терминалом и главным компьютером

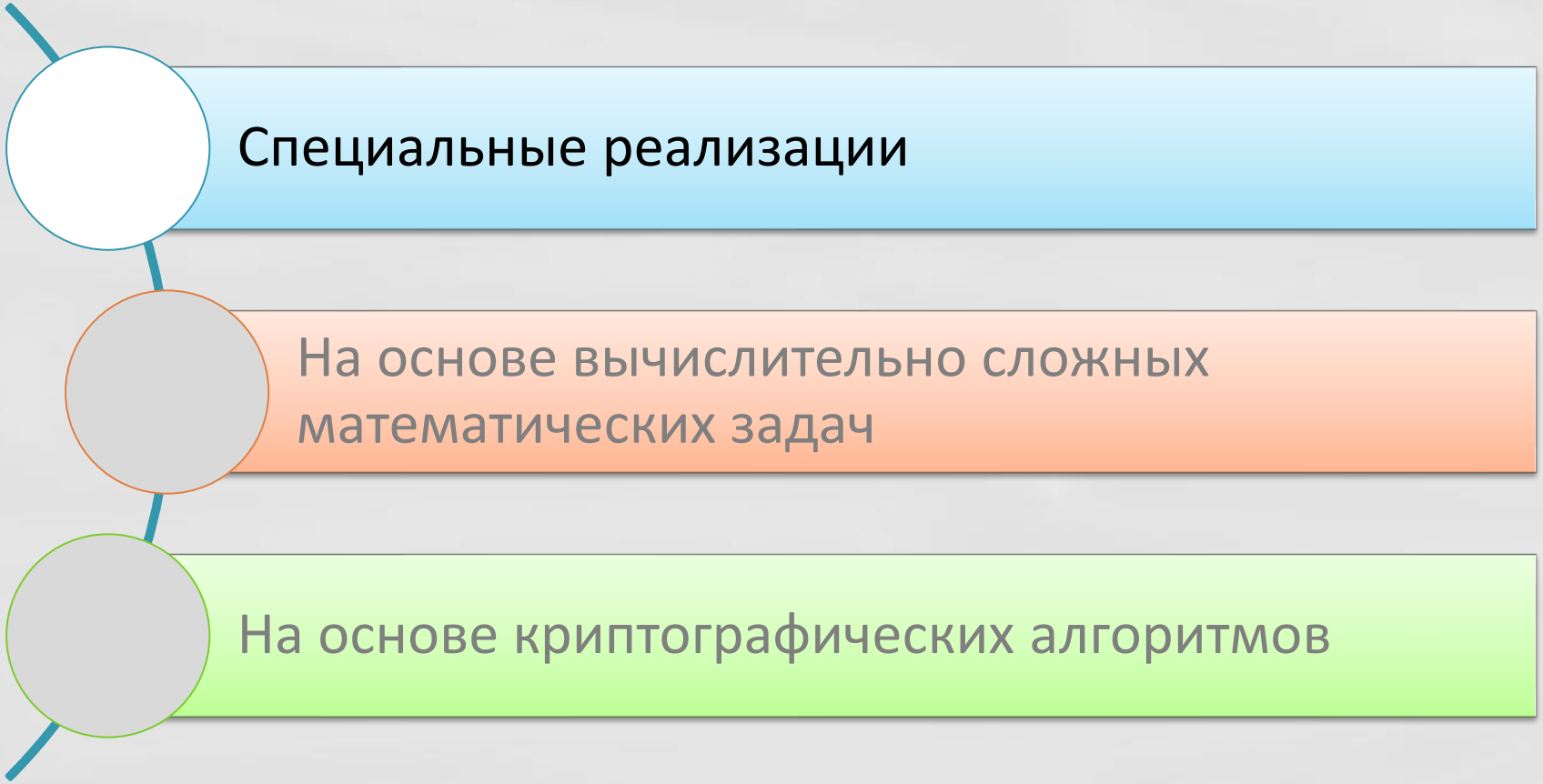
$R_1 = IV,$

$$\begin{cases} P_i = C_i \oplus T_s(e_K(MSB_n(R_i))), \\ R_{i+1} = LSB_{m-s}(R_i) \| C_i, \end{cases}$$

