Поточные шифры

Лекция №3

Шифр Вермана

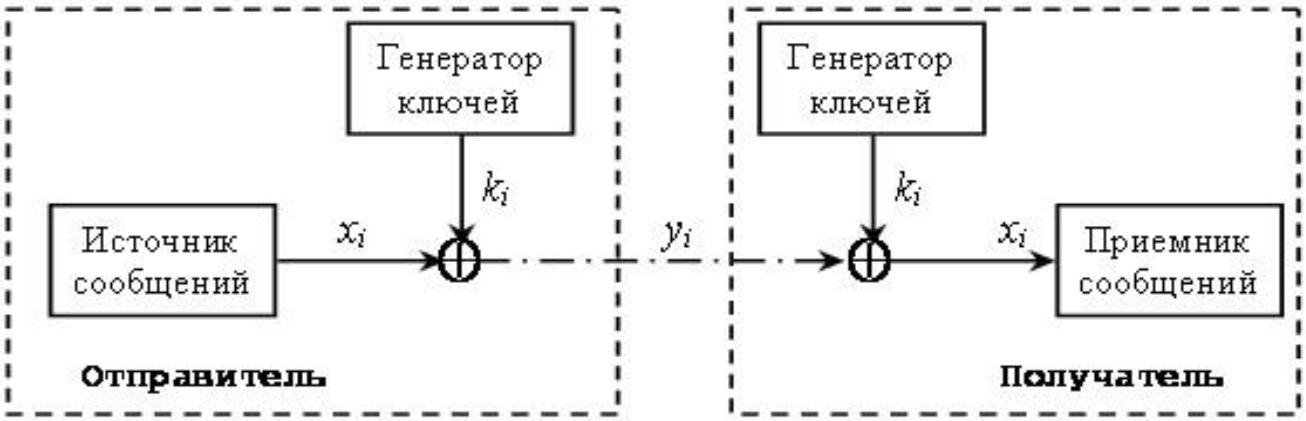
Типичным примером реализации абсолютно стойкого шифра является шифр, который осуществляет побитовое сложение n-битового открытого текста и n-битового ключа:

$$y_i = x_i \oplus k_i, \quad i = 1, \ldots, n.$$

Для абсолютной стойкости необходимо чтобы было выполнено три условия:

1. **Полная случайность** (равновероятность) ключа (это, в частности, означает, что ключ нельзя вырабатывать с помощью какого-либо детерминированного устройства)
2. **Равенство** длины ключа и длины открытого текста
3. **Однократность** использования ключа

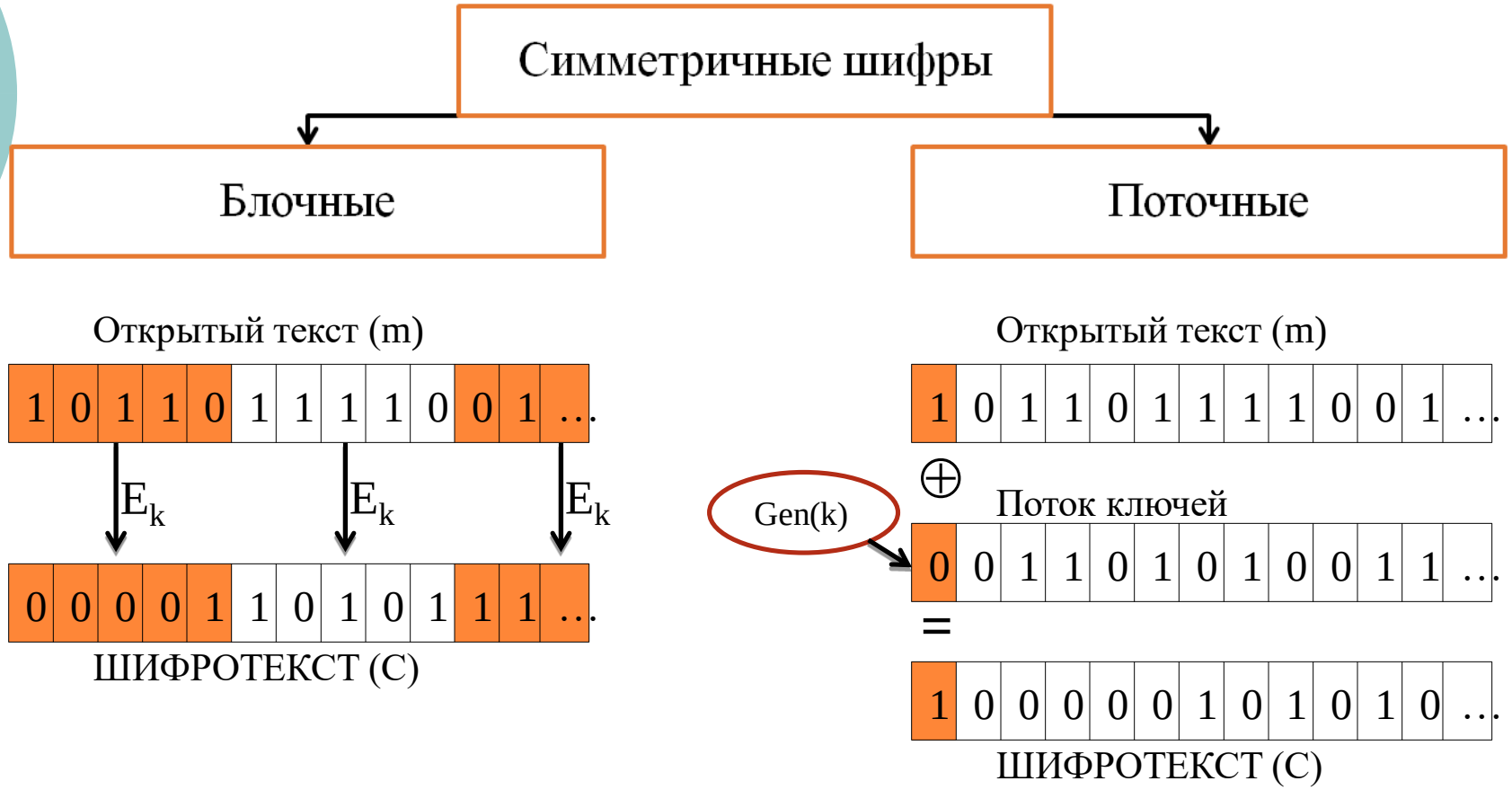
Работа поточного шифра



Шифрование: $y_i = x_i \oplus k_i, i = 1, 2, \ldots, n$

Дешифрование: $x_i = y_i \oplus k_i, \ i = 1, 2, \ldots, n$

Блочные и поточные шифры



Основные понятия

- **Случайное число** – число, представляющее собой реализацию случайной величины
- **Детерминированный алгоритм** – алгоритм, который возвращает те же выходные значения при тех же входных значениях
- **Псевдослучайное число** – число, полученное детерминированным алгоритмом, используемое в качестве случайного числа
- **Физическое случайное число (истинно случайное)** – случайное число, полученное на основе некоторого физического явления
- **Генератор псевдослучайных числе (ГСПЧ)** — это устройство или алгоритм, который выдает последовательность статистически независимых и несмещенных чисел (то есть подчиняющихся закону распределения)

