American Journal of Applied Science and Technology

VOLUME Vol.05 Issue 05 2025
PAGE NO. 70-73
DOI 10.37547/ajast/Volume05Issue05-16

# Random and Pseudo-Random Number Generation Methods

Karimov Madjit Malikovich

Agency for Assessment of knowledge and competences under the ministry of Higher Education, Science and Innovation of the Republic of Uzbekistan, Tashkent, Uzbekistan

Komil Tashev

Department of Cryptology, Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Tashkent, Uzbekistan

Nuriddin Safoev

Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Tashkent, Uzbekistan

Tashmatova Shaxnoza Sabirovna

Tashkent State Technical University named after Islam Karimov, Tashkent, Uzbekistan

Qurbonova Kabira Erkinovna

Tashkent State Technical University named after Islam Karimov, Tashkent, Uzbekistan

Fayziraxmonov Boburjon Baxtiyorjon oʻgʻli

Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Tashkent, Uzbekistan

**Received:** 23 March 2025; **Accepted:** 19 April 2025; **Published:** 21 May 2025

**Abstract:** Random number generation is a fundamental aspect of computer science, cryptography, simulations, and statistical sampling. This paper explores the definitions, classifications, and implementations of random and pseudo-random number generators (RNGs and PRNGs). We examine true random number generators (TRNGs), which derive randomness from physical phenomena, and pseudo-random number generators (PRNGs), which use deterministic algorithms to produce sequences that mimic randomness. Case studies, including Random.org, HotBits, laser-based RNGs, and the Linux random number generator, illustrate practical implementations. We also discuss vulnerabilities, security considerations, and the importance of entropy in generating unpredictable sequences.

**Keywords:** Random number generation, pseudo-random number generation, entropy, cryptography, simulations.

## Introduction:

Randomness plays a foundational role in modern computing, especially in domains such as cryptography, simulations, data sampling, and statistical modeling. A sequence of numbers is said to be random if its values are both uniformly distributed and statistically independent of previous values (Marsaglia, 2005). In the context of cryptography, the unpredictability of such sequences directly correlates with the strength of security protocols.

However, generating true randomness within deterministic digital systems is a non-trivial task. As a result, computer systems rely on two main categories of random number generation:

1. True Random Number Generators (TRNGs): These depend on inherently unpredictable physical phenomena—such as thermal noise, radioactive decay, or quantum fluctuations—to produce non-deterministic outputs. TRNGs are typically slower but offer high entropy and unpredictability, making them suitable for critical cryptographic operations.
2. Pseudo-Random Number Generators (PRNGs): These use deterministic algorithms to produce long sequences of numbers that appear random, starting from an initial seed value. While faster and more reproducible than TRNGs, PRNGs can be vulnerable to attacks if the seed or algorithm is compromised.

Modern operating systems implement secure RNG subsystems by blending both approaches: they harvest entropy from physical events (as in TRNGs) and feed it into cryptographically secure PRNGs, often compliant with standards like NIST SP 800-90A. This hybrid architecture allows for scalable, secure random number generation for system processes and user applications.

This paper provides a comparative analysis of random number generation architectures across three major operating systems—Windows, Linux, and macOS/iOS. It explores the design principles, entropy sources, API interfaces, cryptographic algorithms used, and implications for security and performance in each case.

## Defining Randomness

In both theoretical and applied computer science, randomness refers to the unpredictability and uniformity of outcomes in a data sequence. A truly random sequence must adhere to two foundational statistical properties:

1. Uniform Distribution – Each possible value in the output space must have an equal probability of occurring. This ensures that no single value or range of values is favored over others, thereby eliminating bias in the output.
2. Independence – The occurrence of one value must not influence or provide information about subsequent values. Each generated output should be statistically independent of the previous and next values in the sequence (Kenny, 2005).

A classic real-world analogy is the roll of a fair six-sided die: every number from 1 to 6 should appear with equal probability (uniformity), and the result of one roll should not affect or predict the result of the next (independence).

In computational systems, maintaining these properties is essential, especially in cryptographic contexts where any deviation from randomness can lead to patterns that adversaries may exploit. Thus, rigorous mathematical and empirical tests are used to assess the randomness of outputs produced by random number generators.

