Random Number Generators

Błażej Sowa

Definition

Random Numbers - numbers that occur in a sequence such that two conditions are met:

1. the values are uniformly distributed over a defined interval or set
2. it is impossible to predict future values based on past or present ones.

Practical applications and uses

- Gambling
- Computer simulation
- Statistical sampling
- Gaming
- **Cryptography**
  - key generation
  - one-time pads
  - salts

Types of RNGs

- **Pseudo-random Number Generator** algorithm that uses mathematical formulas to produce sequences of numbers approximating the properties of random numbers

- **True Random Number Generator** device that generates random numbers from a physical process, rather than a computer program

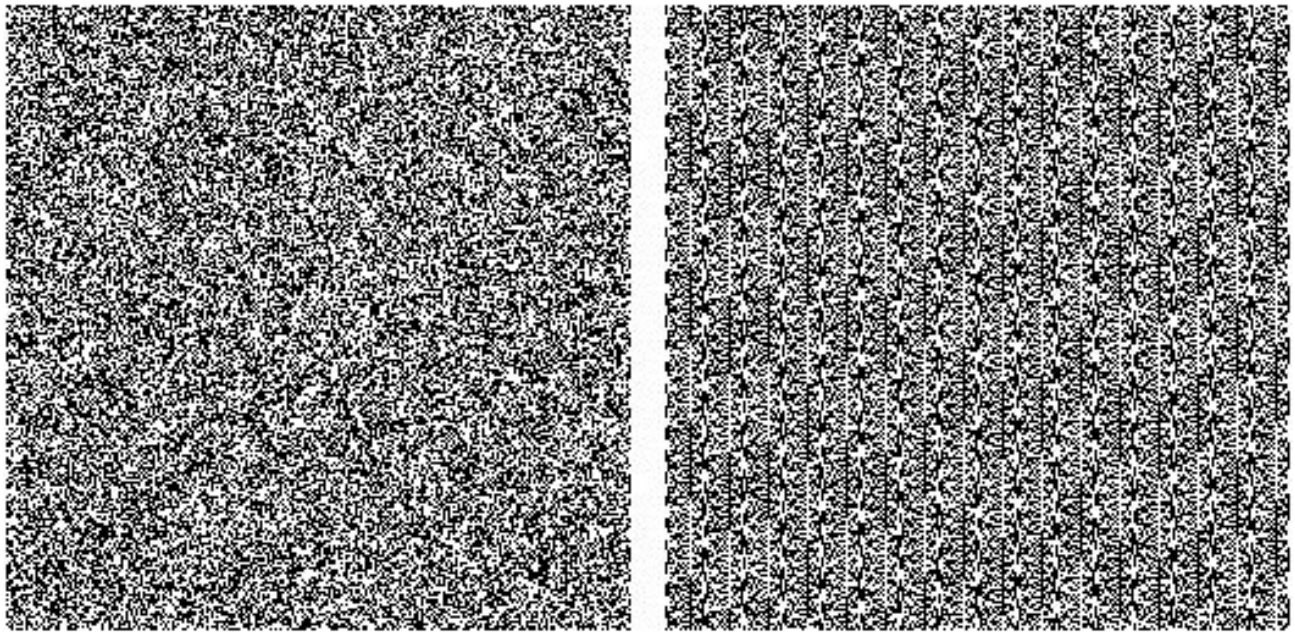
PRNG vs TRNG

Characteristic	Pseudo-Random Number Generators	True Random Number Generators
Efficiency	Excellent	Poor
Determinism	Deterministic	Nondeterministic
Periodicity	Periodic	Aperiodic

Randomness Tests

Simple Visual Analysis

Representing in a graphical or visual data and observing by human eye



TestU01

- A software library, implemented in the ANSI C language, that offers a collection of utilities for the empirical statistical testing of random number generators
- Implements several types of random number generators in generic form
- Offers several batteries of tests including
  - "Small Crush" (10 tests),
  - "Crush" (96 tests), and
  - "Big Crush" (160 tests)

