**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФГБОУ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**РФ КГУ 090303.65.КР.101301107 1**

**Курсовой проект**

По дисциплине «Криптографические методы защиты информации»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

Листов 23

2015

**АННОТАЦИЯ**

Документ содержит общие сведения о предметной области, примеры алгоритмов и пример создания алгоритма.

СОДЕРЖАНИЕ

# Оглавление

[Оглавление 3](#_Toc438054420)

[Что же такое ГСПЧ? 4](#_Toc438054421)

[Генератор псевдослучайных чисел 4](#_Toc438054422)

[Источники случайных чисел 4](#_Toc438054423)

[Детерминированные ГПСЧ 5](#_Toc438054424)

[ГПСЧ с источником энтропии или ГСЧ 6](#_Toc438054425)

[Пример простейшего ГСЧ с источником энтропии 7](#_Toc438054426)

[Примеры ГСЧ и источников энтропии 7](#_Toc438054427)

[ГПСЧ в криптографии 10](#_Toc438054428)

[Примеры криптостойких ГПСЧ 10](#_Toc438054429)

[Циклическое шифрование 10](#_Toc438054430)

[Аппаратные ГПСЧ 11](#_Toc438054431)

[Применение ГСЧ в лотереях 12](#_Toc438054432)

[Уязвимости ГПСЧ 14](#_Toc438054433)

[Линейный конгруэнтный ГПСЧ(LCPRNG) 14](#_Toc438054434)

[Предсказание результатов линейно-конгруэнтного метода 14](#_Toc438054435)

[Взлом встроенного генератора случайных чисел в Java 15](#_Toc438054436)

[Создание ГПСЧ. 19](#_Toc438054437)

[Программная реализация ГПСЧ 21](#_Toc438054438)

[Вывод 23](#_Toc438054439)

# Что же такое ГСПЧ?

Генератор псевдослучайных чисел (**ГПСЧ**, англ. *pseudorandom number generator*, *PRNG*) — алгоритм, порождающий последовательность чисел, элементы которой почти независимы друг от друга и подчиняются заданному распределению (обычно равномерному).

Современная информатика широко использует псевдослучайные числа в самых разных приложениях — от метода Монте-Карло и имитационного моделирования до криптографии. При этом от качества используемых ГПСЧ напрямую зависит качество получаемых результатов.

## Источники случайных чисел

Источники настоящих случайных чисел найти крайне трудно. Физические шумы, такие, как детекторы событий ионизирующей радиации, дробовой шум в резисторе или космическое излучение, могут быть такими источниками. Однако применяются такие устройства в приложениях сетевой безопасности редко. Сложности также вызывают грубые атаки на подобные устройства.

Криптографические приложения используют для генерации случайных чисел особенные алгоритмы. Эти алгоритмы заранее определены и, следовательно, генерируют последовательность чисел, которая теоретически не может быть статистически случайной. В то же время, если выбрать хороший алгоритм, полученная численная последовательность — **псевдослучайных чисел** — будет проходить большинство тестов на случайность. Одной из характеристик такой последовательности является период повторения, который должен быть больше рабочего интервала, из которого берутся числа.

Генератор псевдослучайных чисел включён в состав многих современных процессоров, например, [RdRand](https://ru.wikipedia.org/wiki/RdRand" \o "RdRand) входит в набор инструкций IA-32.

Альтернативным решением является создание набора из большого количества случайных чисел и опубликование его в некотором [словаре](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B2%D0%B0%D1%80%D1%8C), называемом «[одноразовым блокнотом](https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%BD%D0%BE%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B1%D0%BB%D0%BE%D0%BA%D0%BD%D0%BE%D1%82)». Тем не менее, и такие наборы обеспечивают очень ограниченный источник чисел по сравнению с тем количеством, которое требуется приложениям сетевой безопасности. Хотя данные наборы действительно обеспечивают статистическую случайность, они недостаточно безопасны, так как злоумышленник может получить копию словаря.

## Детерминированные ГПСЧ

Никакой [детерминированный алгоритм](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) не может генерировать полностью случайные числа, он может только [аппроксимировать](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D1%80%D0%BE%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) некоторые их свойства. Как сказал Джон фон Нейман, «*всякий, кто питает слабость к арифметическим методам получения случайных чисел, грешен вне всяких сомнений*».