## Types of Random Number Generators

Random Number Generators (RNGs) can be broadly classified into two categories: True Random Number Generators (TRNGs) and Pseudo-Random Number Generators (PRNGs). Each class serves specific needs and presents distinct trade-offs between randomness quality, performance, and practicality.

### ○ True Random Number Generators (TRNGs)

TRNGs derive randomness from inherently unpredictable physical phenomena, such as radioactive decay or electronic noise. Because their outputs are based on non-deterministic processes, TRNGs provide high entropy and are theoretically immune to prediction. However, TRNGs often suffer from slow output rates, hardware dependencies, and implementation complexities.

- **Random.org**: This online service utilizes atmospheric noise as an entropy source to generate random numbers. It is validated by third-party randomness testing and is suitable for applications requiring high-quality randomness (Haahr, 2011). However, since it delivers output over a network, it is not recommended for cryptographic applications due to the risk of transmission interception.
- **HotBits**: Developed by John Walker, HotBits uses radioactive decay—a quantum process—as its entropy source. While this method guarantees true randomness, its generation speed is very limited, producing only about 100 bytes per second (HotBits, 2012), making it unsuitable for high-throughput applications.
- **Laser-Based RNGs**: These devices exploit chaotic fluctuations in laser intensity to generate entropy at extremely high speeds, often exceeding 10 Gbps. However, the raw output requires sophisticated post-processing to eliminate bias and ensure uniform distribution. They are effective but expensive and complex to deploy (Li, Wang, & Zhang, 2010).
- **Oscillator-Based RNGs**: One of the most commonly used hardware-based approaches, oscillator RNGs extract entropy from clock jitter—variations in the timing of electronic signals.

While cost-effective, such systems may be vulnerable to environmental manipulation (e.g., temperature or voltage changes) and often require robust bias correction to maintain randomness quality (Sunar et al., 2006).

- Pseudo-Random Number Generators (PRNGs)

Unlike TRNGs, PRNGs use deterministic algorithms to produce sequences of numbers that appear random. Initialized with a seed, these generators produce reproducible sequences, which is beneficial for simulations, games, and repeatable experiments. However, in security contexts, predictability can be a serious vulnerability if the seed or algorithm is compromised.

- Linear Congruential Generator (LCG): One of the oldest and simplest PRNGs, the LCG uses the formula s_(i+1)=(a·s_i+c)mod m where a, c, and i are constants. Despite its simplicity, LCGs suffer from short periods and high predictability when parameters are known, making them unsuitable for cryptographic applications (Chan, 2009).
- Lagged Fibonacci Generator: This method improves upon LCGs by incorporating earlier values in the sequence: s_(i+1)=(s_(i-p)±s_(i-q) )mod m where p>q. While offering longer periods, it remains deterministic and is thus not ideal for security-sensitive tasks (Chan, 2009).
- Feedback Shift Registers: These systems manipulate bit sequences using XOR and shift operations. A prominent example is the Mersenne Twister, which offers a very long period of 2^19937-1 and high statistical quality (Nishimura, 2000). However, it is not cryptographically secure, as its internal state can be reconstructed after observing a few hundred outputs.

**Security Considerations**

Random Number Generators (RNGs) are foundational components in cryptographic systems. However, both True Random Number Generators (TRNGs) and Pseudo-Random Number Generators (PRNGs) present specific security risks that can compromise their effectiveness if not properly addressed.

- Vulnerabilities in TRNGs

TRNGs, despite deriving entropy from physical phenomena, are susceptible to hardware-related attacks:

- Physical Attacks: Environmental factors such as temperature fluctuations or electromagnetic interference can introduce biases in oscillator-based TRNGs, reducing their unpredictability.
- Wear and Tear: Long-term hardware degradation can affect the consistency and quality of entropy generation, leading to a decline in randomness over time.

- **Vulnerabilities in PRNGs**

PRNGs, being algorithmically generated, rely heavily on secure initialization and design:

- Seed Predictability: If the initial seed is predictable or poorly generated, attackers can reconstruct the PRNG's output sequence, leading to complete compromise of systems relying on it.
- Backdoors: A notable example is the NSA-influenced Dual_EC_DRBG, which was suspected to include a cryptographic backdoor due to its mathematical structure (Schneier, 2007). This case underscores the importance of transparency and peer review in cryptographic standards.