TestU01 - Example tests

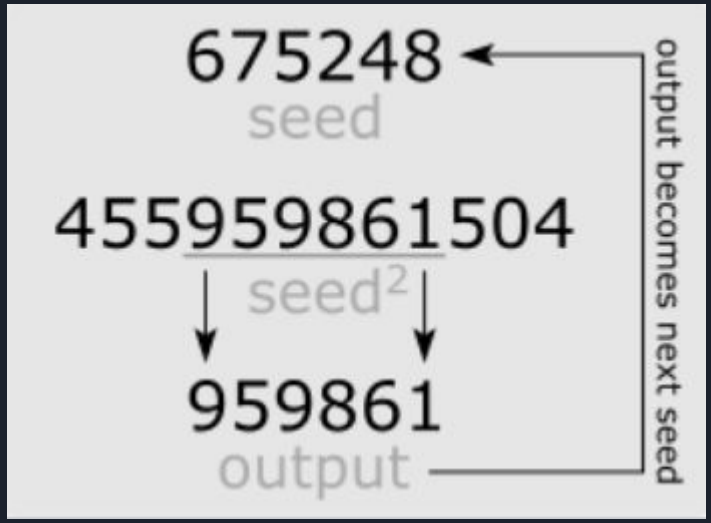
- **BirthdaySpacings** - Choose random points on a large interval. The amount of spacings between the points that occur more than once should be asymptotically exponentially distributed.
- **Collisions** - Set up a large number of groups relative to the length of the possible values, then calculate number of collisions (number of times a point hits a cell already occupied)
- **Simplified Poker** - Generate n groups of k integers, by making nk calls to the generator, and for each group compute the number of distinct integers in the group

Examples of PRNG

Middle-square method

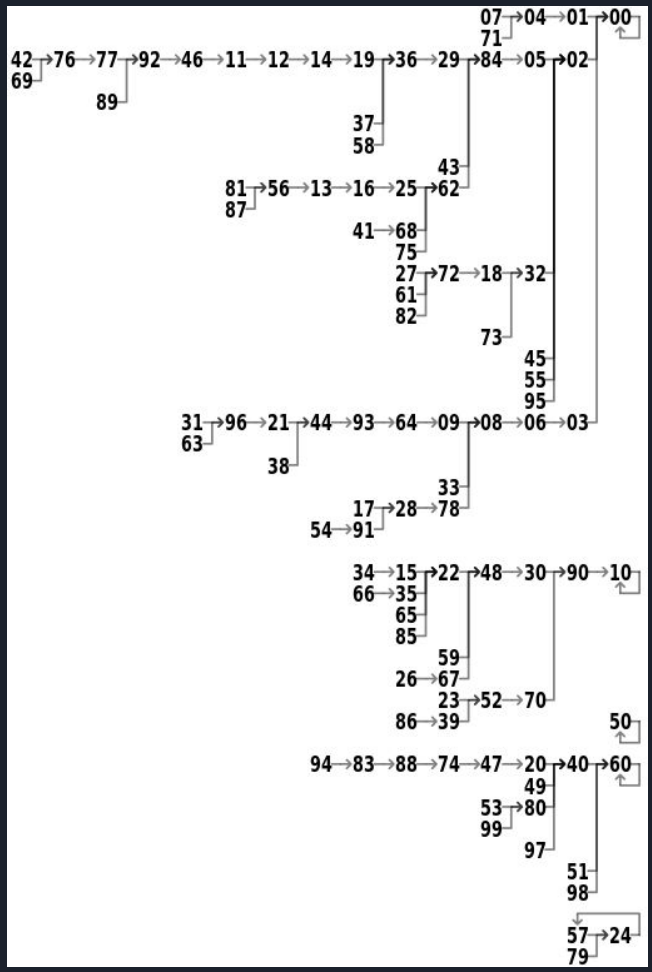
- Developed by John von Neumann around 1946
- First ever PRNG?

- The next number in a sequence is obtained by squaring the previous one and slicing out the middle digits



Middle-square method

- no matter what seed value is used, the sequence will eventually fall into a short repeated cycle of numbers
- Nicholas Metropolis reported sequences of 750,000 digits before "destruction" by means of using 38-bit numbers



Directed graph of all 100 2-digit pseudorandom numbers

Middle-square method

"Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin."

John von Neumann

Middle-square method

"Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin."

John von Neumann

Can it be improved?

Middle Square Weyl Sequence PRNG

- A new implementation of John von Neumann's middle square random number generator proposed by Bernard Widynski in 2017
- A Weyl sequence is utilized to keep the generator running through a long period:

  (k, 2k, 3k, 4k, ...

  The sequence for any odd integer $k < 2^n$ is equidistributed modulo 2^n
- The period of this PRNG is at least 2^n

Middle Square Weyl Sequence PRNG

1. The previous result is squared (modulo 2^{64})
2. Next value of Weyl sequence is added
3. The middle is extracted by shifting right 32 bits (most significant bits of the lower 64 bits of the result)

```c
#include <stdint.h>

uint64_t x = 0, w = 0, s = 0xb5ad4eceda1ce2a9;

inline static uint32_t msws() {

   x *= x;
   x += (w += s);
   return x = (x>>32) | (x<<32);

}
```

Middle Square Weyl Sequence PRNG

1. The previous result is squared (modulo 2^{64})
2. Next value of Weyl sequence is added
3. The middle is extracted by shifting right 32 bits (most significant bits of the lower 64 bits of the result)

```c
#include <stdint.h>

uint64_t x = 0, w = 0, s = 0xb5ad4eceda1ce2a9;

inline static uint32_t msws() {

   x *= x;
   x += (w += s);
   return x = (x>>32) | (x<<32);

}
```

- 2 different **x** values can produce the same sequences
- The constant **s** should be non-zero in the upper 32 bits and 1 in the least significant bit. That gives us (2^{63} - 2^{31}) different seed values
- The MSWS generator passes all the tests in the stringent BigCrush battery in TestU01
- It's fast!