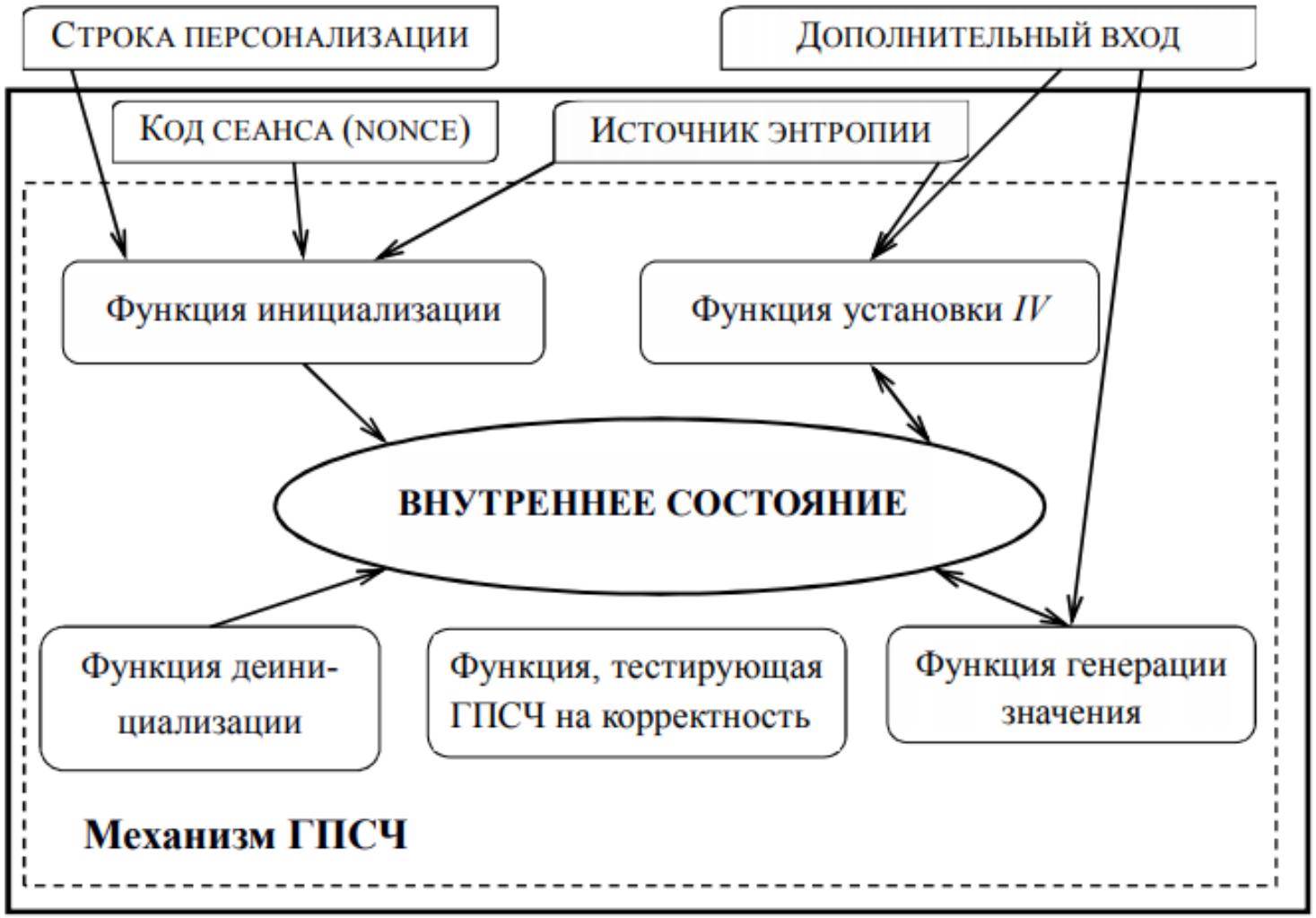
Отличие ГСЧ от ГПСЧ

Источники энтропии используются для накопления энтропии с последующим получением из неё начального значения (initial value, seed), необходимого генераторам случайных чисел (ГСЧ) для формирования случайных чисел. **ГПСЧ использует единственное начальное значение**, откуда и следует его псевдослучайность, а ГСЧ всегда формирует случайное число, имея в начале высококачественную случайную величину, предоставленную **различными источниками энтропии**.

Энтропия – это мера беспорядка. Информационная энтропия — мера неопределённости или непредсказуемости информации.