Любой ГПСЧ с ограниченными ресурсами рано или поздно зацикливается — начинает повторять одну и ту же последовательность чисел. Длина циклов ГПСЧ зависит от самого генератора и составляет около 2*n*/2, где *n* — размер внутреннего состояния в битах, хотя линейные конгруэнтные и [LFSR](https://ru.wikipedia.org/wiki/LFSR)-генераторы обладают максимальными циклами порядка 2*n*. Если порождаемая последовательность ГПСЧ сходится к слишком коротким циклам, то такой ГПСЧ становится предсказуемым и непригодным для практических приложений.

Большинство простых арифметических генераторов хотя и обладают большой скоростью, но страдают от многих серьёзных недостатков:

* Слишком короткий период/периоды.
* Последовательные значения не являются независимыми.
* Некоторые биты «менее случайны», чем другие.
* Неравномерное одномерное распределение.
* Обратимость.

В частности, алгоритм [RANDU](https://ru.wikipedia.org/wiki/RANDU), десятилетиями использовавшийся на [мейнфреймах](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B9%D0%BD%D1%84%D1%80%D0%B5%D0%B9%D0%BC), оказался очень плохим, что вызвало сомнения в достоверности результатов многих исследований, использовавших этот алгоритм.

Наиболее распространены [линейный конгруэнтный метод](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BD%D0%B3%D1%80%D1%83%D1%8D%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4), [метод Фибоначчи с запаздываниями](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%A4%D0%B8%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D0%B8_%D1%81_%D0%B7%D0%B0%D0%BF%D0%B0%D0%B7%D0%B4%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%D0%BC%D0%B8), [регистр сдвига с линейной обратной связью](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80_%D1%81%D0%B4%D0%B2%D0%B8%D0%B3%D0%B0_%D1%81_%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%BE%D0%B9_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C%D1%8E), [регистр сдвига с обобщённой обратной связью](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80_%D1%81%D0%B4%D0%B2%D0%B8%D0%B3%D0%B0_%D1%81_%D0%BE%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B9_%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C%D1%8E).

**Линейный конгруэнтный метод** — один из методов генерации псевдослучайных чисел. Применяется в простых случаях и не обладает криптографической стойкостью. Входит в стандартные библиотеки различных компиляторов.

Линейный конгруэнтный метод был предложен Д.Г. Лемером в 1949 году. Суть метода заключается в вычислении последовательности случайных чисел , полагая

где  — модуль ,  — множитель ,  — приращение ,  — начальное значение .

Эта последовательность называется *линейной конгруэнтной последовательностью*. Например, для,  получим последовательность

Из современных ГПСЧ широкое распространение также получил «[вихрь Мерсенна](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)», предложенный в 1997 году Мацумото и Нисимурой. Его достоинствами являются колоссальный период (219937−1), равномерное распределение в 623 измерениях (линейный конгруэнтный метод даёт более или менее равномерное распределение максимум в 5 измерениях), быстрая генерация случайных чисел (в 2-3 раза быстрее, чем стандартные ГПСЧ, использующие линейный конгруэнтный метод). Однако существуют алгоритмы, распознающие последовательность, порождаемую вихрем Мерсенна, как неслучайную.

## ГПСЧ с источником энтропии или ГСЧ

Наравне с существующей необходимостью генерировать легко воспроизводимые последовательности случайных чисел, также существует необходимость генерировать совершенно непредсказуемые или попросту абсолютно случайные числа. Такие генераторы называются *генераторами случайных чисел* (ГСЧ — [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *random number generator, RNG*). Так как такие генераторы чаще всего применяются для генерации уникальных симметричных и асимметричных ключей для шифрования, они чаще всего строятся из комбинации криптостойкого ГПСЧ и внешнего источника [энтропии](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%8D%D0%BD%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D1%8F) (и именно такую комбинацию теперь и принято понимать под ГСЧ).

Почти все крупные производители микрочипов поставляют аппаратные ГСЧ с различными источниками энтропии, используя различные методы для их очистки от неизбежной предсказуемости. Однако на данный момент скорость сбора случайных чисел всеми существующими микрочипами (несколько тысяч бит в секунду) не соответствует быстродействию современных процессоров.

