1,2, 314, 271.. SecureRandom in JVM. Hitchhiker's Guide

Mikhail Dudarev
@MikhailDudarev
Licel Corporation
JPoint 2016

О нас

Licel Corporation

Web: https://licelus.com

Поиграем ?)

https://xkcd.com/221/

```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll
              // guaranteed to be random
}
```

https://geektimes.ru/post/274048/

9 апреля в 00:08

Глава отдела информационной безопасности лотереи в США «доработал» генератор случайных чисел и выиграл миллионы долларов

Финансы в IT-индустрии, Информационная безопасность

Где используется случайность ? Реальный мир.

Где используется случайность ?
Наш с вами мир ☺

1. Генерация ключей
2. Шифрование (paddings/IV)
3. ЭЦП
4. Криптопротоколы (nonce)

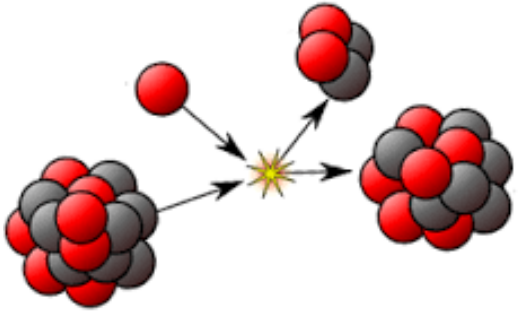
Где используется случайность ?
Наш с вами мир ☺

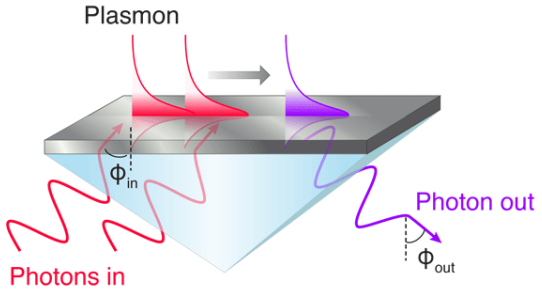
1. Маркеры доступа
2. CRSF-токены
3. Идентификаторы сессий
4. Одноразовые пароли