Можно сказать, что **ГСЧ = ГПСЧ + источник энтропии**

Состав генератора ПСЧ

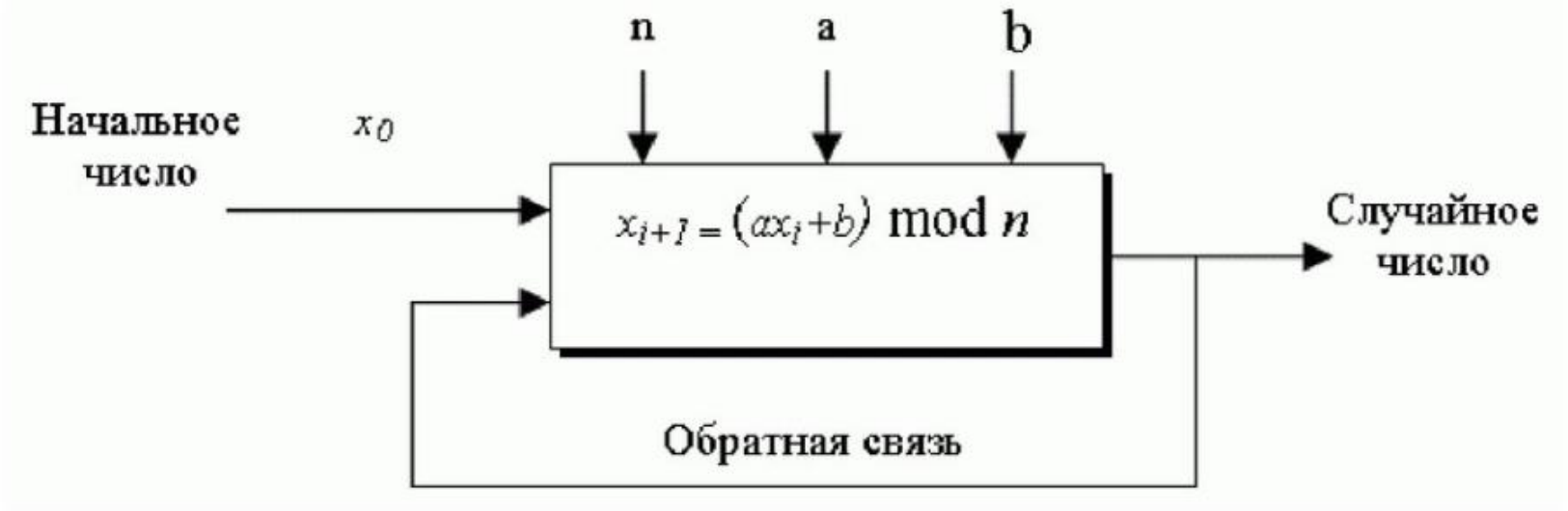


Свойства ГПСЧ

- Период последовательности должен быть очень большой
- Порождаемая последовательность должна быть "почти" неотличима от действительно случайной
- Вероятности порождения различных значений должны быть в точности равны
- Для того, чтобы только законный получатель мог расшифровать сообщение, следует при получении потока ключевых битов ki использовать и учитывать некоторый секретный ключ, причем вычисление числа ki+1 по известным предыдущим элементам последовательности ki без знания ключа должно быть трудной задачей

Конгруэнтные генераторы

Общая методика для генерирования псевдослучайных последовательностей – это **линейный конгруэтный рекурсивный** метод, предложенный Лехмером



X0 – начальное число (между 0 и n-1, n – модуль соотношения).

Последовательность периодическая (период зависит от a и b)

Рекомендации к выбору a,b,m

- Оптимальный выбор модуля **m** - это наибольшее простое число, близкое к размеру слова, используемого в компьютере. Рекомендуется использовать тридцать первое простое число Мерсенны как модуль:

$$m = M_{31} = 2^{31} - 1$$

- Чтобы создавать период, равный значению модуля, значение первого коэффициента, **a**, должно быть первообразным корнем главного модуля. Хотя целое число 7 - первообразный корень M_{31}, рекомендуют использовать 7^k, где k - целое число, взаимно-простое с (M_{31} - 1). Некоторые рекомендованные значения для k - это 5 и 13. Это означает, что (a = 75) или (a = 713)
- Для эффективного использования значение второго коэффициента **b** должно быть нулевым.

Влияние коэффициентов a,b,m

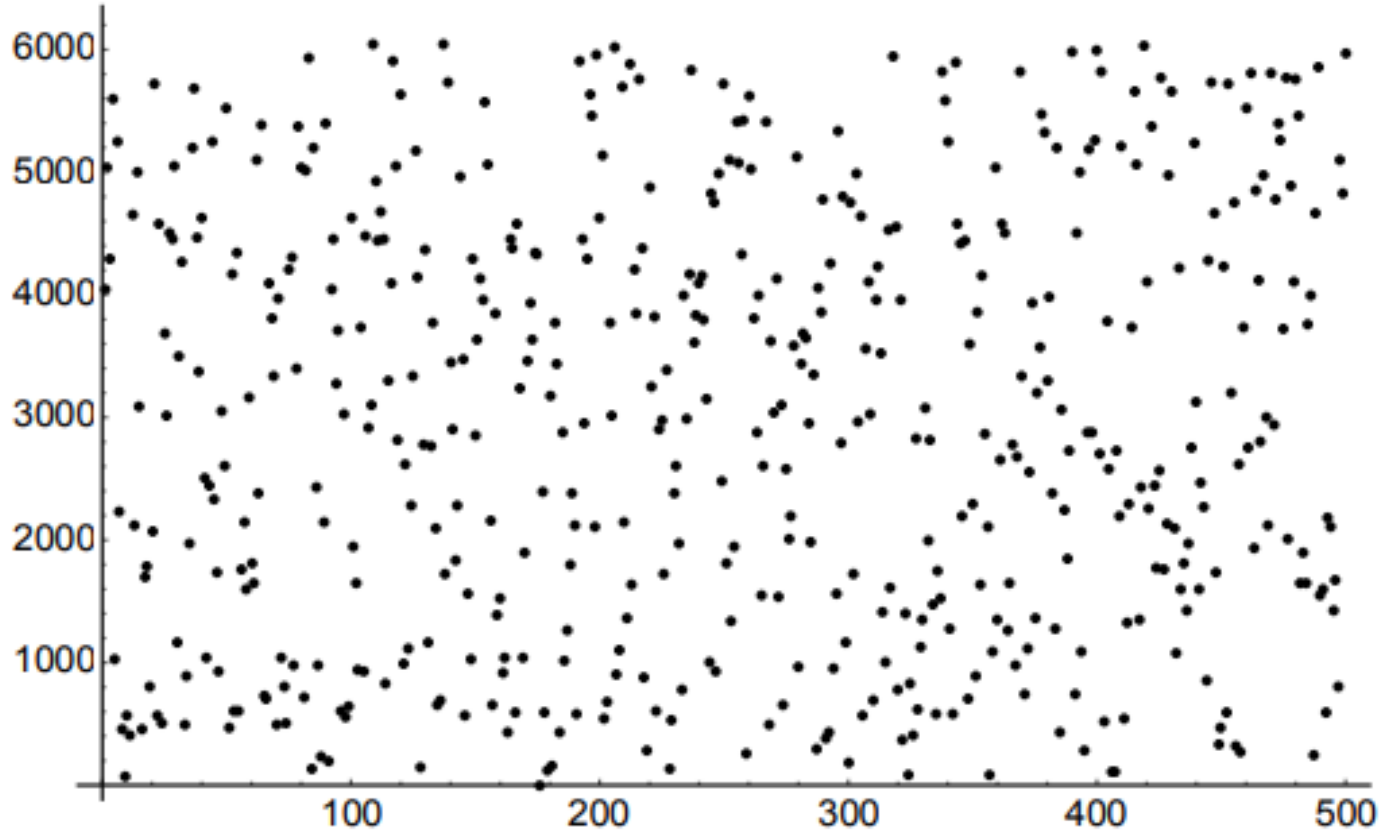


Рисунок 3.1. График ЛКП для $X_0 = 7, a = 106, c = 1283, m = 6075$.