В современных исследованиях осуществляются попытки использования измерения физических свойств объектов (например, [температуры](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0)) или даже [квантовых](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D1%84%D0%B8%D0%B7%D0%B8%D0%BA%D0%B0) [флуктуаций](https://ru.wikipedia.org/wiki/%D0%A4%D0%BB%D1%83%D0%BA%D1%82%D1%83%D0%B0%D1%86%D0%B8%D1%8F) [вакуума](https://ru.wikipedia.org/wiki/%D0%92%D0%B0%D0%BA%D1%83%D1%83%D0%BC) в качестве источника энтропии для ГСЧ.

В персональных компьютерах авторы программных ГСЧ используют гораздо более быстрые источники энтропии, такие, как шум звуковой карты или [счётчик тактов процессора](https://ru.wikipedia.org/wiki/%D0%A1%D1%87%D1%91%D1%82%D1%87%D0%B8%D0%BA_%D1%82%D0%B0%D0%BA%D1%82%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%B0). Сбор энтропии являлся наиболее уязвимым местом ГСЧ. Эта проблема до сих пор полностью не разрешена во многих устройствах (например, [смарт-картах](https://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82-%D0%BA%D0%B0%D1%80%D1%82%D0%B0)), которые таким образом остаются уязвимыми. Многие ГСЧ используют традиционные испытанные, хотя и медленные, методы сбора энтропии вроде измерения реакции пользователя (движение [мыши](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BC%D1%8B%D1%88%D1%8C) и т. п.), как, например, в [PGP](https://ru.wikipedia.org/wiki/PGP) и Yarrow, или взаимодействия между [потоками](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C), как, например, в [Java](https://ru.wikipedia.org/wiki/Java) SecureRandom.

## Пример простейшего ГСЧ с источником энтропии

Если в качестве источника энтропии использовать текущее время, то для получения [целого числа](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) от 0 до *N* достаточно вычислить [остаток от деления](https://ru.wikipedia.org/wiki/%D0%9E%D1%81%D1%82%D0%B0%D1%82%D0%BE%D0%BA_%D0%BE%D1%82_%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) текущего времени в [миллисекундах](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BA%D1%83%D0%BD%D0%B4%D0%B0) на число *N*+1. Недостатком этого ГСЧ является то, что в течение одной миллисекунды он выдает одно и то же число.

## Примеры ГСЧ и источников энтропии

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | [**Источник энтропии**](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA_%D1%8D%D0%BD%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D0%B8) | **ГПСЧ** | **Достоинства** | **Недостатки** |
| [**/dev/random**](https://ru.wikipedia.org/wiki/dev/random)**в**[**UNIX**](https://ru.wikipedia.org/wiki/UNIX)**/[Linux](https://ru.wikipedia.org/wiki/Linux" \o "Linux)** | Счётчик тактов процессора, однако собирается только во время аппаратных прерываний | [LFSR](https://ru.wikipedia.org/wiki/LFSR), с хешированием выхода через [SHA-1](https://ru.wikipedia.org/wiki/SHA-1) | Есть во всех Unix, надёжный источник энтропии | Очень долго «нагревается», может надолго «застревать», либо работает как ГПСЧ (*/dev/urandom*) |
| ***Yarrow* отБрюса Шнайера** | Традиционные методы | [AES](https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard)-256 и [SHA-1](https://ru.wikipedia.org/wiki/SHA-1) маленького внутреннего состояния | Гибкий криптостойкий дизайн | Медленный |
| **Microsoft**[**CryptoAPI**](https://ru.wikipedia.org/wiki/CryptoAPI) | Текущее время, размер жёсткого диска, размер свободной памяти, номер процесса и NETBIOS-имя компьютера | [MD5](https://ru.wikipedia.org/wiki/MD5)-хеш внутреннего состояния размером в 128 бит | Встроен в Windows, не «застревает» | Сильно зависит от используемого криптопровайдера (CSP). |
| [**Java**](https://ru.wikipedia.org/wiki/Java)**SecureRandom** | Взаимодействие между потоками | [SHA-1](https://ru.wikipedia.org/wiki/SHA-1)-хеш внутреннего состояния (1024 бит) | Большое внутреннее состояние | Медленный сбор энтропии |
| **Chaos от Ruptor** | Счётчик тактов процессора, собирается непрерывно | Хеширование 4096-битового внутреннего состояния на основе нелинейного варианта [Marsaglia](https://en.wikipedia.org/wiki/George_Marsaglia" \o "en:George Marsaglia)-генератора | Пока самый быстрый из всех, большое внутреннее состояние, не «застревает» | Оригинальная разработка, свойства приведены только по утверждению автора |
| **RRAND от Ruptor** | Счётчик тактов процессора | Зашифровывание внутреннего состояния поточным шифром [EnRUPT](https://ru.wikipedia.org/wiki/EnRUPT" \o "EnRUPT) в authenticated encryption режиме (aeRUPT) | Очень быстр, внутреннее состояние произвольного размера по выбору, не «застревает» | Оригинальная разработка, свойства приведены только по утверждению автора. Шифр [EnRUPT](https://ru.wikipedia.org/wiki/EnRUPT" \o "EnRUPT) не является криптостойким. |
| [**RdRand**](https://ru.wikipedia.org/wiki/RdRand)**от intel** | Шумы токов | Построение ПСЧ на основе "случайного" битового считывания значений от токов | Очень быстр, не «застревает» | Оригинальная разработка, свойства приведены только по утверждению статьи из habrahabr[[1]](http://habrahabr.ru/) - уточнить. |
| **ГПСЧ Stratosphera от ORION** | Счетчик тактов процессора, собирается непрерывно (также используется соль в виде случайно выбранного целого числа) | Построение ПСЧ на основе алгоритма от Intel с многоразовой инициализацией и сдвигом | Достаточно быстр, не «застревает», проходит все тесты DIEHARD | Оригинальная разработка, свойства приведены только исходя из информации на сайте oriondevteam.com[[2]](http://www.oriondevteam.com/) - (уточнение от 23-10-2013). |