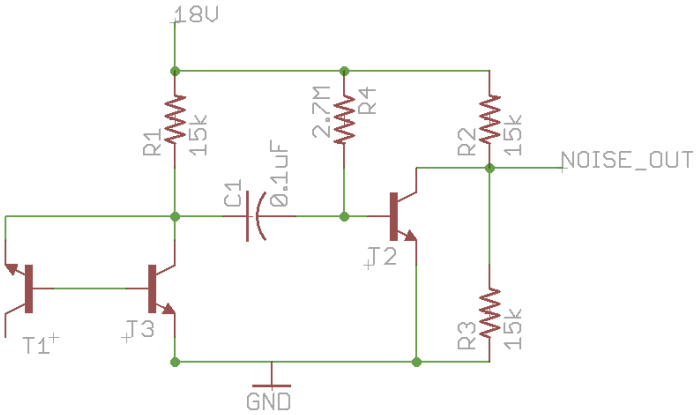
“Главный” источник энтропии

Реально главный источник энтропии

Квантовые источники энтропии



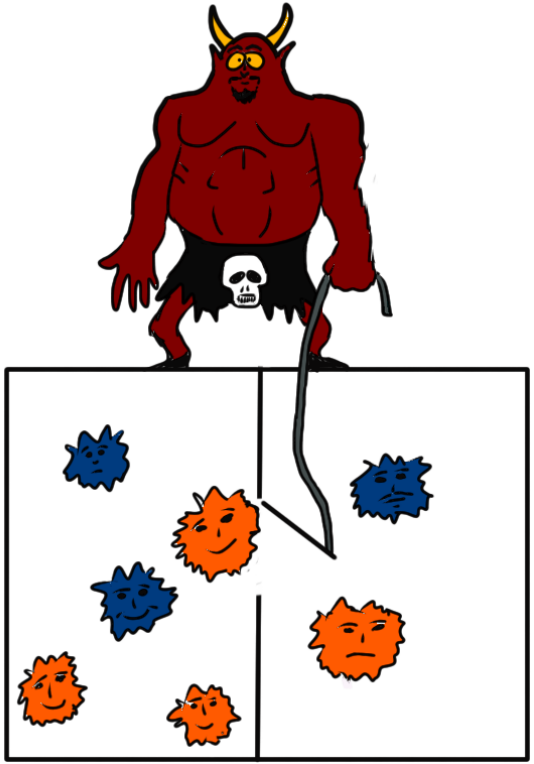




Биологические источники энтропии

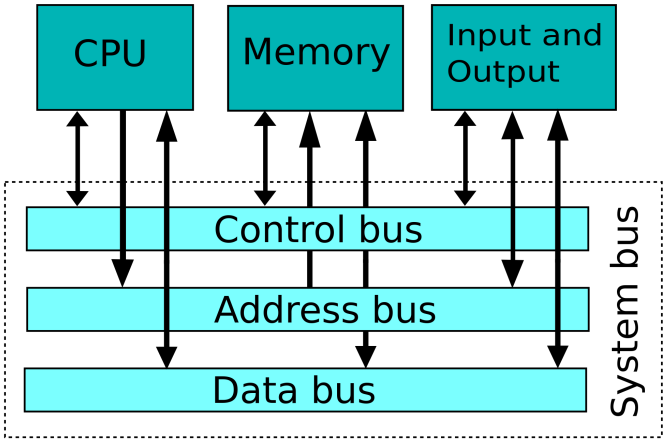
?

Мифические источники энтропии



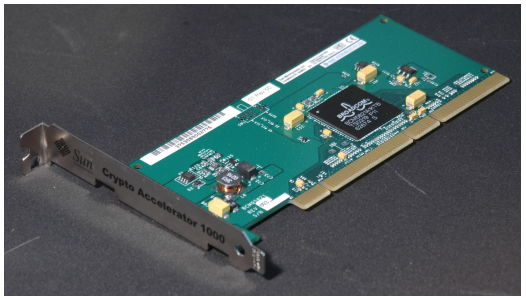
Программные источники энтропии

- Системное время и время с последней загрузки
- Счетчики с CPU
- Замеры переключений контекста и системных прерываний
- Замеры операций ввода/вывода

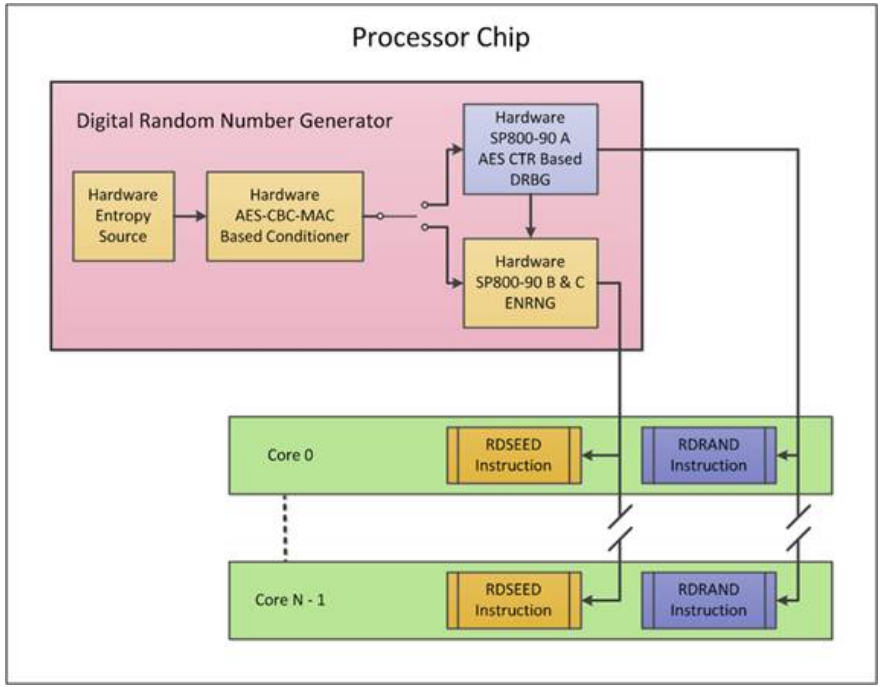


Аппаратные источники энтропии

- Встроенные в микропроцессор
  - Intel DRNG
- Внешние платы расширения
- Смарт-карты