Linear congruential generator

$$X_{n+1} = (aX_n + c) \bmod m$$

where, X is the sequence of pseudorandom values, and:

- m, $0 < m$ - the modulus
- a, $0 < a < m$ - the multiplier
- c, $0 \leq c < m$ - the increment
- X_0, $0 \leq X_0 < m$ - the seed

Linear congruential generator

$$X_{n+1} = (aX_n + c) \bmod m$$

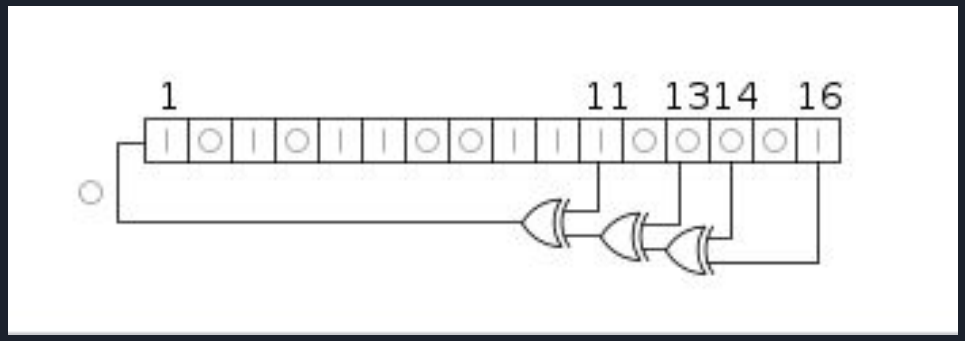
- included in built-in rand() functions in runtime libraries of various compilers, such as: glibc, Turbo Pascal, Java's java.util.Random
- fast and require minimal memory
- with appropriate choice of parameters, the period is known and long
- a modulo-2^{64} LCG which returns the high 32 bits passes TestU01's SmallCrush suite and a 96-bit LCG passes the most stringent BigCrush suite.

Linear congruential generator

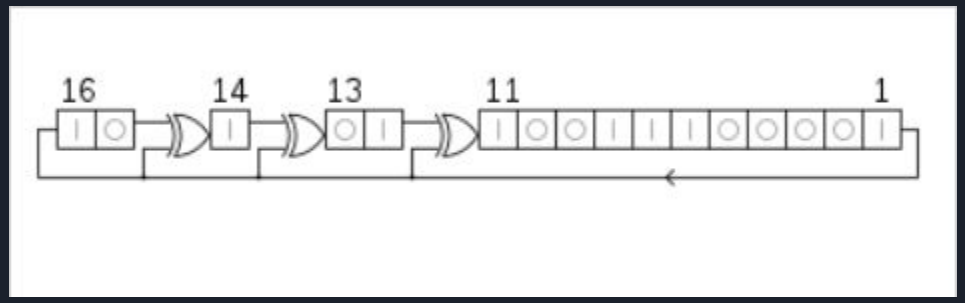
- **m prime, c=0**
  - The period is m−1 if the multiplier a is chosen to be a primitive element of the integers modulo m
- **m a power of 2, c=0**
  - allows the modulus operation to be computed by simply truncating the binary representation
  - maximal period m/4, achieved if a ≡ 3 or a ≡ 5 (mod 8)
  - low bits have a shorter period than the high bits. The lowest-order bit of X never changes (X is always odd), and the next two bits alternate between two states.
- **c≠0**
  - correctly chosen parameters allow a period equal to m, for all seed values. This will occur if and only if:
    i. m and c are relatively prime,
    ii. a-1 is divisible by all prime factors of m,
    iii. a-1 is divisible by 4 if m is divisible by 4

Linear Feedback Shift Register

A shift register whose input bit is a linear function of its previous state (most commonly XOR function)



Fibonacci LFSR



Galois LFSR

Linear Feedback Shift Register

- Fibonacci and Galois LFSRs produce the same output stream, but software implementation of Galois LFSR is more efficient, as the XOR operations can be implemented a word at a time

  if (n & 1) { n = (n >> 1) ^ t; }
  else { n = (n >> 1); }
- A maximum-length LFSR has a period of 2^m - 1
- The arrangement of taps for feedback in an LFSR can be expressed as a polynomial mod 2

  $$x^{16} + x^{14} + x^{13} + x^{11} + 1$$

  The LFSR is maximal-length if and only if the corresponding feedback polynomial is **primitive**
- LFSRs can be implemented in hardware, and this makes them useful in applications that require very fast generation of a pseudo-random sequence