## 

# ГПСЧ в криптографии

Разновидностью ГПСЧ являются ГПСБ (PRBG) — генераторы псевдо-случайных бит, а также различных [поточных шифров](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80). ГПСЧ, как и поточные шифры, состоят из внутреннего состояния (обычно размером от 16 бит до нескольких мегабайт), функции инициализации внутреннего состояния [ключом](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) или *зерном* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *seed*), функции обновления внутреннего состояния и функции вывода. ГПСЧ подразделяются на простые арифметические, сломанные криптографические и [криптостойкие](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D1%82%D0%BE%D0%B9%D0%BA%D0%BE%D1%81%D1%82%D1%8C" \o "Криптостойкость). Их общее предназначение — генерация последовательностей чисел, которые невозможно отличить от случайных вычислительными методами.

Хотя многие криптостойкие ГПСЧ или поточные шифры предлагают гораздо более «случайные» числа, такие генераторы гораздо медленнее обычных арифметических и могут быть непригодны во всякого рода исследованиях, требующих, чтобы процессор был свободен для более полезных вычислений.

В военных целях и в полевых условиях применяются только засекреченные синхронные криптостойкие ГПСЧ (поточные шифры), [блочные шифры](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80) не используются. Примерами известных криптостойких ГПСЧ являются [RC4](https://ru.wikipedia.org/wiki/RC4), [ISAAC](https://ru.wikipedia.org/wiki/ISAAC), [SEAL](https://ru.wikipedia.org/wiki/SEAL_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)), [Snow](https://ru.wikipedia.org/w/index.php?title=Snow&action=edit&redlink=1), совсем медленный теоретический [алгоритм Блюм — Блюма — Шуба](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%BB%D1%8E%D0%BC_%E2%80%94_%D0%91%D0%BB%D1%8E%D0%BC%D0%B0_%E2%80%94_%D0%A8%D1%83%D0%B1%D0%B0), а также счётчики с криптографическими хеш-функциями или криптостойкими блочными шифрами вместо функции вывода.

## Примеры криптостойких ГПСЧ

## Циклическое шифрование

В данном случае используется способ генерации ключа сессии из мастер-ключа. Счетчик с периодом N используется в качестве входа в шифрующее устройство. Например, в случае использования 56-битного ключа [DES](https://ru.wikipedia.org/wiki/DES) может использоваться счетчик с периодом 256. После каждого созданного ключа значение счетчика повышается на 1. Таким образом, псевдослучайная последовательность, полученная по данной схеме, имеет полный период: каждое выходное значение Х0, Х1,…XN-1 основано на разных значениях счетчика, поэтому Х0 ≠ X1 ≠ XN-1. Так как мастер-ключ является секретным, легко показать, что любой секретный ключ не зависит от знания одного или более предыдущих секретных ключей.