RdRand & RdSeed



The Register®
Biting the hand that feeds IT

DATA CENTRE SOFTWARE NETWORKS SECURITY INFRASTRUCTURE DEVOPS BUSINESS HARDWAR

Security

Torvalds shoots down call to yank 'backdoored' Intel RdRand in Linux crypto

'We actually know what we are doing. You don't' says kernel boss

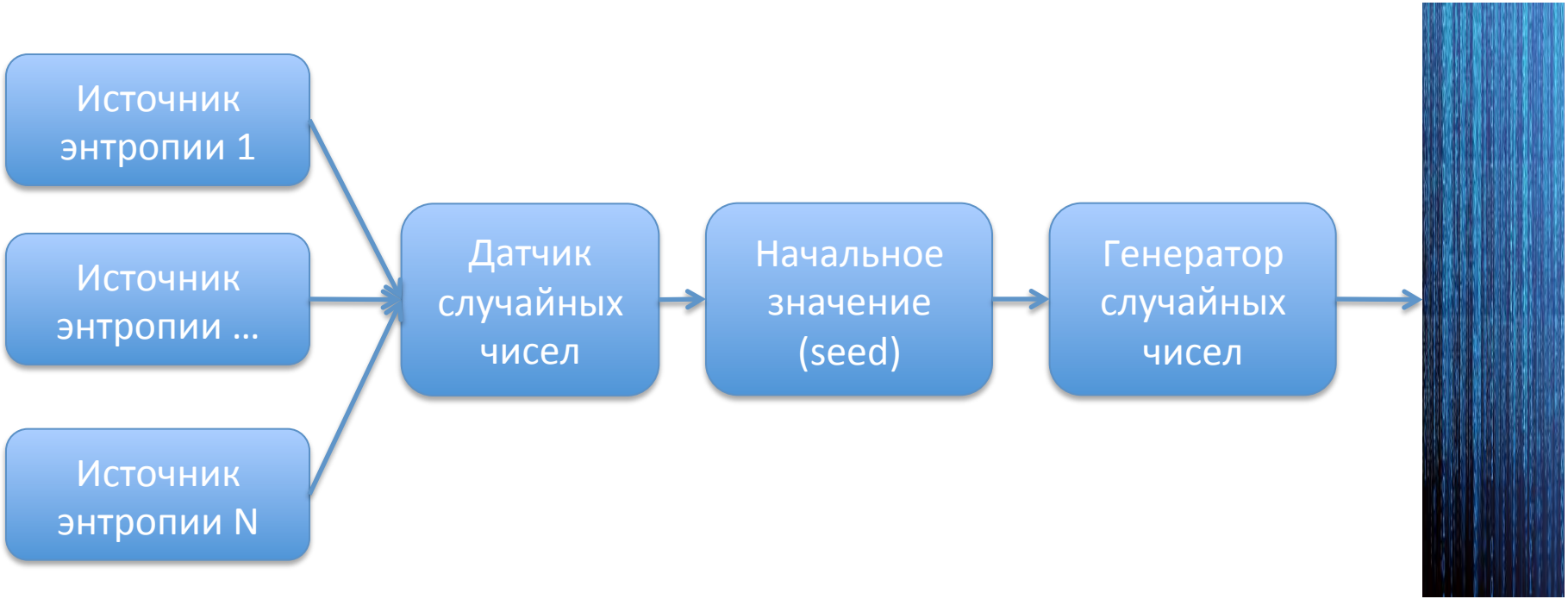
10 Sep 2013 at 17:03, Gavin Clarke

73 10

Энтропии на всех не хватит


DON'T PANIC

Псевдослучайный генератор случайных чисел (PRNG)



Тестируем случайность

А как дела в Java ?