Bits	# Perfect Feedbacks
4	2
5	6
6	6
7	18
8	16
9	48
10	60
11	176
12	144
13	630
14	756
15	1,800
16	2,048
17	7,710
18	7,776
19	27,594
20	24,000
21	84,672
22	120,032
23	356,960
24	276,480
25	1,296,000
26	1,719,900
27	4,202,496
28	4,741,632
29	18,407,808
30	17,820,000
31	69,273,666
32	67,108,864

KISS (Keep It Simple Stupid)

- Family of pseudorandom number generators introduced by George Marsaglia
- The original 1993 generator combines:
  - linear congruential generator
  - 2 linear feedback shift registers

  has a period 2^{95}, good speed and good statistical properties; however, it fails the LinearComplexity test in the Crush and BigCrush batteries
- A newer version from 1999 combines:
  - linear congruential generator
  - Xorshift (subset of LFSRs, more efficient)
  - 2 multiply-with-carry generators (variation of LCG)

  has a period of 2^{123} and passes all tests in TestU01

KISS (Keep It Simple Stupid)

"A random number generator is like sex:
When it's good, its wonderful;
And when it's bad, it's still pretty good."

Add to that, in line with my recommendations on combination generators;

"And if it's bad, try a twosome or threesome."

George Marsaglia

Cryptographically secure pseudorandom number generators (CSPRNG)

CSPRNG

Most PRNGs are not suitable for use in cryptography. To be considered Cryptographically Secure they must (besides passing statistical tests):

- Satisfy the next-bit test - given the first k bits of a random sequence, there is no polynomial-time algorithm that can predict the (k+1)th bit with probability of success non-negligibly better than 50%
- Withstand "state compromise extensions" - If part of it's state has been revealed or correctly guessed, it should be impossible to reconstruct the stream of random numbers prior to the revelation

Blum Blum Shub

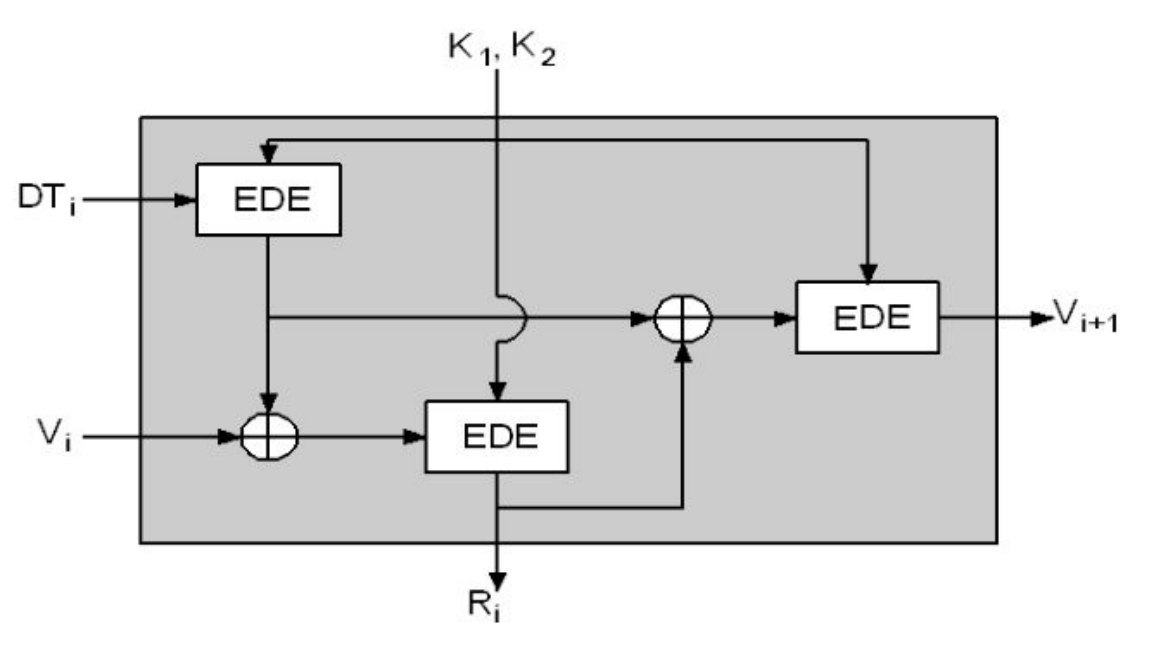
$$X_{n+1} = X_n^2 \bmod M$$

- M = pq is the product of two large primes p and q
- The seed X_0 should be an integer that is co-prime to M
- The two primes, p and q, should both meet property $p \equiv 3 \pmod 4$
- the output is commonly either the bit parity of X_{n+1} or one or more of the least significant bits of X_{n+1}

