# Лекция 5. Случайные числа

Во многих алгоритмах шифрования требуются последовательности случайных чисел (далее – СЧ). Так при инициализации роторов и рефлектора Энигмы вы уже использовали случаные числа.

Диалог молодого и опытного программиста: «Товарищ программист, мне нужен генератор случайных чисел. 14!»

Предположим, что у нас есть **машина**, которая показывает нам **числа от нуля до девяти**: 2, 4, 3, 0, 2, 9, 4, 5, 9, 5, 9, 7, 8, 2 и т.д. Случайны ли они?

Зависит от нашей **задачи** – ведь даже **если подряд следует** десять семерок это может быть последовательностью (то есть, не случаны), а может быть просто **совпадением** (случайны)!

СЧ нужны для проведения экспериментов (плохие СЧ – недостоверные результаты эксперимента)

Также существуют таблицы случайных чисел для артиллеристов (для поправок и расчета числа попаданий – как в методе Монте-Карло). В «Курсе Артиллерии» Блинов А. (1944) изложен Закон рассеивания:

- небеспредельность

- симметричность

- неравномерность

Гауссово (нормальное) распределение – попадания в цель

Алгоритм артиллериста:

- выбрать центр попадания

- выбрать шкалу рассеивания

- => % попаданий

Криптографическим системам необходимы случайные числа для создания ключей. Безопасность зависит от **произвольности** используемых ею случайных чисел. Произвольность зависит от **надежности** используемого **генератора** случайных чисел.

## Действительно случайные процессы:

* **физические процессы**
  + физические шумы в резисторе,
  + шум звуковой карты
  + счётчик тактов процессора
  + электрические шумы диодов,
  + данные счетчика Гейгера от космического или земной радиации,
  + сигналы приемника радиопомех,
  + цифровые снимки фотоаппарата, направленного на один или несколько стробоскопов,
  + турбулентность воздуха в дисководах
  + момент времени поступления сетевых пакетов в сетевой адаптер.

Редко применяются в чистом виде, так как сложны в реализации, но подвержены грубым атакам.

* **действия пользователя**
  + набор на клавиатуре большой строки произвольных символов, которая может быть использована для
    - создания случайных чисел сама
    - для получения времени между нажатиями клавиш
  + движения мыши
  + записать произнесенные в микрофон звуки.

## Применение генераторов случайных чисел:

- полученная информация **без изменений** (создается отрывной одноразовый блокнот, экземпляр которого выдается каждому участнику обмена – проблема ограниченности)

-- потоковое **сложение с последовательностью** по модулю 2

- служит **начальным числом** для математической функции генерации случайных чисел:

-- порядковый номер в массиве чисел

-- шаг в последовательности псевдослучайных чисел

-- возведение битов в случайном числе

## Псевдослучайные числа

Псевдослучайные числа образуют последовательность, случайную с точки зрения статистических законов.

ПСЧ – последовательность независимых друг от друга чисел, имеющих равномерное распределение

ПСЧ нужны в алгоритмах, которые циклически обращаются к случайному числу.

Требования к генератору ПСЧ

- длина периода д.б. больше интервала использования ПСЧ

- независимость последовательных значений (в частности, случайность всех битов последовательности)

- равномерность в n измерениях (полиномы n-ой степени, отсечения n бит)

- необратимость

Генератор ПСЧ **включен в современные процессоры** (например, функция RdRand есть в процессорах Intel – набор инструкций IA-32)

**Квазислучайные числа**

Последовательность чисел с **большей равномерностью** распределения, чем случайные и псевдослучайные (для более определенного поведения)

Из полученной последовательности создается ключ.

(фактически один пароль заменяется другим, более длинным, если мы знаем алгоритм генерации чисел)

Последовательность чисел может использоваться для:

- определения размера блока

- порядка перестановок

Похожее на истинно случайное число можно получить по формуле:

Rand=sec100+sec100\*msec

**Алгоритмы получения псевдослучайных чисел**

## Алгоритм Фон Неймана (метод серединных квадратов)

(основатель цифровых электронных компьютеров, разработчик водородной бомбы) (1946)

Имеется четырехзначное число R0. Это число возводится в квадрат и заносится в R1. Далее из R1 берется середина (четыре средних цифры), это — новое случайное число, оно записывается в R0. Затем эта процедура повторяется многократно. Недостатком этого способа является его вырождаемость.