# Аппаратные ГПСЧ

Кроме устаревших, хорошо известных LFSR-генераторов, широко применявшихся в качестве аппаратных ГПСЧ в XX веке, к сожалению, очень мало известно о современных аппаратных ГПСЧ (поточных шифрах), так как большинство из них разработано для военных целей и держатся в секрете. Почти все существующие коммерческие аппаратные ГПСЧ запатентованы или [держатся в секрете](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BC%D0%B5%D1%80%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B0%D0%B9%D0%BD%D0%B0). Аппаратные ГПСЧ ограничены строгими требованиями к расходуемой памяти (чаще всего использование памяти запрещено), быстродействию (1-2 такта) и площади (несколько сотен [FPGA](https://ru.wikipedia.org/wiki/FPGA)- или [ASIC](https://ru.wikipedia.org/wiki/ASIC)-ячеек). Из-за таких строгих требований к аппаратным ГПСЧ очень трудно создать криптостойкий генератор, поэтому до сих пор все известные аппаратные ГПСЧ были взломаны. Примерами таких генераторов являются [Toyocrypt](https://ru.wikipedia.org/w/index.php?title=Toyocrypt&action=edit&redlink=1" \o "Toyocrypt (страница отсутствует)) и [LILI-128](https://ru.wikipedia.org/w/index.php?title=LILI-128&action=edit&redlink=1), которые оба являются LFSR-генераторами, и оба были взломаны с помощью алгебраических атак.

Из-за недостатка хороших аппаратных ГПСЧ производители вынуждены применять имеющиеся под рукой гораздо более медленные, но широко известные блочные шифры ([DES](https://ru.wikipedia.org/wiki/DES), AES) и хеш-функции (SHA-1) в поточных режимах.

# Применение ГСЧ в лотереях

Генератор случайных чисел для [лотерей](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D1%82%D0%B5%D1%80%D0%B5%D1%8F) — аппаратно-программный комплекс, применяющийся в розыгрышах, в которых необходимо угадывать комбинацию из определенного количества чисел. Любое из возможных чисел имеет одинаковую вероятность появления.

Попытки создать генератор случайных чисел относятся к 3500 году [до н. э.](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%88%D0%B0_%D1%8D%D1%80%D0%B0) и связаны с древнеегипетской настольной игрой [Сенет](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BD%D0%B5%D1%82" \o "Сенет). В Сенете два игрока играют за две стороны. Ходы определяются с помощью 4 плоских палочек, что и может считаться генератором случайных чисел того времени. Бросают все четыре палочки сразу. Подсчет очков происходит следующим образом: 1 палочка упала белой стороной вверх — 1 очко и дополнительный бросок; 2 — 2 очка; 3 — 3 очка, 4 — 4 и дополнительный бросок. Одна из сторон палочки черная и если все четыре палочки падали черной стороной вверх — это максимальный результат — 5 очков и дополнительный бросок.

Известный генератор случайных чисел [ERNIE](https://ru.wikipedia.org/w/index.php?title=ERNIE&action=edit&redlink=1) применялся на протяжении многих лет для определения выигрышных номеров британской лотереи.

Главные требования к ГСЧ, используемому для проведения розыгрышей:

* Каждое число получено случайно, не имея ничего общего с другими числами в последовательности;
* Каждое число из целого ряда имеет равные шансы на выпадение;
* Каждое число имеет заданную вероятность появления в любой заданной области значений.

Самый распространенный метод генерации случайных чисел называется [*линейный конгруэнтный метод*](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BD%D0%B3%D1%80%D1%83%D1%8D%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4), но есть ещё и другой — [*аддитивный конгруэнтный метод*](https://ru.wikipedia.org/w/index.php?title=%D0%90%D0%B4%D0%B4%D0%B8%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BD%D0%B3%D1%80%D1%83%D1%8D%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4&action=edit&redlink=1). Эти методы генерируют последовательность чисел, удовлетворяющую условию случайности. Основой для использования этих и других методов генерации случайных чисел служит программное обеспечение, бесконечно генерирующее числа, независимо от того, находится в данное время участник в процессе игры или нет. Благодаря этому исключается возможность того, что игрок сможет самостоятельно определить метод генерации, использующийся в данный момент, и «угадать» выпадающие числа.