- security of BBS comes from the computational difficulty of solving the quadratic residuosity problem

- requires to use of multi-precision number system
- it’s very inefficient

ANSI X9.17 PRNG



EDE - triple-DES encryption function

K_1, K_2 - pair of 56-bit keys used for 3DES module

DT - 64-bit representation of current date/time

V - 64-bit seed value

R - 64-bit pseudorandom value

Entropy

- **Entropy (information theory)**
  Average amount of information produced by a stochastic source of data. When the data source has a lower-probability value (i.e., when a low-probability event occurs), the event carries more "information" ("surprisal") than when the source data has a higher-probability value

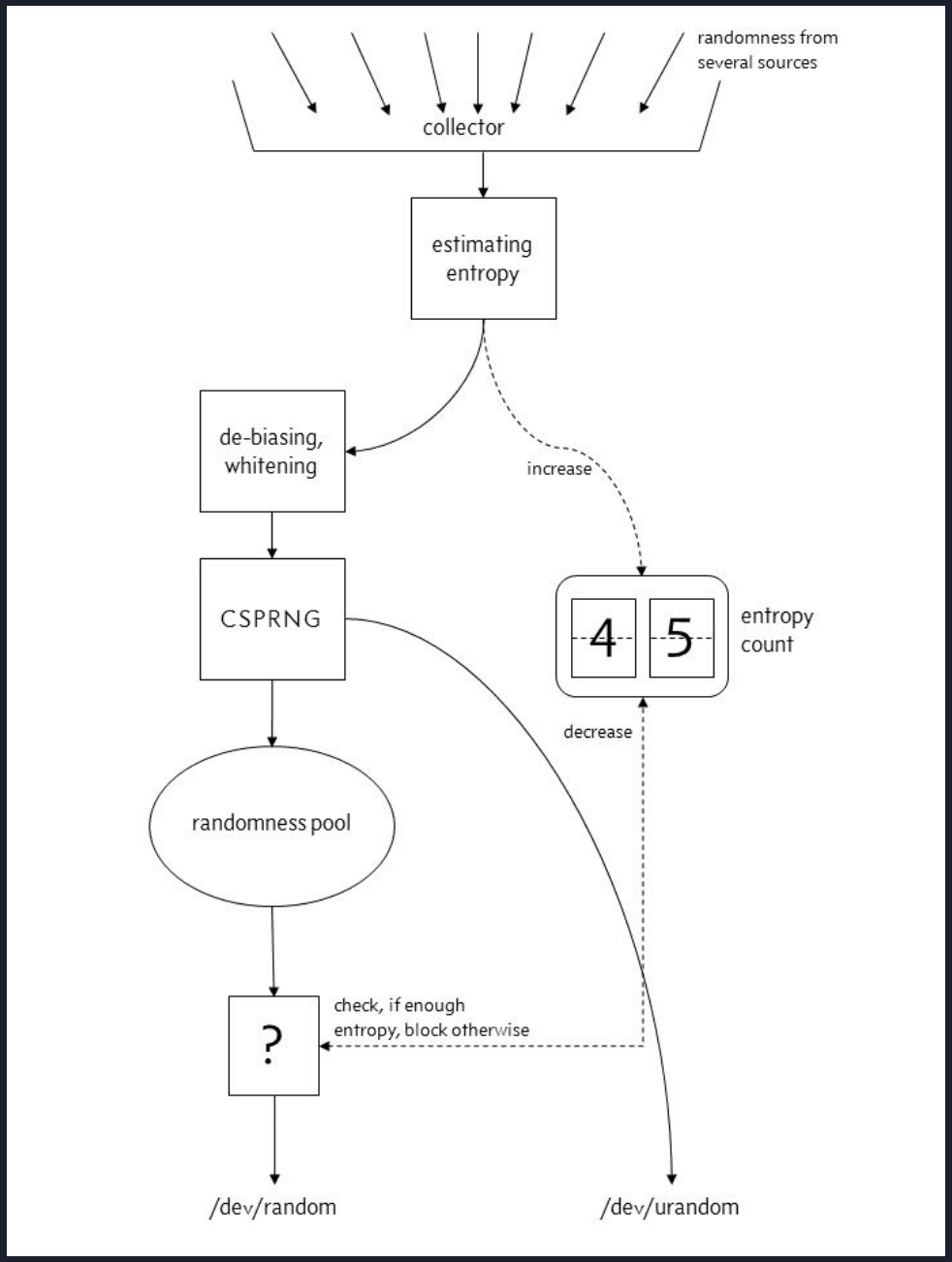
- **Entropy (Computing)**
  Randomness collected by an operating system or application for use in cryptography or other uses that require random data, often collected from hardware sources