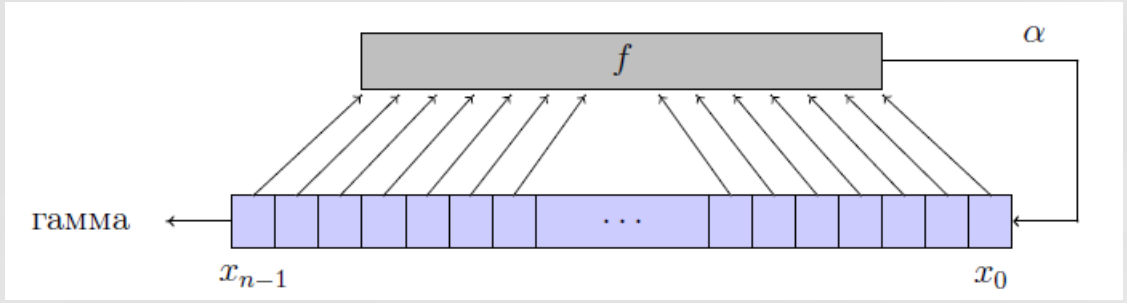
$$P_q = C_q \oplus T_r(e_K(MSB_n(R_q))).$$



Способы реализации криптостойких генераторов

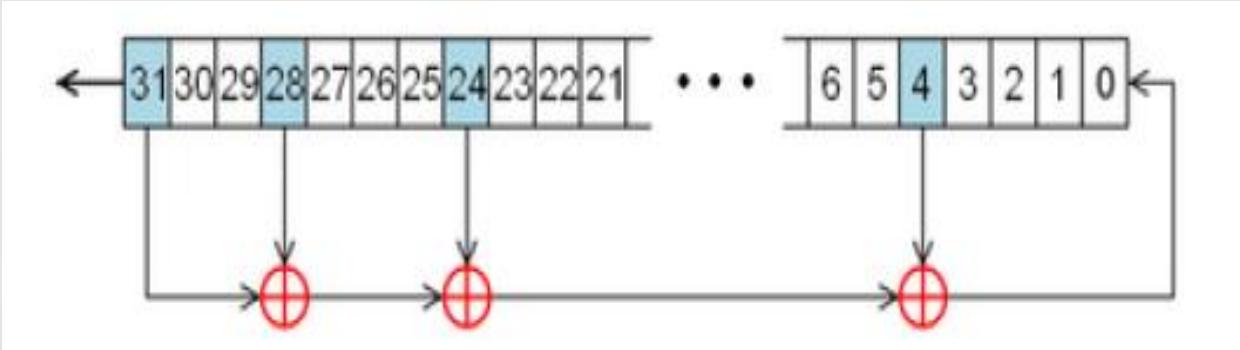


Сдвиговые регистры с обратной связью



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

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

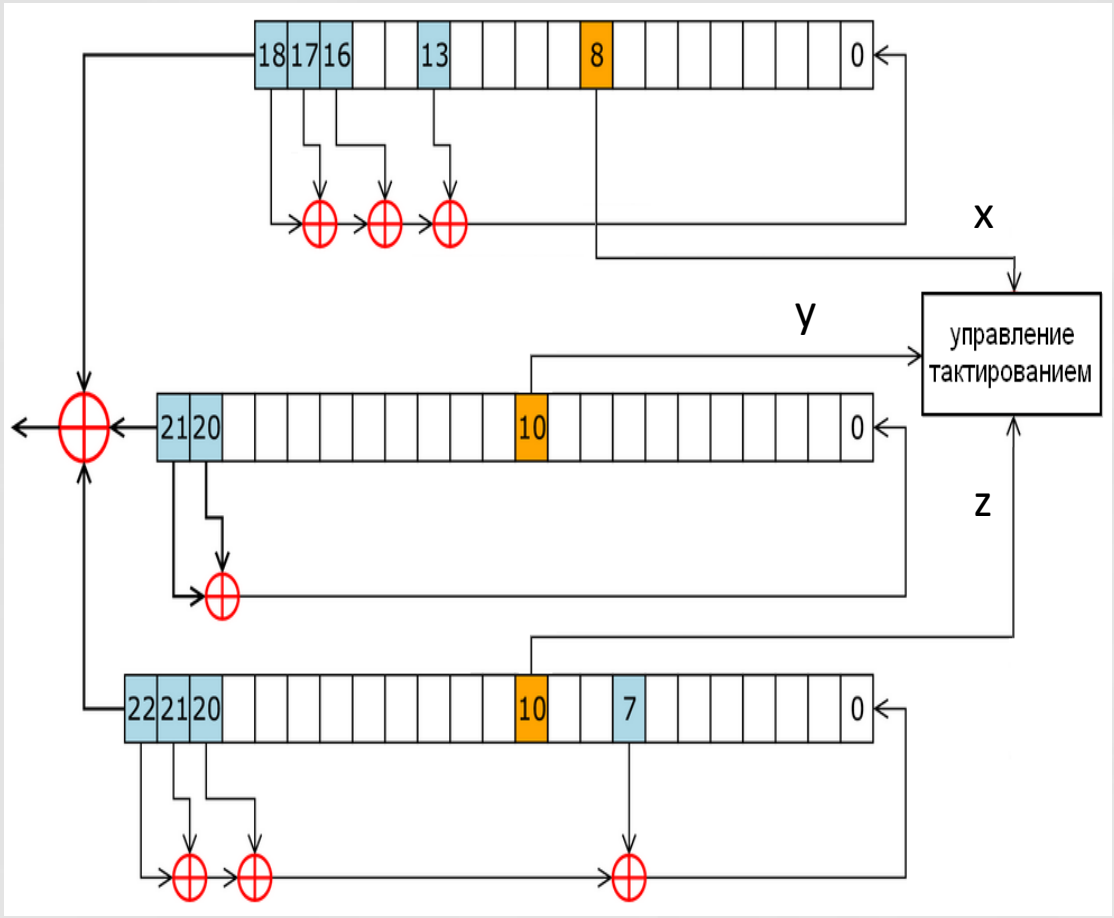


Linear Feedback Shift Register

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

Корреляционная атака

- Цель: найти начальное состояние сдвигового регистра (целевого), участвующего в формировании криптографической гаммы
- Модель нарушителя:
  - Знает устройство шифра на регистрах: количество задействованных регистров, примитивный полином, задающий номера отводов каждого для формирования обратной связи, функцию выхода генератора
  - Знает начальный фрагмент исходного сообщения, подлежащего зашифрованию
  - Умеет перехватывать и сохранять шифрованный поток