Влияние коэффициентов a,b,m

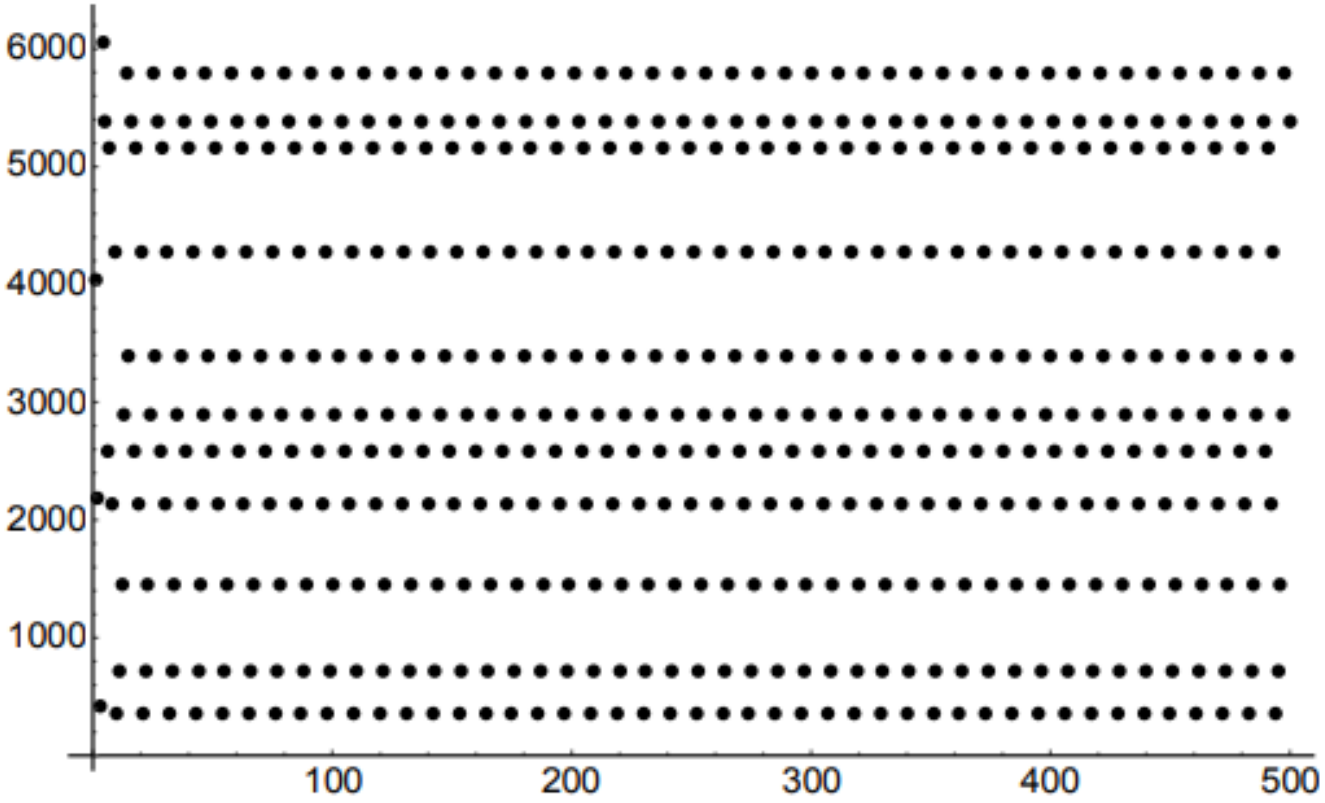


Рисунок 3.2. График ЛКП для $X_0 = 7, a = 105, c = 1283, m = 6075.$

Влияние коэффициентов a,b,m

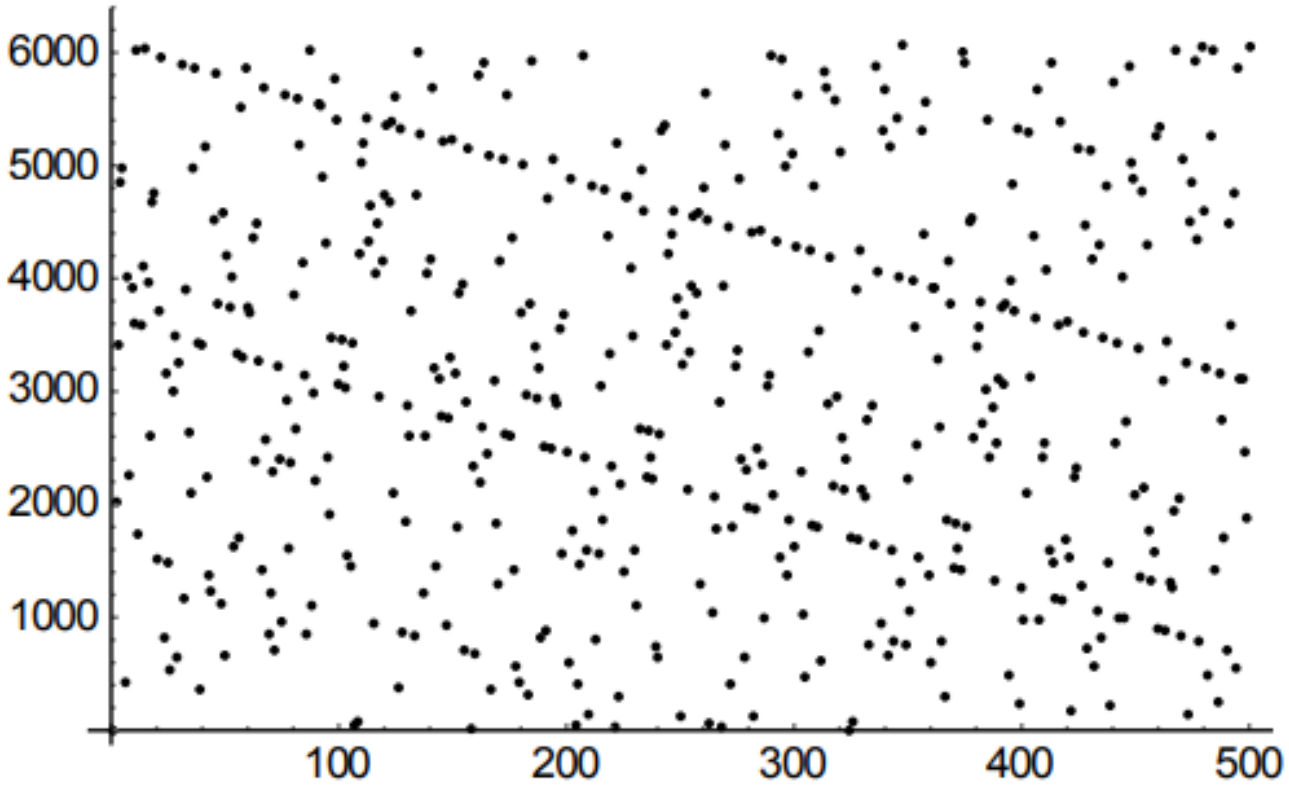


Рисунок 3.3. График ЛКП для параметров: $X_0 = 7, a = 106, c = 1284, m = 6075.$

Результаты КГ

$$\begin{cases} X_1 = (a \cdot X_0 + c) \mod m \\ X_2 = (a \cdot X_1 + c) \mod m \\ X_3 = (a \cdot X_2 + c) \mod m \end{cases}$$

Решив эту систему, можно определить коэффициенты a, c, m.