Sources of entropy

- Commonly Used
  - timings between key presses
  - mouse position and movements
  - timings between interrupts and related events
- Potential
  - noise from microphone input
  - noise from TV or radio tuner
  - air turbulence inside disk drive
  - camera pointed at constantly moving picture

Linux kernel

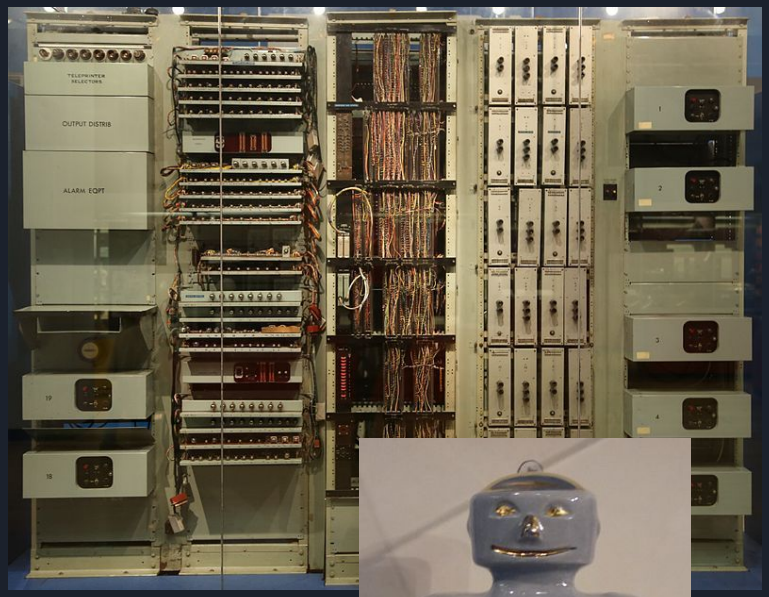
- generates entropy from keyboard timings, mouse movements, and IDE timings
- makes the random character data available to other operating system processes through the special files:
  - **/dev/random**
    creates random values from entropy pool. When read, the /dev/random device will only return random bytes within the estimated number of bits of noise in the entropy pool. When the entropy pool is empty, reads from /dev/random will block until additional environmental noise is gathered
  - **/dev/urandom**
    uses a CSPRNG periodically reseeded with bits from entropy pool to output a stream of random data. The call will never block


randomness from
several sources
collector
estimating
entropy
de-biasing,
whitening
increase
CSPRNG
4
5
entropy
count
decrease
randomness pool
?
check, if enough
entropy, block otherwise
/dev/random
/dev/urandom

Hardware Random Number Generators

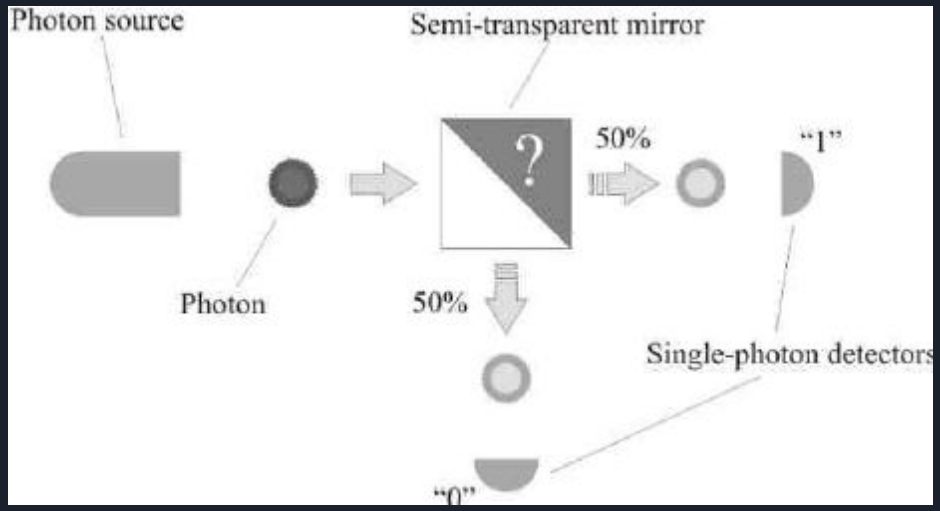
ERNIE (Electronic Random Number Indicator Equipment)