Например, по законам США требуется, чтобы генератор случайных чисел в игровых автоматах функционировал все время. Кроме того, этим вопросом занимаются непосредственно сами поставщики программного обеспечения.

В российских государственных лотереях («Гослото «5 из 36», «Гослото «6 из 45», «Гослото «7 из 49», «Спортлото 6 из 49», «Рапидо», «Кено-Спортлото», «Топ-3», «12/24», «Всё по сто») для определения победителей используется генератор случайных чисел — аппаратно-программный комплекс, сертифицированный АНО МИЦ и отвечающий рекомендациям [ФГУП ВНИИМС](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D1%80%D0%BE%D1%81%D1%81%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D0%BD%D0%B0%D1%83%D1%87%D0%BD%D0%BE-%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82_%D0%BC%D0%B5%D1%82%D1%80%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B9_%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D1%8B).

При использовании ГСЧ для розыгрышей лотерей необходимо соблюдение следующих требований:

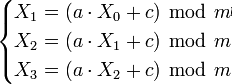
* Тестирование потока чисел на случайность.
* Исключение возможностей вмешательства с целью подтасовки результатов розыгрыша.
* Передача результатов розыгрыша в центр обработки ставок в момент определения выигрышной комбинации с точностью до миллисекунды.
* Возможность резервирования с автоматическим переключением на запасное оборудование в случае неисправности.

# Уязвимости ГПСЧ

* Предсказуемая зависимость между числами.
* Предсказуемое начальное значение генератора.
* Малая длина периода генерируемой последовательности случайных чисел, после которой генератор зацикливается.

## Линейный конгруэнтный ГПСЧ(LCPRNG)

Распространённый метод для генерации псевдослучайных чисел, не обладающий криптографической стойкостью. Линейный конгруэнтный метод заключается в вычислении членов линейной рекуррентной последовательности по модулю некоторого натурального числа m, задаваемой следующей формулой:

где a(multiplier), c(addend), m(mask) — некоторые целочисленные коэффициенты. Получаемая последовательность зависит от выбора стартового числа (seed) X0 и при разных его значениях получаются различные последовательности случайных чисел.  
  
Для выбора коэффициентов имеются свойства позволяющие максимизировать длину периода(максимальная длина равна m), то есть момент, с которого генератор зациклится.  
  
Пусть генератор выдал несколько случайных чисел X0, X1, X2, X3. Получается система уравнений  
  
  
  
Решив эту систему, можно определить коэффициенты a, c, m.

## Предсказание результатов линейно-конгруэнтного метода

Основным алгоритмом предсказания чисел для линейно-конгруэнтного метода является Plumstead’s. Простая реализация конгруэнтного метода на Java.

1. **public** **static** **int** a = 45;
2. **public** **static** **int** c = 21;
3. **public** **static** **int** m = 67;
4. **public** **static** **int** seed = 2;
6. **public** **static** **int** getRand() {
7. seed = (a \* seed + c) % m;
8. **return** seed;
9. }
11. **public** **static** **void** main(String[] args) {
12. **for**(**int** i=0; i<30; i++)
13. System.out.println(getRand());
14. }

Отправив 20 чисел можно с большой вероятностью получить следующие. Чем больше чисел, тем больше вероятность.

## Взлом встроенного генератора случайных чисел в Java

Многие языки программирования, например C(rand), C++(rand) и Java используют LСPRNG. Рассмотрим, как можно провести взлом на примере java.utils.Random. Зайдя в исходный код(jdk1.7) данного класса можно увидеть используемые константы

1. **private** **static** **final** **long** multiplier = 0x5DEECE66DL; *// 25214903917*
2. **private** **static** **final** **long** addend = 0xBL; *// 11*
3. **private** **static** **final** **long** mask = (1L << 48) - 1; *// 281474976710655 = 2^48 – 1*

Метод java.utils.Randon.nextInt() выглядит следующим образом (здесь bits == 32)

1. **protected** **int** next(**int** bits) {
2. **long** oldseed, nextseed;
3. AtomicLong seed = **this**.seed;
4. **do** {
5. oldseed = seed.get();
6. nextseed = (oldseed \* multiplier + addend) & mask;
7. } **while** (!seed.compareAndSet(oldseed, nextseed));
8. **return** (**int**)(nextseed >>> (48 - bits));
9. }

Результатом является nextseed сдвинутый вправо на 48-32=16 бит. Данный метод называется truncated-bits, особенно неприятен при black-box, приходится добавлять ещё один цикл в brute-force. Взлом будет происходить методом грубой силы(brute-force).   
  