1. java.util.Random
   - nextXXX()
2. java.security.SecureRandom
   - nextXXX()
   - generateSeed()

java.util.Random

java.util.Random под капотом

```
protected int next(int bits) {
    long oldseed, nextseed;
    AtomicLong seed = this.seed;
    do {
            oldseed = seed.get();
            nextseed = (oldseed * multiplier + addend) & mask;
        } while (!seed.compareAndSet(oldseed, nextseed));
    return (int)(nextseed >>> (48 - bits));
}
```

java.util.Random под капотом

seed = (seed * multiplier + addend) mod (2 ^ precision)

multiplier == 25214903917

addend == 11

precision== (1L << 48) - 1

Предсказываем java.util.Random

```
Random random = new Random();
long v1 = random.nextInt();
long v2 = random.nextInt();
for (int i = 0; i < 65536; i++) {
    long seed = v1 * 65536 + i;
    if (((seed * multiplier + addend) & mask) >>> 16) == v2) {
        System.out.println("Seed found: " + seed);
        break;
    }
}
```

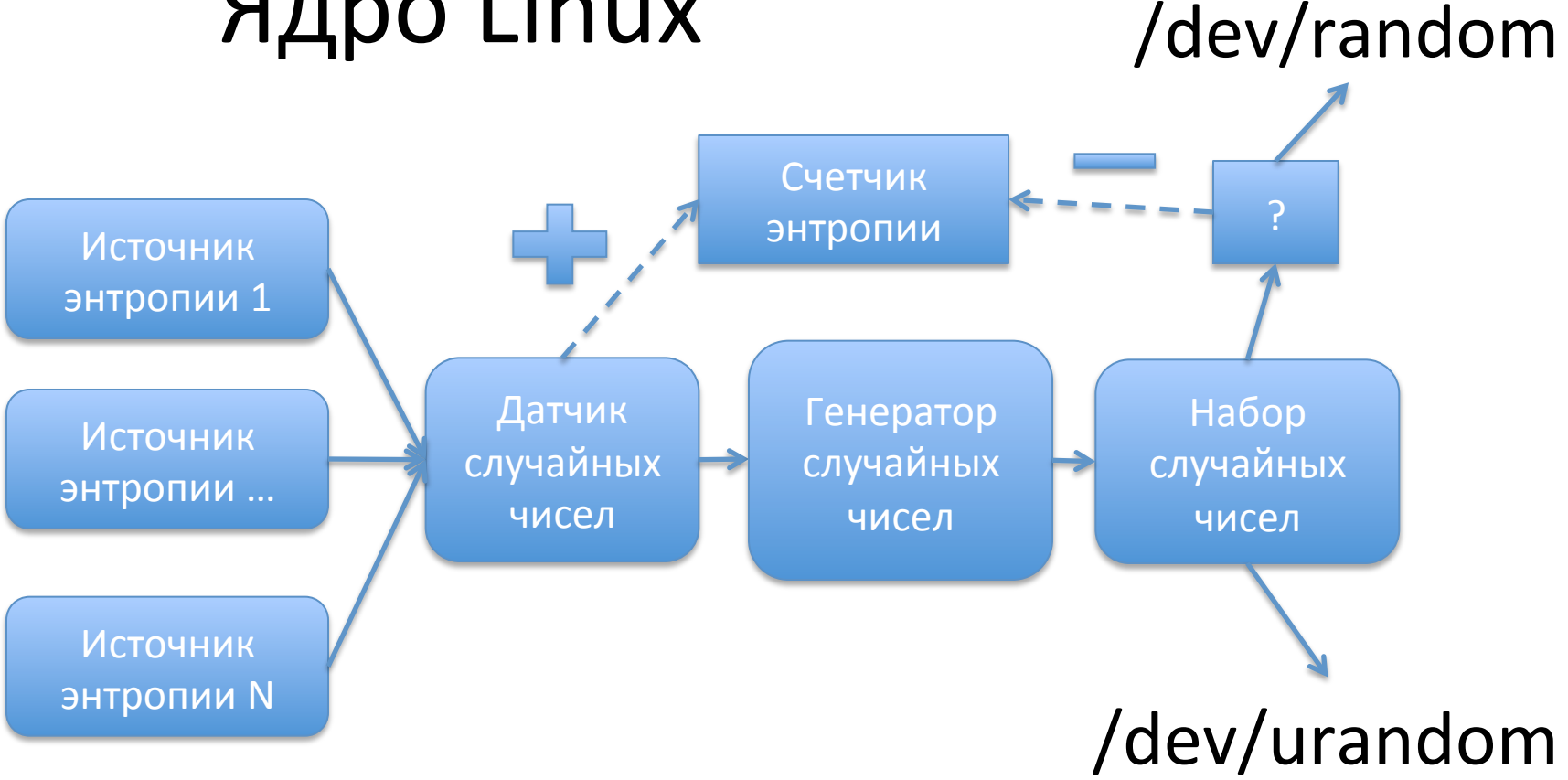
java.security.SecureRandom

```
SecureRandom rnd = new SecureRandom();

byte[] rndData = new byte[32];
rnd.nextBytes(rndData);

byte[] seed = rnd.generateSeed(32);
```