берем любое четырехзначное число 1234 и циклически возводим в квадрат две средние цифры: 0**52**9-1**70**4-4**90**0-8**10**0-0**10**0-0**10**0. Получили вырождение

Так если на каком, то шаге R0 превратится в 0, то и все последующие числа станут равными нулю

## Алгоритм на числах Фибоначи

Последовательность Фибоначи: 0,1,1,2,3,5,8,13,21,34,…

Берутся последние цифры: 0,1,1,2,3,5,8,3,1,4,…

## Метод серединных произведений

Число R0 умножается на R1, из полученного результата R2 извлекается середина R2\* (это очередное случайное число) и умножается на R1. По этой схеме вычисляются все последующие случайные числа.

R(n+1)=середина числа R(N)\*R

## Метод перемешивания

Идея метода состоит в следующем. Пусть в ячейке хранится начальное число R0. Циклически сдвигая содержимое ячейки влево на 1/4 длины ячейки, получаем новое число R0\*. Точно так же, циклически сдвигая содержимое ячейки R0 вправо на 1/4 длины ячейки, получаем второе число R0\*\*. Сумма чисел R0\* и R0\*\* дает новое случайное число R1. Далее R1 заносится в R0, и вся последовательность операций повторяется.

R(n+1)=R(N)>>2+ R(N)<<2

## Линейный конгруэнтный генератор

Линейный конгруэнтный генератор ПСЧ вычисляет члены линейной рекуррентной последовательности по модулю некоторого натурального числа m.

Каждый член данной последовательности задается следующей формулой:

R(n+1)=(a\*R(n)+c)mod(m), где a и c - некоторые целочисленные коэффициенты.

При разных значениях стартового числа R(0) получаются различные последовательности случайных чисел.

Многие свойства последовательности определяются выбором коэффициентов в формуле и не зависят от выбора стартового числа (что ограничивает использование линейного конгруэнтного генератора в криптографии).

m<2^(длина слова в битах)

Важно взять A>2^(длина слова в битах/2) для получения большого периода повторения

Изменение порождающего числа приводит к изменению

- последовательности чисел

- длины периода генератора

Предпочтительные значения констант

|  |  |  |  |
| --- | --- | --- | --- |
| **a** | **b** | **m** | **Переполнение при** |
| 106 | 1283 | 6075 | 2^20 |
| 430 | 2531 | 11979 | 2^23 |
| 84589 | 45989 | 217728 | 2^35 |

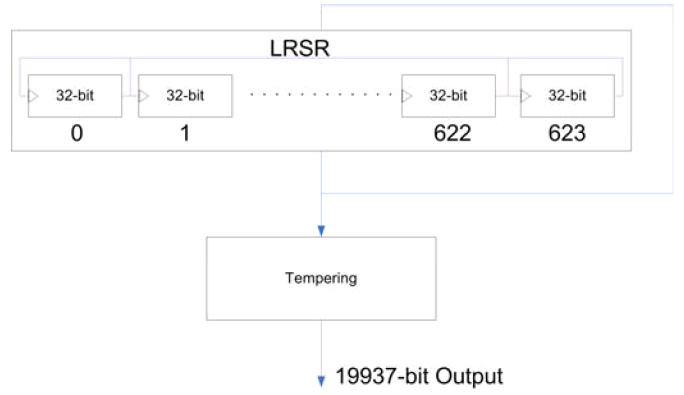
## Квадратичный конгруэнтный генератор

x(n+1)=(a\*x(n)\*x(n)+b\*x(n)+c)mod(m), где a,b и c - некоторые целочисленные коэффициенты.

**Вихрь Мерсенна**

Вихрь Мерсенна состоит из двух основных частей: **рекурсия** и **закалка**. Рекурсивная часть реализуется регистром сдвига с линейной обратной связью, в котором все биты состояния определяются рекурсивно; биты выхода определяются также рекурсивно на основе битов состояния.

Блок-схема

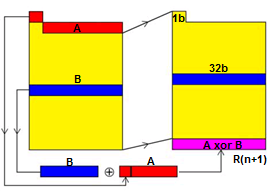


Регистр сдвига = 624 элементов по 32 бита, первый элемент имеет только 1 бит (всего 19937 бит).