Пусть мы знаем два подряд сгенерированных числа x1 и x2. Тогда необходимо перебрать 2^16 = 65536 вариантов oldseed и применять к x1 формулу:

до тех пор, пока она не станет равной x2. Код для brute-force может выглядеть так

1. **public** **class** PasswordCracking {
2. **public** **static** **final** **long** multiplier = 0x5DEECE66DL;
3. **public** **static** **final** **long** addend = 0xBL;
4. **public** **static** **final** **long** mask = (1L << 48) - 1;
6. **public** **static** **void** main(String[] args) {
7. Random random = **new** Random();
8. **long** v1 = random.nextInt();
9. **long** v2 = random.nextInt();
10. **long** v3 = random.nextInt();
11. **long** v4 = random.nextInt();
12. System.out.println("v1=" + v1 + "**\n**v2=" + v2 + "**\n**v3=" + v3 + "**\n**v4=" + v4);
13. **for** (**int** i = 0; i < 65536; i++) {
14. **long** seed = (((**long**) v1) << 16) + i;
15. **int** nextInt = (**int**)(((seed \* multiplier + addend) & mask) >>> 16);
16. **if** (nextInt == v2) {
17. System.out.println("Seed found: " + seed);
18. Random crackingRandom = **new** Random();
19. **try** {
20. Field privateSeedField = Random.**class**.getDeclaredField("seed");
21. privateSeedField.setAccessible(**true**);
22. AtomicLong crackingSeed = (AtomicLong)privateSeedField.get(crackingRandom);
23. crackingSeed.set(seed);
24. }**catch**(Exception e) {
25. System.out.println(e.toString());
26. System.exit(1);
27. }
28. **long** cv1 = crackingRandom.nextInt();
29. **long** cv2 = crackingRandom.nextInt();
30. **long** cv3 = crackingRandom.nextInt();
31. **long** cv4 = crackingRandom.nextInt();
32. System.out.println("cv1=" + cv1 + "\ncv2=" + cv2 + "\ncv3=" + cv3 + "\ncv4=" + cv4);
33. break;
34. }
35. }
36. }
37. }

Вывод данной программы будет примерно таким:

1. v1 = -1184958941
2. v2 = 274285127
3. v3 = -1566774765
4. v4 = 30466408
5. Seed found: -77657469128792
6. cv1 = 274285127
7. cv2 = -1566774765
8. cv3 = 30466408
9. cv4 = -803980434

Несложно понять, что мы нашли не самый первый seed, а seed, используемый при генерации второго числа. Для нахождения первоначального seed необходимо провести несколько операций, которые Java использовала для преобразования seed, в обратном порядке.

1. **public** **static** **long** getPreviousSeed(**long** prevSeed) {
2. **long** seed = prevSeed;
3. seed -= addend;
4. **long** result = 0;
5. **for** (**int** i = 0; i < 48; i++) {
6. **long** mask = 1L << i;
7. **long** bit = seed & mask;
8. result |= bit;
9. **if** (bit == mask) {
10. seed -= multiplier << i;
11. }
12. }
13. System.out.println("Previous seed: " + result);
14. **return** result;
15. }

И теперь в исходном коде заменим  
crackingSeed.set(seed);  
на  
crackingSeed.set(getPreviousSeed(seed));  
  
И всё, мы успешно взломали ГПСЧ в Java.

# Создание ГПСЧ.

Ломать, конечно, весело, но намного интереснее что-то создавать. Поэтому было решено все-таки сделать что-то сложнее, чем разбор чужого кода.

Для ГПСЧ требуется источник энтропии, любой – время, капли, движение ядер в частицах. Последние, на самом деле, довольно редко для этого используются. Однако нам, используя ПК, доступен другой источник – процессорные тики.

Для получения этих чисел (процессорных тиков) я воспользовался встроенным asm компилятором в c++ и командой **rdtsc.**

1. static inline uint64\_t get\_cycles()
2. {
3. uint64\_t t;
4. \_\_asm volatile ("rdtsc" : "=A"(t));
5. return t;
6. }

Генерировать числа я решил с помощью простых сдвигов битов. Это рекурсивный генератор, который в состоянии генерировать устойчивые наборы битов.