Основным алгоритмом предсказания чисел для линейно-конгруэнтного метода является Plumstead's — алгоритм, реализацию, которого можно найти здесь:

http://www.staff.uni-mainz.de/pommeren/Kryptologie99/English.html

Виды конгруэнтных генераторов

Линейный конгруэнтный генератор

$$X_n = (aX_{n-1} + b) \bmod m$$

Генератор Парка Миллера

$$X_n = (7^5 \cdot X_{n-1}) \bmod 2^{31} - 1$$

Квадратичный конгруэнтный генератор

$$X_n = (aX_{n-1}^2 + bX_{n-1} + c) \bmod m$$

Кубический конгруэнтный генератор

$$X_n = (aX_{n-1}^3 + bX_{n-1}^2 + cX_{n-1} + d) \bmod m$$

Суперпозиция КГ

Для увеличения размера периода повторения конгруэнтных генераторов часто используют их **объединение (суперпозицию)** посредством нелинейного преобразования (функции). При этом криптографическая безопасность не уменьшается, но такие генераторы обладают лучшими характеристиками в некоторых статистических тестах.

```
// Long  - 32-bit integer
static long s1 = 1;
static long s2 = 1;
//MODMULT: s * b mod m, m = a * b + c, 0 <= c < m
#define MODMULT(a, b, c, m, s) \
q = s/a; s = b*(s-a*q)-c*q; \
if (s < 0) s+=m;
```

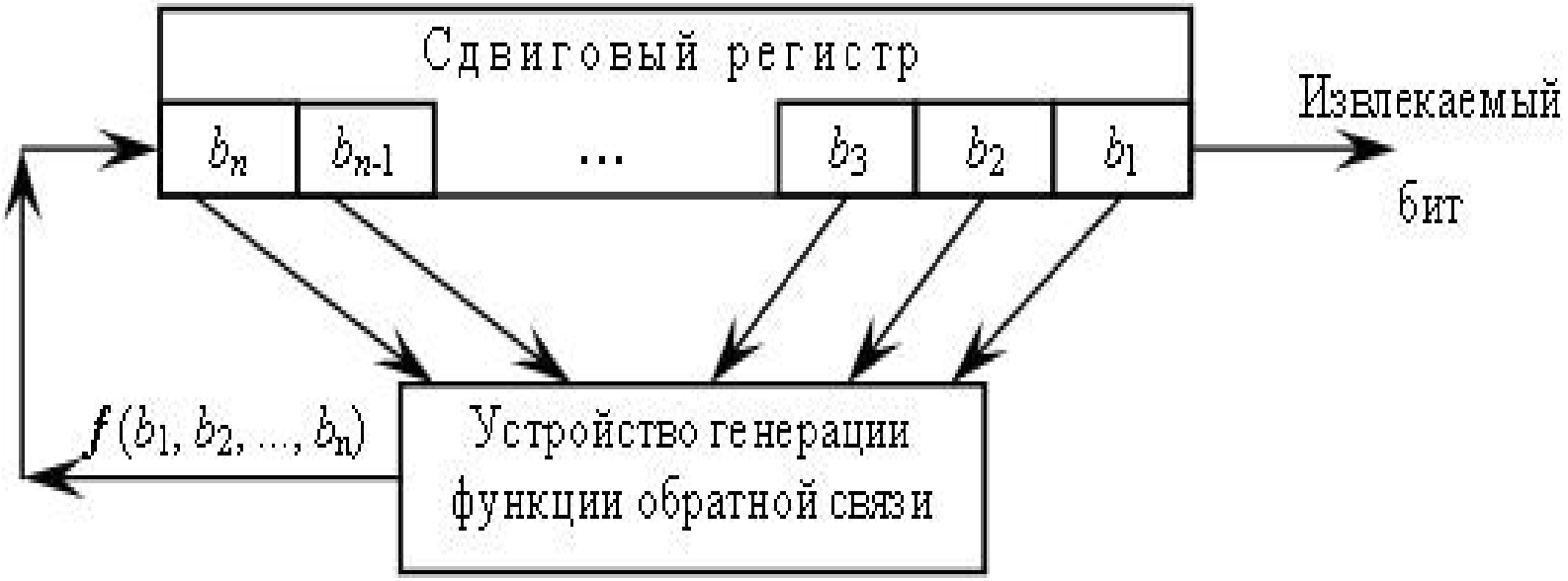
Суперпозиция КГ

```
double combinedLCG (void)
{
      long q;
      long z;
      MODMULT (53668, 40014, 12211, 2147483563L, s1)
      MODMULT (52774, 40692, 3791, 2147483399L, s2)
      z = s1 - s2;
      if (z < 1) z += 2147483562;
      return z * 4.656613e-10;
}
void InitLCG (long InitS1, long InitS2)
{
      s1 = InitS1;
      s2 = InitS2;
}
```