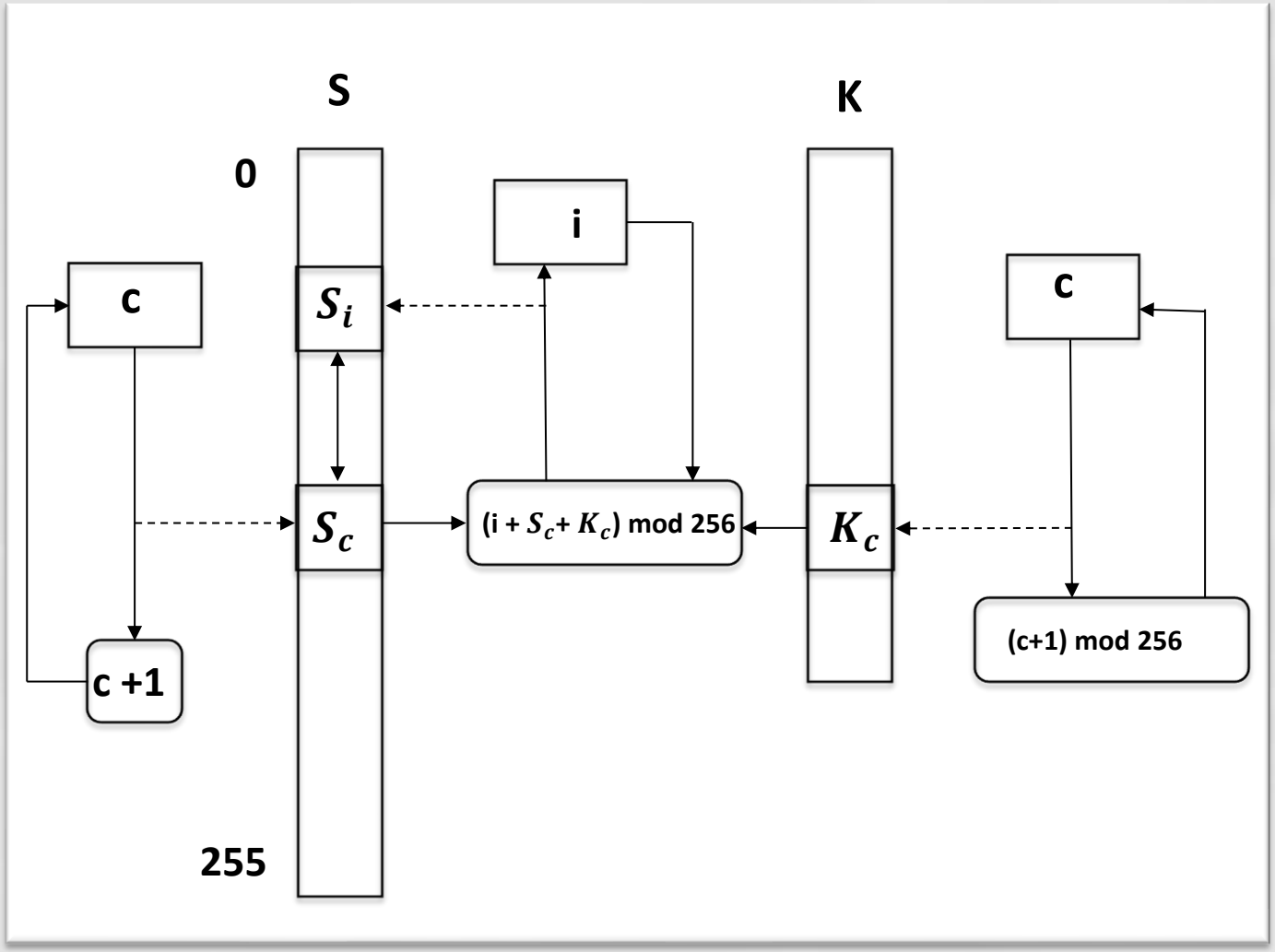
Тактика атакующего

- Обладая исходным текстом и шифровкой, восстанавливает начальный фрагмент гаммы $z_i = c_i \otimes m_i$, i=1, N
- Зная количество регистров, устройство обратной связи каждого и функцию выхода генератора, можно построить таблицу со всеми комбинациями выходов регистров и соответствующих выходов генератора $\{R_1, R_2, \ldots, R_n, Z\}$
- Выбрать целевой регистр R_T, выход которого в наибольшей степени коррелирует (не менее 50% совпадений) со значениями выхода Z генератора
- Перебором начального состояния целевого регистра вычислить выход регистра и оценить корреляцию с уже известным фрагментов гаммы
- Выбрать начальное состояние с наибольшей корреляцией