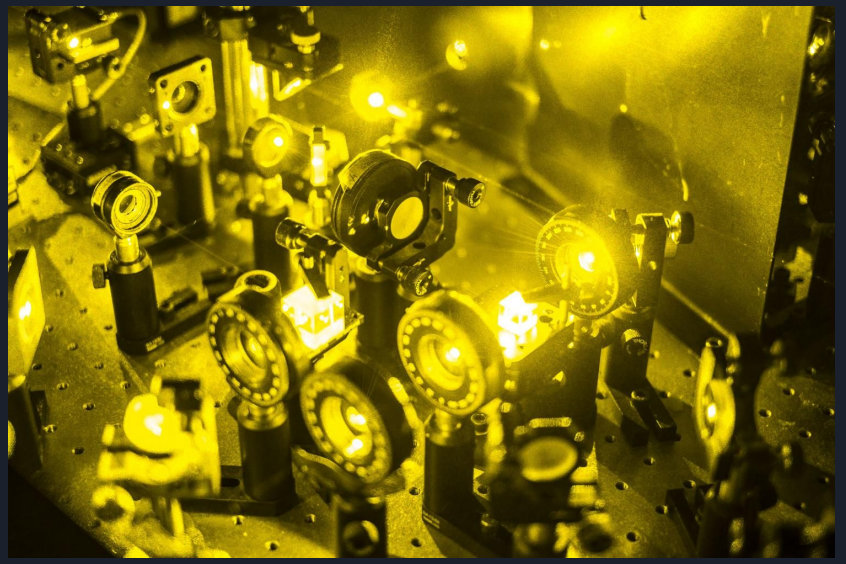
- designed by the now-famous Bletchley Park WWII codebreaking team in the 1940s
- used to generate random numbers for the UK Premium Bond lottery
- generated bond numbers based on the signal noise created by neon tubes
- To quell fears about the fairness and accuracy of ERNIE, the Post Office made a great documentary called *The Importance of Being E.R.N.I.E*

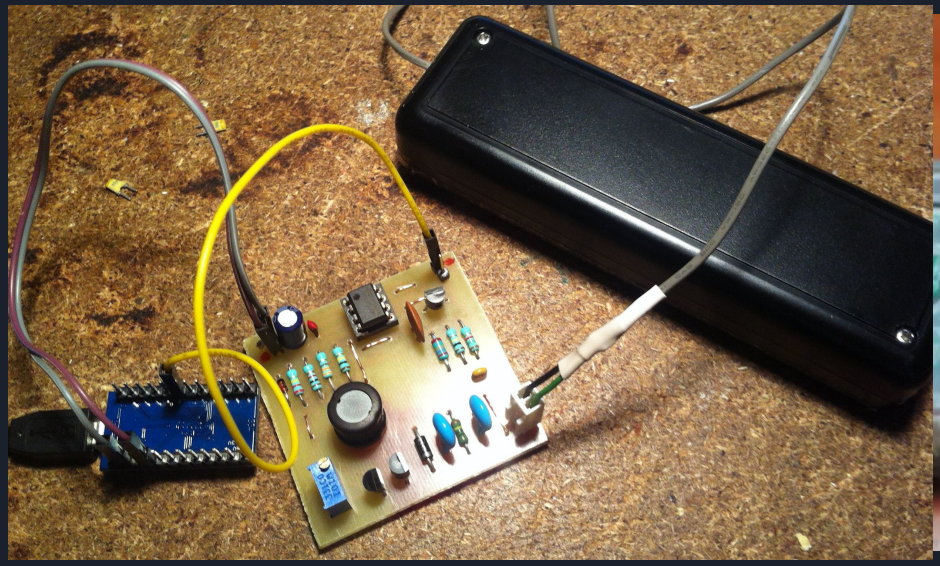


Quantum Random Number Generators

- Quantum mechanics predicts that certain physical phenomena, are fundamentally random and cannot, in principle, be predicted. Because of that, they are the ‘gold standard’ for random number generation
- Some quantum phenomena used for random number generation include:
  - A nuclear decay radiation source
  - Photons travelling through a semi-transparent mirror
  - Shot noise, a quantum mechanical noise source in electronic circuits


Photon source
Semi-transparent mirror
?
50%
"1"
Photon
50%
Single-photon detectors
"0"






QUANTUM R.N.G.
WARNING!
HIGH RANDOMNESS!

LavaRand

- Designed and patented by Silicon Graphics around 1996
- Camera directed to a wall o lava lamps
- generates a fixed-size chunks of fresh entropy called a “beacon”. It then mixes this beacon into the entropy system (on Linux, by writing the beacon to /dev/random)
- As of 2017, Cloudflare maintains a similar system of lava lamps for securing approximately 10% of the Internet's traffic

Attacks at RNGs

Attacks at RNGs

- **Direct cryptanalytic attack**
  the attacker is directly able to distinguish between PRNG outputs and random outputs. So an attacker directly exploits a vulnerability in the cryptographic properties of the PRNG system.
- **Input-based attacks**
  the attacker has access to the input of the PRNG and uses knowledge or control over the input to cryptanalyze the PRNG.
- **State compromise extension attacks**
  the internal secret state of the RNG is known at some time and used by the attacker to predict future output or to recover previous outputs