- **Forward and Backward Security**

Modern secure RNGs aim to preserve confidentiality even in the event of partial compromise:

- Forward Security: Ensures that past outputs remain confidential even if the current internal state is exposed.
- Backward Security: Guarantees that future outputs cannot be predicted based on knowledge of previous internal states or outputs.

## CONCLUSIONS

Random number generation is essential in computing, with TRNGs and PRNGs serving different needs. TRNGs offer true unpredictability but are slow and hardware-dependent. PRNGs are fast and reproducible but require secure initialization. Future research should focus on improving entropy sources and hybrid models for better security and efficiency.

## REFERENCES

Chan, W. K. (2009). Random Number Generation in Simulation.

Gutterman, Z., Pinkas, B., & Reinman, T. (2006). Analysis of the Linux Random Number Generator.

Haahr, M. (2011). Introduction to Randomness and Random Numbers.

Marsaglia, G. (2005). Random Number Generators.

Schneier, B. (2007). Dual_EC_DRBG: A Case Study in Backdoors.

Sunar, B., Martin, W., & Stinson, D. (2006). A Provably Secure True Random Number Generator.

Barker, E., & Kelsey, J. (2015). Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised). NIST Special

Publication 800-90A Rev. 1. https://doi.org/10.6028/NIST.SP.800-90Ar1

Eastlake, D., Schiller, J., & Crocker, S. (2005). Randomness Requirements for Security. RFC 4086. https://www.rfc-editor.org/rfc/rfc4086

Microsoft. (2023). Cryptography API: Next Generation. Microsoft Docs. https://learn.microsoft.com/en-us/windows/win32/seccng/cng-portal

Microsoft. (2023). BCryptGenRandom function (bcrypt.h). Microsoft Docs. https://learn.microsoft.com/en-us/windows/win32/api/bcrypt/nf-bcrypt-bcryptgenrandom

Linux Kernel Documentation. (2023). Random Number Generator. https://www.kernel.org/doc/html/latest/admin-guide/dev-random.html

Linux man-pages project. (2023). getrandom(2) – Linux manual page. https://man7.org/linux/man-pages/man2/getrandom.2.html

Apple Developer Documentation. (2023). SecRandomCopyBytes. https://developer.apple.com/documentation/security/1399291-secrandomcopybytes

Apple. (2020). Platform Security Guide. https://support.apple.com/guide/security/welcome/web

Gutterman, Z., Pinkas, B., & Reinman, T. (2006). Analysis of the Linux Random Number Generator. IEEE Symposium on Security and Privacy. https://doi.org/10.1109/SP.2006.26

Dorrendorf, L., Gutterman, Z., & Pinkas, B. (2007). Cryptanalysis of the Random Number Generator of the Windows Operating System. ACM CCS. https://doi.org/10.1145/1315245.1315274

Lacharme, P. (2012). Security flaws in Linux's /dev/random. https://eprint.iacr.org/2012/251

BSD Unix. (2022). arc4random and related APIs. https://man.openbsd.org/arc4random

Kelsey, J., Schneier, B., Ferguson, N. (1999). Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator. https://www.schneier.com/paper-yarrow.pdf

Dodis, Y., et al. (2013). Security Analysis of Pseudorandom Number Generators with Input: /dev/random is not Robust. ACM CCS. https://doi.org/10.1145/2508859.2516661

Intel Corporation. (2014). Intel® Digital Random Number Generator (DRNG) Software Implementation Guide. https://www.intel.com/content/www/us/en/content-details/671488/intel-digital-random-number-generator-drng-software-implementation-guide.html

National Institute of Standards and Technology. (2012). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST SP 800-22 Rev. 1a. https://doi.org/10.6028/NIST.SP.800-22r1a

Müller, T. (2013). Security of the OpenSSL PRNG. International Journal of Information Security, 12(4), 251–265. https://doi.org/10.1007/s10207-013-0213-7

Debian Security Advisory. (2008). Debian OpenSSL Predictable PRNG Vulnerability (DSA-1571). https://www.debian.org/security/2008/dsa-1571