В моей реализации он выглядит так:

1. static u32 \_\_random(struct rnd\_state \*state)
2. {
3. #define SHIFT(s,a,b,c,d) ((s&c)<<d) ^ (((s <<a) ^ s)>>b)
5. state->s1 = SHIFT(state->s1, 13, 19, 4294967294UL, 12);
6. state->s2 = SHIFT(state->s2, 2, 25, 4294967288UL, 4);
7. state->s3 = SHIFT(state->s3, 3, 11, 4294967280UL, 17);
9. return (state->s1 ^ state->s2 ^ state->s3);
10. }

Некий сдвиговый рекурсивный генератор. Определена директива, для простоты пользования формулой и с помощью каких-то значений возможна генерация бит. Данный генератор, кроме всего прочего, может использоваться и в криптографии. Ведь с одними и теми же числами на входе мы можем получать те же самые последовательности. Т.е. в определенных случаях числа можно использовать для блочного шифрования.

Пример использования:

1. rnd\_state \*st = new rnd\_state();
2. st->s1 = 5;
3. st->s2 = 15;
4. st->s3 = 25;
5. for(int i=0; i<=3; i++) {
6. cout << \_\_random(st) << endl;
7. }

Данный пример сгенерирует 4 случайных числа в зависимости от заданных значений (числа взяты случайно).

Однако, если использовать описанную в начале функцию для получения процессорных тиков, можно модифицировать данный генератор и получить полностью рабочий ГПСЧ.

1. rnd\_state \*st = new rnd\_state();
2. st->s1 = get\_cycles();
3. st->s2 = get\_cycles() + 200L;
4. st->s3 = get\_cycles() + 200200L;
5. for(int i=0; i<=3; i++) {
6. cout << \_\_random(st) << endl;
7. }

В результате выполнения приведенного кода мы получим случайные числа, восстановить порядок которых не представляется возможным, если предварительно не сохранить порядок тиков, которые мы получили с помощью обращения к процессору.

# Программная реализация ГПСЧ

В практической части курсовой работы реализуется генератор псевдослучайных чисел с источником энтропии. Источником энтропии выступает количество процессорных тиков.

private int ticks()

{

int result = Environment.TickCount & Int32.MaxValue;

return result;

}

Реализуется класс для генерации такого рода чисел: rg (RandomGenerator) с методами, перечисленными ниже.

|  |  |
| --- | --- |
| Void rg() | Конструктор, задающий значение ключа равному количеству процессорных тиков. |
| Void rg(int) | Конструктор, задающий значение ключа равному передаваемому параметру. |
| Void setSeed() | Установка ключа равному количеству процессорных тиков. |
| Void setSeed(int) | Установка ключа равному передаваемому параметру. |
| Ulong generateNext() | Генерация следующего случайного числа. |
| List<Ulong> generateList(int) | Генерация списка случайных чисел длинной равной значению передаваемого параметра. |

Для генерации был выбран сдвиговый генератор, описанный в теоретической части.

ulong a = 6364136223846793005;

ulong c = 1442695040888963407;

ulong x = a \* this.seed + c;

ulong y = a \* x + c;

ulong z = a \* y + c;

А дальше ключом устанавливается значение переменной z. Это делается для того, чтобы получить следующее значение в цепочке и сохранить криптографическую ценность алгоритма.

Если генерировать числа каждый раз с учетом нового семени (seed), то для расшифровки сообщения, зашифрованного с помощью этой последовательности, понадобится сохранить всю последовательность. А в данном случае достаточно будет только начального значения.

Больше информации о программной реализации можно получить, просмотрев “описание программы” и “Текст программы”.

# Вывод

Генераторы псевдослучайных чисел очень важны для научных исследований и криптографии. Однако, из полученной в ходе написания курсовой работы информации, я могу сделать вывод, что очень тяжело создать действительно работающий ГПСЧ, который подходил бы по всем параметрам как ученым, так и людям, связанным с криптографией.

Действительно работающие генераторы либо засекречены, либо работают очень медленно, используя внешние источники энтропии.

Использование внутренних источников может быть ошибочным из-за того, что шум, генерируемый, например, звуковой картой в ПК не так уж и случаен, а потому и последовательности, которые можно было бы генерировать, используя этот источник энтропии не могут похвастаться хорошими результатами прохождения специализированных тестов.