Microsoft Windows 2000/XP RNG “CryptGenRandom”

- In November 2007, Leo Dorrendorf published a paper titled *Cryptanalysis of the Random Number Generator of the Windows Operating System.*
- The paper presented serious weaknesses in Microsoft's approach at the time. The paper's conclusions were based on disassembly of the code in Windows 2000, but according to Microsoft applied to Windows XP as well.
- The paper's attacks are based on the fact that CryptGenRandom uses the stream cipher RC4, which can be run backwards once its state is known
- They also take advantage of the fact that CryptGenRandom runs in user mode, allowing anyone who gains access to the operating system at user level, for example by exploiting a buffer overflow, to get CryptGenRandom's state information for that process.

Debian OpenSSL

- On May 13th, 2008 the Debian project announced that Luciano Bello found an interesting vulnerability in the OpenSSL package they were distributing.
- The bug was introduced in 2006 by 2 lines of code that were removed because they caused the Valgrind and Purify tools to produce warnings
- The side effect of crippling the seeding process for the OpenSSL PRNG. Instead of mixing in random data for the initial seed, the only "random" value that was used was the current process ID (32,768 possible values)
- The vulnerability was promptly patched after it was reported, but any services still using keys that were generated by the old code remain vulnerable

Predictable Netscape seed

- Early versions of Netscape's Secure Socket Layer (SSL) encryption protocol used pseudo-random quantities derived from a PRNG seeded with three variable values:
  - time of the day
  - process ID
  - parent process ID
- These quantities are often relatively predictable, and so have little entropy and are less than random, and so that version of SSL was found to be insecure as a result.
- The problem in the running code was discovered in 1995 by Ian Goldberg and David Wagner, who had to reverse engineer the object code because Netscape refused to reveal the details of its random number generation

Possible Backdoor in Elliptical Curve DRBG

- The U.S. National Institute of Standards and Technology has published a collection of "deterministic random bit generators" it recommends as NIST Special Publication 800-90
- One of the generators, Dual_EC_DRBG, was favored by the National Security Agency.
- In August 2007, Dan Shumow and Niels Ferguson of Microsoft showed that the constants could be constructed in such a way as to create a kleptographic backdoor in the algorithm
- In December 2013, Reuters reported that documents released by Edward Snowden indicated that the NSA had paid RSA Security $10 million to make Dual_EC_DRBG the default in their encryption software

Resources and Further Reading

Resources and Further Reading

- brief history of random number generators
  https://medium.freecodecamp.org/a-brief-history-of-random-numbers-9498737f5b6c
- TestU01
  http://simul.iro.umontreal.ca/testu01/tu01.html
- Middle-square method
  https://en.wikipedia.org/wiki/Middle-square_method
- Middle-square Weyl Sequence
  https://arxiv.org/pdf/1704.00358.pdf
- Linear Congruential Generator
  https://en.wikipedia.org/wiki/Linear_congruential_generator
- Linear Feedback Shift Register
  http://datagenetics.com/blog/november12017/index.html
- KiSS (Keep It Simple Stupid)
  https://en.wikipedia.org/wiki/KISS_(algorithm)
  https://eprint.iacr.org/2011/007.pdf
- Cryptographically secure pseudorandom number generators
  https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator

Resources and Further Reading

- entropy
  https://en.wikipedia.org/wiki/Entropy_(computing)
  https://hackaday.com/2017/11/02/what-is-entropy-and-how-do-i-get-more-of-it/
- /dev/random
  https://www.2uo.de/myths-about-urandom/
- ERNIE RNG
  https://en.wikipedia.org/wiki/Premium_Bond#ERNIE
  https://www.youtube.com/watch?time_continue=360&v=rOAfbb5D3Dw
- Quantum Random Number Generators
  https://en.wikipedia.org/wiki/Hardware_random_number_generator#Quantum_random_properties
- Nuclear powered random number generator
  https://hackaday.com/2015/08/16/hackaday-prize-entry-nuclear-powered-random-number-generator/
- LavaRand
  https://en.wikipedia.org/wiki/Lavarand
  https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/

Resources and Further Reading

- RNG Attacks
  https://en.wikipedia.org/wiki/Random_number_generator_attack
- Debian OpenSSL Attack
  https://www.schneier.com/blog/archives/2008/05/random_number_b.html
- Microsofts CyptGenRandom vulnerability
  https://www.theregister.co.uk/2007/11/13/windows_random_number_gen_flawed/
- NSA backdoor
  https://en.wikipedia.org/wiki/Dual_EC_DRBG
  https://blog.cloudflare.com/how-the-nsa-may-have-put-a-backdoor-in-rsas-cryptography-a-technical-primer/