Реализации java.security.SecureRandom

	Windows	Linux	Solaris	OS X
PKCS11			X	
NativePRNG		X	X	X
SHA1PRNG	X	X	X	X
NativePRNGBlocking		X	X	X
NativePRNGNonBlocking		X	X	X
Windows-PRNG	X			

Ядро Linux


/dev/random
Счетчик энтропии
?
Источник энтропии 1
Источник энтропии …
Источник энтропии N
Датчик случайных чисел
Генератор случайных чисел
Набор случайных чисел
/dev/urandom

NativePRNG

```
rnd = SecureRandom.
          getInstance("NativePRNG");

// /dev/urandom >>
rnd.nextBytes(rndData);

// /dev/random >>
byte[] seed = rnd.generateSeed(32);
```

NativePRNGBlocking

```
rnd = SecureRandom.
          getInstance("NativePRNGBlocking");

// /dev/random >>
rnf.nextBytes(rndData);

// /dev/random >>
byte[] seed = rnd.generateSeed(32);
```

NativePRNGNonBlocking

```
rnd = SecureRandom.
          getInstance("NativePRNGNonBlocking");

// /dev/urandom >>
rnd.nextBytes(rndData);

// /dev/urandom >>
byte[] seed = rnd.generateSeed(32);
```

SHA1PRNG

```
// egdSource (Native/URL/FallBack) >>
rnd = SecureRandom.
          getInstance(”SHA1PRNG");

// SHA1(state) >>
random.nextBytes(rndData);

// egdSource (Native/URL/FallBack) >>
byte[] seed = random.generateSeed(32);
```

Что же выбрать ?

NativePRNG ?

NativePRNGNonBlocking ?

NativePRNGBlocking ?

SHA1PRNG ?

new SecureRandom() ?

С Java 8 выбирай SecureRandom.getInstanceStrong()

JEP 273: DRBG-Based SecureRandom

- Java 9
- Новые кроссплатформенные и криптографические стойкие (NIST 800-90Ar1) датчики и генераторы псевдослучайных чисел
  - Hash_DRBG (SHA-512)
  - HMAC_DRBG (HmacSHA512)
  - CTR_DRBG (AES-256)

Выводы

- Никогда не используйте java.util.Random
- Для генерации ключей и криптографических примитивов

```
SecureRandom rnd = SecureRandom.getInstanceStrong();

KeyPairGenerator kpg = KeyPairGenerator.
                                        getInstance("RSA");
kpg.initialize(4096, rnd);
```

Выводы

- Для генерации случайных данных

```
SecureRandom strongRnd = SecureRandom.getInstanceStrong();

SecureRandom rnd = SecureRandom.getInstance("SHA1PRNG");

// FIPS рекомендует 440 бит энтропии для SHA1
rnd.setSeed(strongRnd.generateSeed(55));

byte[] rndBytes = new byte[1024];
rnd.nextBytes(rndBytes);
```

Контакты

Email: kinash@licelus.com
Twitter: @ivan_kinash

Email: dudarev@licelus.com
Twitter: @MikhailDudarev

Licel Corporation
Web: https://licelus.com