Random Number Generation

Subjects: Mathematics, Interdisciplinary Applications | Mathematics, Applied | Computer Science, Theory & Methods

Contributor: Nikolaos Athanasios Anagnostopoulos , Nico Mexis , Tolga Arul

Ever since the antiquity, random number generation has played an important role both in common everyday life activities, such as leisure games, as well as in the advancement of science. Such means as dice and coins have been employed since the ancient times in order to generate random numbers that were used for gambling, dispute resolution, leisure games, and perhaps even fortune-telling. The theory behind the generation of random numbers, as well as the ability to potentially predict the outcome of this process, has been heavily studied and exploited by mathematics, in an attempt to either ensure the randomness of the process, to gain an advantage in correctly predicting its future outcomes, or to approximate the results of rather complicated computations. Especially in cryptography, random numbers are used due to the aforementioned properties, so that attackers have no other option but to guess the secret. This fact, in conjunction with the ongoing digitalisation of our world, has led to an interest in random number generation within the framework of computer science. In this context, random number generation systems are classified into two main categories: pseudorandom number generators and true random number generators, with the former generating sequences of numbers that appear to be random, but are in fact completely predictable when the initial value (being referred to as the *seed*) and conditions used for the number generation process are known, and with the latter generating truly random sequences of numbers that can only be predicted (correctly) with negligible probability, even if the initial value and conditions are known.

random numbers | pseudorandom number generators | true random number generators

1. Introduction

The term "random number generation" refers to the production of numbers (in a very broad sense) in such a manner that there is, ideally, no way in which the outcome of this production process can be predicted. Ever since the antiquity, random number generation has played an important role both in common everyday life activities, such as leisure games, as well as in the advancement of science. One of the oldest ways in which humans generated random numbers has been through the use of dice. It does not seem to be known when dice were invented, but they have been employed since ancient times, alongside with coin flipping, for predicting the future, decision-making, fortune-telling, gambling, dispute resolution, and leisure games. However, coin tosses are known to have a certain bias, which has been studied extensively[1]. In addition, coins could even rarely land on the edge, rendering the result useless[2]. Modern usages of random numbers include Monte Carlo experiments, game decisions, and even Cryptography (see also Cryptographically-Secure Pseudorandom Number Generator).

Random Number Generators, which are usually machines, hardware, and/or software that automate the generation of random numbers, are often abbreviated and referred to as RNGs. To this end, the ancient method of coin tossing to produce a, most usually, binary outcome, can essentially be viewed as a forerunner of modern binary RNGs, whereas dice throwing can be considered as a forerunner of high-entropy RNGs that allow for an even higher number of outcomes. It is also important to note here that in the context of random number generation, the term "number" does not exclusively refer to a mathematical number, but rather to the actual outcome of the hard (and, ideally, impossible) to predict process that is referred to as "random number generation", in the sense that a coin toss or a die throw would still be considered as random number generation, especially if the outcome of these processes was unbiased, even if the sides of the relevant coin(s) and di(c)e did not depict any numbers.

2. Desirable Properties of Random Number Generators

An RNG should have four desirable properties:

1. *Uniformity*: Each individual output must be equally probable. No biases towards any of the outcomes should exist.
2. *Independence*: The occurrence of one output must be independent of the occurrence of other outputs.
3. *Long Period*: If the RNG becomes deterministic after some time, the period should be as long as possible. Ideally, the period should be infinite, i.e., the RNG process should not be deterministic, and thus predictable, at any point in time.
4. *Practicability*: Numbers should be generated efficiently and the RNG should be supported on different architectures, systems and environments.

The coin toss mentioned above does not fully exhibit these characteristics, but can still provide sufficiently random numbers for everyday use.

3. Types of Random Number Generators

The two main types of RNGs are called True RNGs (TRNGs) and Pseudo-RNGs (PRNGs). These two types of RNGs can, usually, only be distinguished through the study of a high amount of the numbers that they produce as outcomes.

3.1. True Random Number Generators

A TRNG is able to generate random numbers that can only be predicted (correctly) with negligible probability, even if the initial value and conditions are known. TRNGs are typically slower than PRNGs and may additionally be biased. For debiasing, most often von Neumann correction is deployed[3]. It is worth mentioning that most quantum effects appear to be truly unpredictable and random, leading to TRNGs that are based on quantum mechanics.

3.2. Pseudorandom Number Generators

A PRNG can generate sequences of numbers that appear to be random, but are in fact completely predictable when the initial value (being referred to as the *seed*) and conditions used for the number generation process are known. PRNGs are usually algorithms or simple mathematical formulae, making them faster than TRNGs at the cost of determinism.

On a more general note, the concepts of determinism and randomness are rather hard to fully define. In particular, most processes that appear to be random are rather dependent on the unpredictability of their initial conditions and of the overall physical system. For example, the movement of a leaf in the wind should be fully predictable, if all the relevant parameters are known with extremely high detail, yet in practice, it is rather unpredictable in the general case. In the same fashion, the output of PRNGs may sometimes successfully pass even the most rigorous statistical tests, even though it is completely deterministic. For example, a coin toss and a die throw are also rather fully deterministic processes, at least in the general case, which, however, are most often considered as random. To this end, one should refer to chaos theory, and the way in which minor variations affect the overall time progression of a system in such a way that two or more entirely different outcomes are possible due to them. A similar example from the digital world would be a hash function that exhibits the avalanche effect: for slightly different inputs, it produces totally different outputs. If such a hash function is employed for the production of a series of numbers, by feeding it its output to produce a new number, the outcome may appear as random, especially if this function produces really long numbers, but it will always be fully deterministic and predictable.

