**Random numbers**

Random numbers are a set of numbers that are/or appear completely unpredictable and don’t seem to follow any discernable pattern. A single random number can be a number that was chosen unpredictably from a uniformly distributed set.

Such numbers have a wide range of use in many different fields. Random numbers are important in statistical analysis, probability theory, and modern-day computer simulations and cryptography, of course gambling and video games make a wide use of random numbers as well.

Random number generators (RNG)

Random numbers are most useful for computers, but how can you get random numbers from a computer, which is a purely deterministic logic machine?

You cannot. You can’t get true random numbers while only using a computer, but what you can do is approach true unpredictability, what you can get is;

Pseudorandom number generator (PRNG)

This is an algorithm into which you input a “seed” value and get a seemingly random number as an output. The main characteristic of PRNGs is that their output can be replicated if the same seed is put into the algorithm which was used in the process.

Here’s an early computer-based PRNG, suggested by John von Neumann in 1946. The algorithm is as follows: take any number, square it, remove the middle digits of the resulting number as the "random number", then use that number as the seed for the next iteration. For example, squaring the number "1111" yields "1234321", which can be written as "01234321", an 8-digit number being the square of a 4-digit number. This gives "2343" as the "random" number. Repeating this procedure gives "4896" as the next result, and so on. Von Neumann used 10 digit numbers, but the process was the same.

A black background with white text

Description automatically generated

PRNGs are central in applications such as simulations (for example the Monte Carlo method which relies on repeated random sampling to obtain numerical results), video games (with gambling and procedural generation), as well as cryptography.

A PRNG that is suitable for cryptographic applications is;

Cryptographically-secure PRNG (CSPRNG).

A requirement for a CSPRNG is that someone knowing the seed has only a negligible advantage in distinguishing the generator's output sequence from a true random sequence. PRNGs are only required to pass certain statistical tests to be certified, but a CSPRNG in general, requires years of review before an algorithm can be certified as a CSPRNG.

The German Federal Office for Information Security has established four criteria for quality of RNGs:

K1 – There should be a high probability that generated sequences of random numbers are different from each other.

K2 – A sequence of numbers is indistinguishable from "truly random" numbers according to specified statistical tests. The tests are the monobit test (equal numbers of ones and zeros in the sequence), poker test (a special instance of the chi-squared test, which is used to examine whether two categorical variables are independent in influencing the test statistic), runs test (counts the frequency of runs of various lengths), and the autocorrelation test.

K3 – It should be impossible for an attacker (for all practical purposes) to calculate, or otherwise guess, from any given subsequence, any previous or future values in the sequence, nor any inner state of the generator.

K4 – It should be impossible, for all practical purposes, for an attacker to calculate, or guess from an inner state of the generator, any previous numbers in the sequence or any previous inner generator states.

For cryptographic applications, only generators meeting the K3 or K4 standards are acceptable.

I would love to show one and explain how it works, but unfortunately it is patented and/or classified information which will get you jailed or executed for revealing, since knowing the inner workings of such an algorithm invalidates its value as a security asset.

True random number generator (TRNG)

Are devices or algorithms that generate random numbers that are statistically equivalent to a uniformly distributed data stream, TRNGs rely on unpredictable physical processes or phenomena to generate randomness.

TRNGs can be implemented using hardware or software. Hardware random number generators (HRNGs) utilize physical sources of randomness, such as atmospheric noise, radioactive decay, or electronic noise, to generate random numbers. These physical sources provide a truly random and unpredictable basis for generating numbers. On the other hand, software-based TRNGs use algorithms that exploit unpredictable events, such as mouse movements or keyboard timings, to generate random numbers.



Regardless of which physical phenomenon is used, the process of generating true random numbers involves identifying little, unpredictable changes in the data. For example, service HotBits used little variations in the delay between occurrences of radioactive decay, and RANDOM.ORG uses little variations in the amplitude of atmospheric noise.

TRNGs are generally rather inefficient compared to PRNGs, taking considerably longer time to produce numbers. They are also nondeterministic, meaning that a given sequence of numbers cannot be reproduced, although the same sequence may of course occur several times by chance. TRNGs have no period.

When it comes down to it, I think the most meaningful definition of randomness is that which cannot be predicted by humans. Whether randomness originates from unpredictable weather systems, or subatomic particle events, algorithms which are incomprehensible for the human mind it largely has the same effect. While quantum random number generators can certainly generate true random numbers, it seems to me that for all intents and purposes, all approaches are equivalent.

Sources:

<https://www.random.org/randomness/>

<https://en.wikipedia.org/wiki/Randomness>

<https://www.techtarget.com/whatis/definition/random-numbers>

<https://en.wikipedia.org/wiki/Pseudorandom_number_generator>

<https://en.wikipedia.org/wiki/Applications_of_randomness>

<https://en.wikipedia.org/wiki/Chi-squared_test>