Потоковый шифр RC4

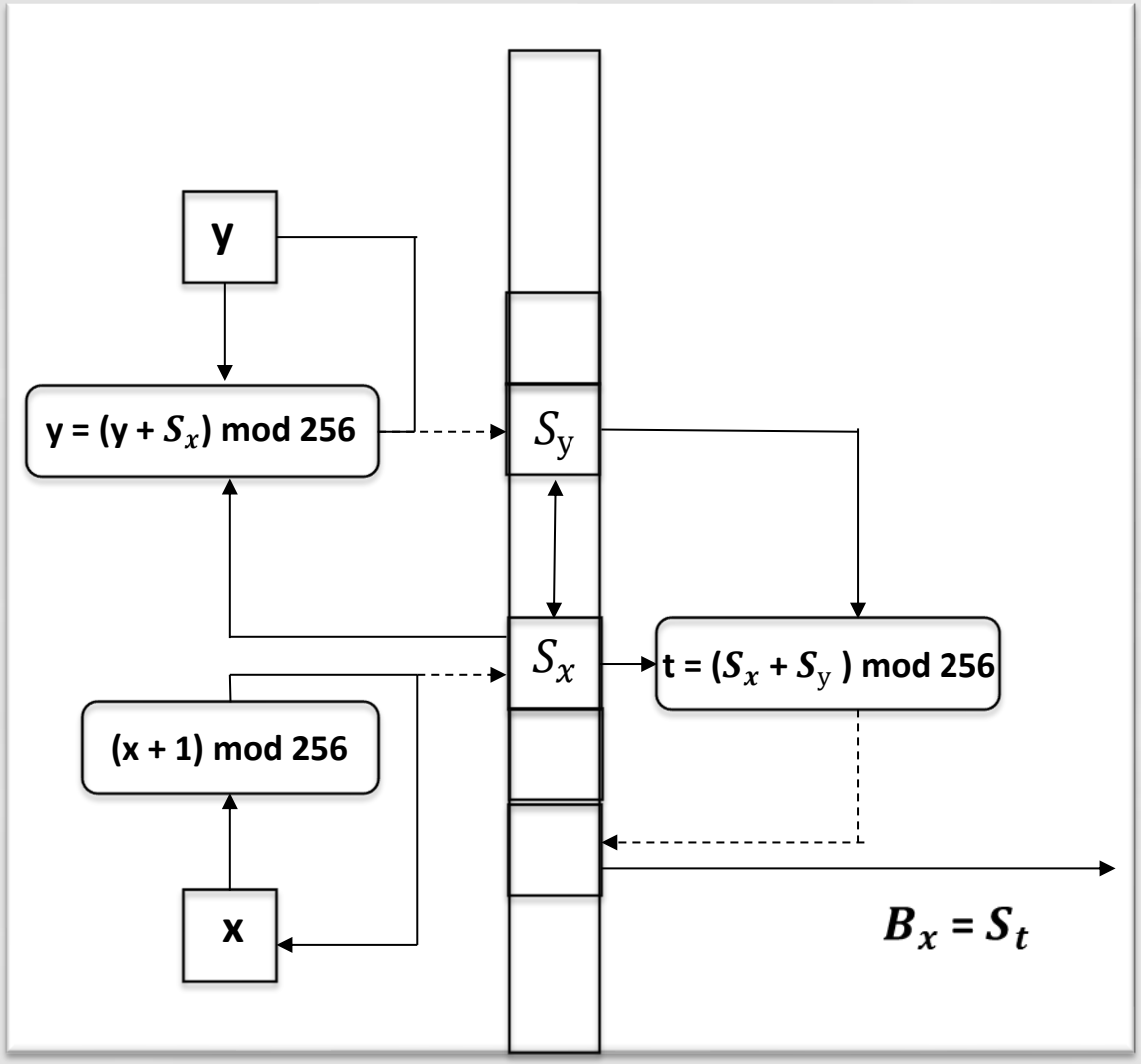
- RC4 — потоковый шифр, разработан в 1984 г. Используется во многих сетевых протоколах, например, SSL/TLS и IEEE 802.11 (стандарт беспроводных сетей)
- Это байт-ориентированный шифр потока, в котором каждый байт исходного текста складывается (XOR) с байтом ключа
- Секретный ключ, на основе которого сгенерированы однобайтовые ключи в потоке ключей, может быть переменной длины (от 5 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. $B_x = S_t$