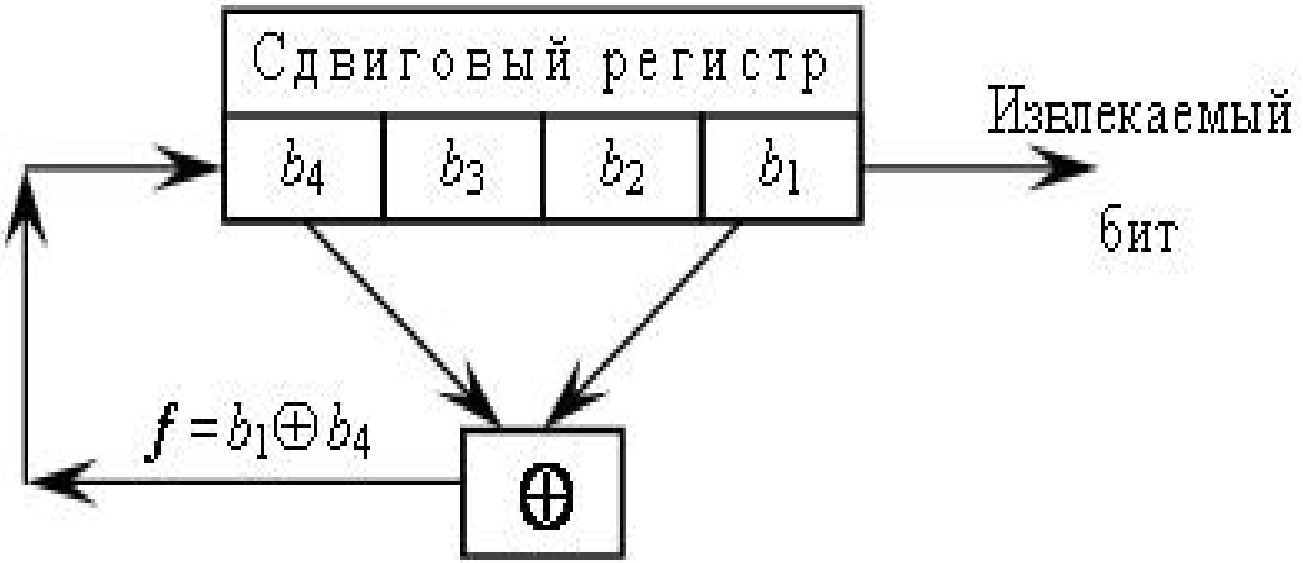
Выводы по КГ

- Последовательность, сгенерированная конгруэнтым уравнением, показывает приемлемую случайность, если следовать вышеуказанным рекомендациям
- Последовательность полезна в некоторых приложениях, где требуется только случайность (например моделирование)
- Последовательность для криптографии является небезопасной, поэтому в чистом виде конгруэнтные генераторы не используются в современной криптографии
- КГ очень просты в реализации, особенно в программной, поэтому существуют в виде готовых реализации во многих языках программирования (random)
- Такие последовательности часто используются в криптографии как вспомогательные для инициализации

Генераторы на сдвиговых регистрах



Пример 4-х разрядного генератора



Максимальный период LFSR

Теорема: Для того чтобы LFSR был LFSR максимальной длины, необходимо и достаточно, чтобы полином, образованный из элементов отводной последовательности плюс единица был **примитивным полиномом** по модулю 2

Пусть **p** – простое число и $\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_t}$ являются различными простыми множителями для $\mathbf{p^m - 1}$, тогда полином является примитивным, если и только если для каждого **i**, $\mathbf{1 \leq i \leq t}$ выполняется условие:

$$x^{\left(p^m-1\right)/r_i} \neq 1 \left(\mathrm{mod}\, f(x)\right)$$

Максимальный период LFSR

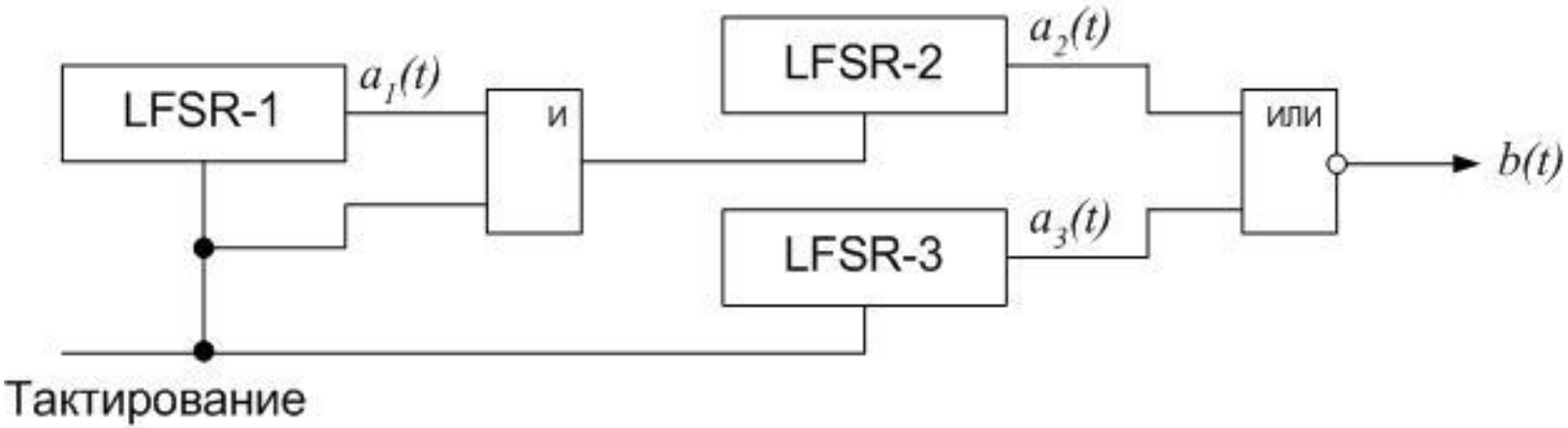
Можем представить полином $x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$
в виде (32, 7, 5, 3, 2, 1, 0)
Числа кроме последнего нуля будут номерами отводных последовательностей. Алгоритм на С:

```
Int LFSR()
{
      Static unsigned long ShiftRegister = 1;
      ShiftRegister = (((( ShiftRegister>>31)
      ^( ShiftRegister>>6)
      ^( ShiftRegister>>4)
      ^( ShiftRegister>>2)
      ^( ShiftRegister>>1)
      ^( ShiftRegister)
      &0x00000001)
      <<31
      | ShiftRegister>>1);
      return ShiftRegister & 0x00000001;
}
```

Генератор Геффа

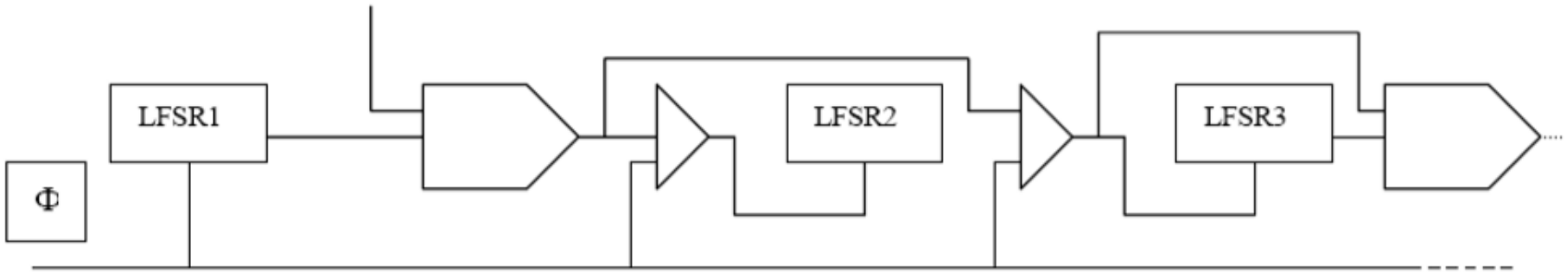
$(n_1+1)n_2n_3$ Период генератора

Генератор «Стоп-пошел»



Каскад Голлмана

Каскад Голлмана состоит из серии LFSR’ов, причем такт каждого следующего LSFR контролируется предыдущим (т.е. если выход LFSR-1 равняется 1 во время $1 - t$, то LFSR-1 меняет свое состояние на следующее). Выход последовательности LFSR есть выход всего генератора.