Процесс генерации:

1. применение битовой маски, отбрасывающей 31 бита (кроме старшего)
2. инициализация *(****x****0,* ***x****1,…,* ***x****623)* любыми беззнаковыми 32-разрядными целыми числами
3. объединение и переходные состояния
4. R(n+1)=

Пространство состояний имеет 19937 бит (624 х 32 — 31). Следующее состояние генерируется сдвигом одного слова вертикально вверх и вставкой нового слова в конец. Новое слово вычисляется гаммированием средней части с исключённой[[13]](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%85%D1%80%D1%8C_%D0%9C%D0%B5%D1%80%D1%81%D0%B5%D0%BD%D0%BD%D0%B0#cite_note-Cryptographic_Mersenne_Twister_and_Fubuki_Stream.2FBlock_Cipher.E2.80.94.E2.80.94.E2.80.94-13). Выходная последовательность начинается с ***x****624,* ***x****625*,…  
Смена состояния МТ



Закалка

Алгоритм в шесть шагов:

**Шаг 0.** Предварительно инициализируется значение констант *u*, *h*, *a*

*u* ← 10…0 битовая маска старших *w-r* бит,

*h* ← 01…1 битовая маска младших *r* бит,

*a* ← *aw-1aw-2…a0* последняя строка матрицы *A*.  
 **Шаг 1.** ***x***[0], ***x***[1],…,***x***[n-1] ← начальное заполнение  
 **Шаг 2.** Вычисление *(****x****iu |* ***x****i+1l)*

***y*** ← (***x***[i] AND *u*) OR (***x***[(i + 1) mod n] AND *h*)  
 **Шаг 3.** Вычисляется значение следующего элемента последовательности по

рекуррентному выражению (1)

***x***[i] ← ***x***[(i + m) mod n] XOR (***y***>>1) XOR *a* если младший бит ***y*** = 1  
 Или

***x***[i] ← ***x***[(i + m) mod n] XOR (***y***>>1) XOR 0 если младший бит ***y*** = 0

**Шаг 4.** Вычисление ***x***[i]*T*

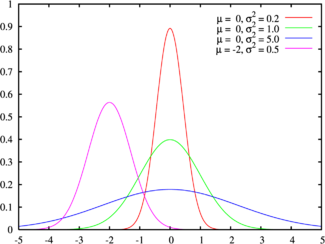
***y*** ← ***x***[i]

***y*** ← ***y*** XOR (***y***>>*u*)

***y*** ← ***y*** XOR ((***y***<<*s*) AND **b**)

***y*** ← ***y*** XOR ((***y***<<*t*) AND **c**)

***z*** ← ***y*** XOR (***y***>>*l*)

 вывод ***y***

**Шаг 5.** i ← (i + 1) mod n   
 **Шаг 6.** Перейти к шагу 2.

## Нормальное (Гауссово) распределение

## Преобразование Бокса-Мюллера

NR=m+(s\*(сумма по k(rand(i))))\*sqrt (3/k),

где m-математическое ожидание,

s-дисперсия (средне-квадратичное отклонение),

rand-равномерное распределение на интервале (-1,1)

s=m(x^2)-(m(x))^2

Очень важен большой период.

## Тест на К-распределение случайности

Спектральный тест и тест на к-распределении считается сильнейшим.

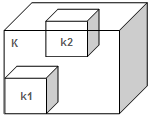
Псевдослучайная последовательность ***x****i* периода *P*, состоящая из *w*-битных целых, имеет *k*-распределение с *v*-битной точностью, если она удовлетворяет следующему условию:

Пусть *truncv(x)* — число, образованное первыми *v* битами последовательности *xi*, рассмотрим *P* векторов вида  длиной *kv* бит.

Тогда каждая из *2kv* полученных отсечений одной длины – возможных комбинаций битов – **встречается равное число раз**, за исключением комбинации, состоящей полностью из нулей, которая встречается на один раз меньше.

## Геометрическая интерпретация

Для каждого *v* = 1, 2,., *w*, пусть *k(v)* — максимальное число, такое, что последовательность является *к*-распределенной с *v*-битной точностью. Разделим каждое целое ***x****i* на 2w для нормализации в псевдослучайное вещественное число ***x****i* из интервала [0, 1]. Поместим *P* точек в *k*- мерный куб с координатами *(****x****i,* ***x****i+1…,* ***x****i+k-1)(i = 0, 1,…, P-1)* для всего периода. Каждая из осей данного *k*-мерного куба разделена на 2v интервалов. Таким образом, мы разделили куб на *2kv* малых куба. Последовательность является *к*-распределенной с *v*-битной точностью, если каждый малый куб содержит равное число точек, кроме куба в начале координат, который содержит на одну меньше точек. Следовательно, чем выше *k(v)* для каждого *v*, тем более многомерным будет распределение с *v*-битной точностью. Под тестом на *k*-распределение, мы будем понимать получение значений *k(v)*.



То есть, любой вложенный в К куб k1, k2… одинакового размера содержит одинаковое число точек.