A concept somehow similar to that of a PRNG is that of a Physical Unclonable Function, where minor variations lead the system to reach a rather stable state, which appears as unpredictable, and thus "unclonable". In this case, every time the physical system is queried, the result is almost the same; thus, a PUF produces an output that can be considered as similar to that of a PRNG that is not biased towards a particular digit (a particular outcome that would occur almost every time), but is extremely biased towards a particular series of digits (a series of outcomes that occurs almost every time).

4. Generation Methods

This section lists some commonly used and newly proposed RNGs.

4.1. TRNGs

- Radioactive decay: The fact that it is impossible to predict when an atom will decay can be used to generate random numbers[4].
- Radio Frequency (RF) noise: FM broadcasting and other means of wireless communication are susceptible to noise, which can be extracted to obtain random numbers[5].
- Thermal noise: Electronic components such as resistors emit thermal noise, which can be measured to extract random numbers[6] (e.g., RdRand).
- Memory timings: The write and read latency of memories is usually prone to noise which can be extracted to generate random numbers[7].

- Ring oscillators: Extracting the jitter of multiple free-running oscillators by comparing them can lead to random numbers[8].

4.2. PRNGs

- Middle-Square method: To generate a new random number, the previous one is squared and the middle digits of the product form the next random number[3].
- Congruential generators: The next random number is generated using a simple congruence relation[9][10].
- Multiply-with-carry: Special case of a congruential generator and Lehmer generator[11].
- Mersenne twister: Uses an algorithm to generate 624 random 32-bit numbers in a single iteration[12].
- Blum-Blum-Shub generator: Uses quadratic residues in a semiprime quotient ring to generate the next random number[13].
- Block ciphers (ANSI X9.17): Appendix C of the standard includes a random generator using a block cipher[14].

5. Attacks

Most RNGs are prone to some kind of attack. Attacks on PRNGs include, but are not limited to[15]:

- *Direct Cryptanalytic Attacks*: An attacker can directly distinguish between the output of the PRNG and random numbers (cryptanalyze the PRNG).
- *Input-based Attacks*: An attacker is able to use knowledge and control of the PRNG inputs to cryptanalyze it.
- *State Compromise Extension Attacks*: The attacker can guess state information due to an earlier breach of security.

6. Modern Applications

- Random noise (e.g., Gaussian noise)
- Key generation (TLS, SSH)
- Salts for Hashes
- RSA prime factors
- Nonces (TLS)
- Initialization Vectors (AES/Rijndael)

References

1. Diaconis, Persi; Holmes, Susan; Montgomery, Richard Dynamical Bias in the Coin Toss. *SIAM Review* **2007**, *49*, 211-235, 10.1137/S0036144504446436.
2. Murray, Daniel B.; Teare, Scott W. Probability of a tossed coin landing on edge. *Phys. Rev. E* **1993**, *48*, 2547-2552, 10.1103/PhysRevE.48.2547.

3. Von Neumann, John Various Techniques Used in Connection With Random Digits. *National Bureau of Standards: Applied Mathematics Series* **1951**, *12*, 36-38.
4. Helmut Schmidt Quantum-Mechanical Random-Number Generator. *Journal of Applied Physics* **1970**, *41*, 462-468, 10.1063/1.1658698.
5. Lee, Kyungroul; Lee, Manhee True Random Number Generator (TRNG) Utilizing FM Radio Signals for Mobile and Embedded Devices in Multi-Access Edge Computing. *Sensors* **2019**, *19*, 4130, 10.3390/s19194130.
6. Huang Zhun and Chen Hongyi, "A truly random number generator based on thermal noise," ASICON 2001. 2001 4th International Conference on ASIC Proceedings (Cat. No.01TH8549), Shanghai, China, IEEE, 2001, pp. 862-864. DOI: 10.1109/ICASIC.2001.982700.
7. Chakraborty, Supriya; Garg, Abhilash; Suri, Manan True Random Number Generation From Commodity NVM Chips. *IEEE Transactions on Electron Devices* **2020**, *67*, 888-894, 10.1109/TED.2019.2963203.
8. Markettos, A.T., Moore, S.W. (2009). The Frequency Injection Attack on Ring-Oscillator-Based True Random Number Generators. In: Clavier, C., Gaj, K. (eds) Cryptographic Hardware and Embedded Systems - CHES 2009. CHES 2009. Lecture Notes in Computer Science, vol 5747. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-04138-9_23
9. Lehmer, D. H. Mathematical methods in large-scale computing units. *Annals of the Computation Laboratory of Harvard University* **1949**, *26*, 141-146.
10. Thomson, W. E. A Modified Congruence Method of Generating Pseudo-random Numbers. *The Computer Journal* **1958**, *1*, 83, 10.1093/comjnl/1.2.83.
11. Marsaglia, George Random Number Generators. *Journal of Modern Applied Statistical Methods* **2003**, *2*, 2-13, 10.22237/jmasm/1051747320.
12. Matsumoto, Makoto; Nishimura, Takuji Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.* **1998**, *8*, 3-30, 10.1145/272991.272995.
13. Blum, Lenore; Blum, Manuel; Shub, Michael A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing* **1986**, *15*, 364-383, 10.1137/0215025.
14. ANSI X 9.17 (Revised), "American National Standard for Financial Institution Key Management (Wholesale)," American Bankers Association, 1985
15. Kelsey, J., Schneier, B., Wagner, D., Hall, C. (1998). Cryptanalytic Attacks on Pseudorandom Number Generators. In: Vaudenay, S. (eds) Fast Software Encryption. FSE 1998. Lecture Notes in Computer Science, vol 1372. Springer, Berlin, Heidelberg. DOI: 10.1007/3-540-69710-1_12