Выводы по LFSR

- Последовательность, сгенерированная LFSR, показывает приемлемую случайность, если следовать вышеуказанным рекомендациям
- Для моделирования последовательности, пригодной для криптографии необходимо вносить нелинейные преобразования (каскад Голлмана, генератор Геффа, генератор Стоп-Пошел и т.п.)
- LFSR очень просты в аппаратной реализации

Аддитивный генератор

Общий вид генератора

$$X_i = \left(X_{i-a} + X_{i-b} + X_{i-c} + \ldots + X_{i-m}\right) \bmod 2^n$$

- X – это одномерный массив, состоящий из чисел
- Полученное новое значение записывается в конец массива, после чего массив сдвигается влево на 1 позицию
- Коэффициенты a,b,c,…,m также называются номерами отводных последовательностей
- Для программной реализации удобно использовать списки, чтобы избежать операции сдвига

Алгоритм FIPS-186

Этот алгоритм является национальным стандартом США и предназначен для генерации конфиденциальных параметров и ключевой информации для национального стандартного алгоритма электронной цифровой подписи DSA.

В качестве односторонней функции может использоваться алгоритм шифрования DES или алгоритм хэширования SHA

Входные данные: количество генерируемых псевдослучайных чисел *m* и 160-битное простое число *q*

Выходные данные: последовательность m псевдослучайных чисел

Алгоритм FIPS-186

1. Задать произвольное число b: $160 \le b \le 512$.
2. Сгенерировать некоторым образом случайное (и секретное) b-битное начальное значение s.
3. Задать вспомогательное 160-битное слово t (в шестнадцатеричной записи):

```
t = 67452301 efcdab89 98badcfe 10325476 c3d2e1f0.
```

4. Для $i = \overline{1,m}$:
   1) произвольно задать b-битное число y_i либо положить его равным нулю;
   2) вычислить $z_i = (s + y_i) \bmod 2^b$;
   3) вычислить $x_i = G(t, z_i) \bmod q$;
   4) вычислить $s = (1 + s + x_i) \bmod 2^b$.
5. Как результат выполнения предыдущего шага формируется псевдослучайная последовательность $x_1, x_2, \dots, x_m$, которую можно использовать в качестве потока псевдослучайных бит.

Алгоритм вычисления значений функции $G(t, c)$:

1. Разбить слово t на пять 32-битных слов:

$$t = H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4.$$

2. К слову c дописать справа $512 - b$ нулей так, чтобы получить 512-битное слово:

$$M = c \,||\, 0^{512-b}.$$

3. Разбить слово U на шестнадцать 32-битных слов $M = M_0 \,||\, M_1 \,||\, \dots \,||\, M_{15}$.
4. Выполнить 1 раз шаг 4 (основной цикл) алгоритма SHA–1, изменяющий H_i (т.е. для $N = 1$).
5. Выходное слово является конкатенацией:

$$G(t, c) = H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4.$$

Алгоритм ANSI X9.17

Этот алгоритм является национальным стандартом США для генерации двоичной псевдослучайной последовательности [2]. Используется в приложениях, обеспечивающих безопасность финансовых платежей, и PGP

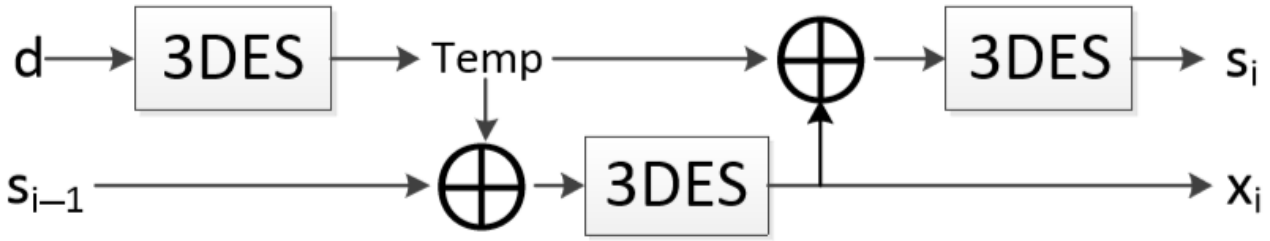
В этом генераторе в качестве односторонней функции используется алгоритм шифрования TripleDES (3DES, «тройной DES»). Этот алгоритм использует два 64-битных ключа K_1 и K_2 следующим образом:

$$F_K(M) = E_{K_1}\left(D_{K_2}\left(E_{K_1}(M)\right)\right),$$

Входные данные: некоторое случайное 64-битное начальное значение S_0, 128-битный составной ключ K и m – количество 64-битных слов

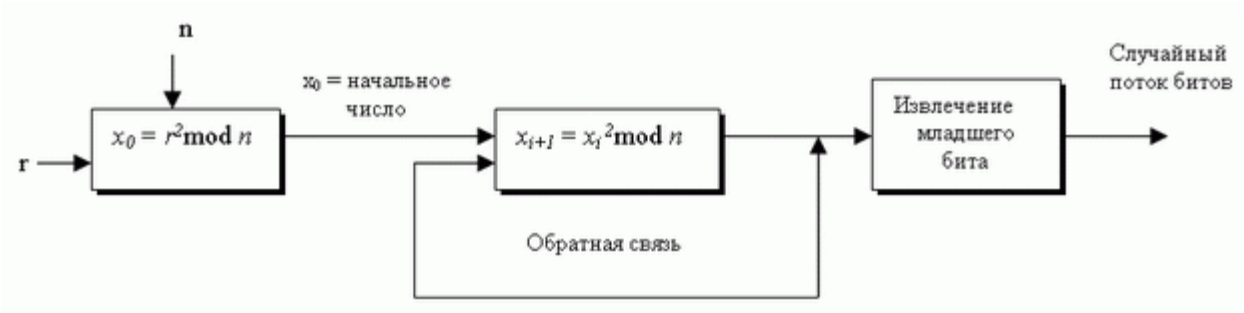
Выходные данные: последовательность m 64-битных слов