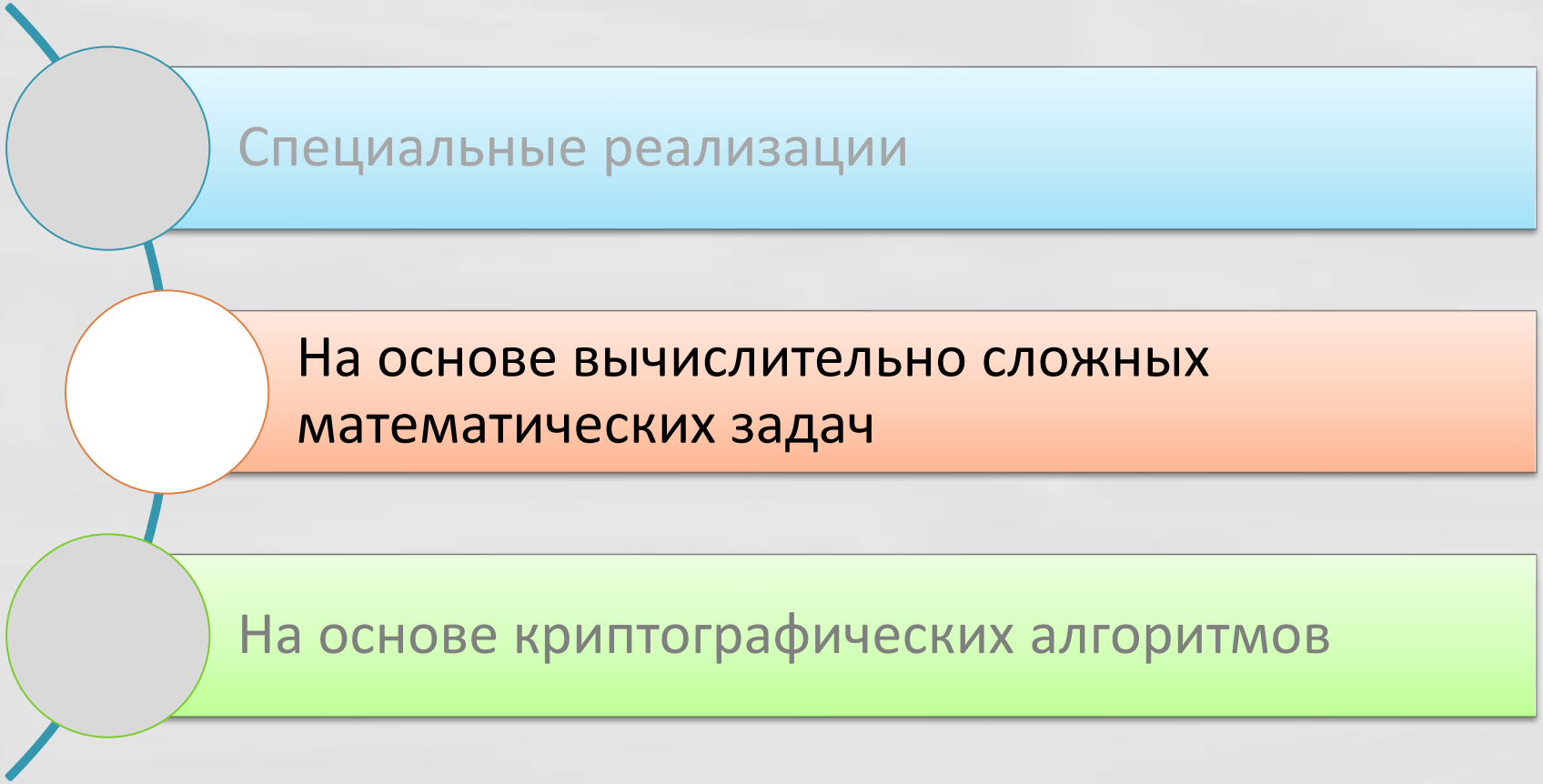
Пример криптоанализа RC4 (2013)

- 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

Способы реализации криптостойких генераторов



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

Спасибо за внимание !