Алгоритм ANSI X9.17



1. Зафиксировать 64-битное представление d даты и времени в момент обращения к программе генерации и вычислить вспомогательное 64-битное двоичное слово $Temp = F_K(d)$.
2. Для $i = \overline{1,m}$:
   1) вычислить значение i-го выходного слова $x_i = F_K(Temp \oplus s_{i-1})$;
   2) вычислить новое значение параметра s_i: $s_i = F_K(x_i \oplus Temp)$.
3. В результате предыдущего шага формируется выходная псевдослучайная последовательность из m слов $x_1, x_2, \ldots, x_m$ либо двоичная псевдослучайная последовательность из $64 \cdot m$ бит: $X = x_1 \,||\, x_2 \,||\, \ldots \,||\, x_m$.

Алгоритм BBS (Blum-Blum-Shub)



1. Сгенерировать два достаточно больших секретных случайных (и различных) простых числа $p, q \equiv 3 \pmod 4$ и вычислить $N = p \cdot q$.
2. Выбрать случайное стартовое целое число s $(1 \le s \le N-1)$ так, что НОД$(s, N) = 1$. Вычислить $u_0 = s^2 \bmod N$.
3. Для $i = \overline{1, m}$:
   1) вычислить $u_i = u_{i-1}^2 \bmod N$;
   2) вычислить x_i как самый младший бит двоичного представления числа u_i.
4. В результате предыдущего шага формируется выходная псевдослучайная последовательность $x_1, x_2, \ldots, x_m$.

Алгоритм RSA (Шамира)

1. Сгенерировать два различных больших простых числа p и q. Вычислить $N = p \cdot q$ и $\phi(N) = (p-1) \cdot (q-1)$.
2. Выбрать случайное целое число k $(1 < k < \phi(N))$, взаимно простое с $\phi(N)$, т.е. $\text{НОД}\big(k, \phi(N)\big) = 1$.
3. Выбрать случайное целое стартовое значение u_0: $1 < u_0 < N - 1$.
4. Для $i = \overline{1, m}$:
   1) вычислить $u_i = u_{i-1}^k \bmod N$;
   2) вычислить $x_i \in \{0,\ 1\}$ – самый младший бит числа u_i в двоичном представлении.
5. В результате предыдущего шага формируется выходная псевдослучайная последовательность $x_1, x_2, \dots, x_m$.

Алгоритм Yarrow-160

Данный алгоритм разработан Б. Шнайером, Дж. Келси и Н. Фергюсоном в 1999 году. В алгоритме для шифрования (функция $E_K()$) может использоваться алгоритм DES (или 3DES), а в качестве хэш-функции $h()$ – SHA–1.

Задание начальных значений:

1) Задать размер шифруемого сообщения $n = 64$, т.к. для шифрования используется алгоритм *DES*.
2) Задать $k = 64$ – размер ключа K, используемого при шифровании.
3) Задать значение P_g ($0 < P_g < 2^{n/3}$, обычно $P_g = 10$), определяющее количество бит, после генерации которых нужно обновить значение ключа K.
4) Задать значение P_t ($P_t > P_g > 0$), определяющее количество бит, после генерации которых нужно запустить механизм обновления ключа K и счётчика C_i, используя накопитель энтропии (*entropy accumulator*).
   Накопитель энтропии конкатенирует «случайные» данные (текущая дата и время, количество запущенных в системе процессов и т.п.). Пусть v – результат конкатенации накопленных данных.
5) Задать $t = 0$, где t – количество запусков механизма обновления ключа и счётчика.
6) Задать некоторое начальное значение n–битного счётчика C_0.
7) Присвоить $curP_g = P_g$, $curP_t = P_t$.

Алгоритм Yarrow-160

Для $i = \overline{1, m}$ выполнить:

1) Если $curP_g = 0$, то:
   а) с помощью функции $G(i)$ сгенерировать k бит, которые будут использоваться в качестве нового ключа K;
   б) присвоить $curP_g = P_g$.
2) Если $curP_t = 0$, то:
   а) вычислить $v_0 = h(v \,||\, t)$;
   б) вычислить $v_i = h(v_{i-1} \,||\, v_0 \,||\, i)$ для $i = 1, \dots, t$;
   в) вычислить $K = H(h(v_t \,||\, K), k)$;
   г) вычислить $C_i = E_K(0)$;
   д) присвоить $curP_t = P_t,\ curP_g = P_g,\ t = t + 1$.
3) Вычислить $x_i = G(i)$, которое является следующим блоком выходной последовательности.
4) Выполнить $curP_g = curP_g - 1$ и $curP_t = curP_t - 1$.

В результате предыдущего шага формируется выходная псевдослучайная последовательность из m слов $x_1, x_2, \dots, x_m$ либо двоичная псевдослучайная последовательность из $n \cdot m$ бит: $X = x_1 \,||\, x_2 \,||\, \dots \,||\, x_m$.

Генерация реальных случайных чисел

1. Использование специальных таблиц RAND.
2. Использование случайного шума.
3. Использование таймера компьютера.
4. Измерение скрытого состояния клавиатуры.
5. Аппаратно-временные характеристики компьютера:
   - положение мыши на экране монитора;
   - текущий номер сектора и дорожки дисковода или винчестера;
   - номер текущей строки развертки монитора;
   - времена поступления сетевых пакетов;
   - выход микрофона
   - тепловой шум резисторов

Плюсы и минусы поточных шифров

Преимущества

- Простая схема шифрования и дешифрования (просто XOR)
- Высокая скорость (исп. для потоковых данных: видео-, аудио-)
- Нет накопления ошибки

Недостатки

- Проблема распределения ключей -нельзя использовать один и тот же ключ дважды:

$$C_1 \oplus C_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$$

- Проблема генерации ключевого потока

Где узнать больше

1. https://www.sgu.ru/sites/default/files/textdocsfiles/2018/07/09/slepovichev_i.i._generatory_psevdosluchaynyh_chisel_2017.pdf
2. https://habr.com/ru/post/151187/
3. Шнайер Б. 16 Генераторы псевдослучайных последовательностей и потоковые шифры // Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — 816 с. — 3000 экз. — ISBN 5-89392-055-4.
4. Шнайер Б. 2.8 Генерация случайных и псевдослучайных последовательностей // Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C. — М.: Триумф, 2002. — 816 с. — 3000 экз. — ISBN 5-89392-055-4.