INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE IN ENGLISH LANGUAGE EDUCATION

PUBLISHED DATE: - 10-12-205

PAGES: - 08-09

CONFERENCE ARTICLE

RANDOM NUMBER GENERATION IN MODERN COMPUTING

Nuriddin Safoev

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

ABSTRACT

Random number generation plays a crucial role in modern computing, especially in cryptography, simulations, statistical analysis, and secure communication. This paper reviews two major approaches to generating randomness: true random number generators (TRNGs), which rely on unpredictable physical phenomena, and pseudo-random number generators (PRNGs), which produce deterministic sequences through mathematical algorithms. While TRNGs offer high-quality entropy required for security-critical tasks, PRNGs provide speed and reproducibility for general-purpose applications. By examining their mechanisms, strengths, and limitations, this study highlights the importance of choosing appropriate RNG models for secure system design.

Keywords: Random number generation, TRNG, PRNG, entropy, cryptography.

INTRODUCTION

Randomness is fundamental to a wide range of computational processes—from encryption keys and nonces in cryptography to statistical sampling and simulation modeling. A sequence is considered random when its elements are uniformly distributed and statistically independent, meaning no value influences another.

Despite its importance, achieving randomness within deterministic digital systems is inherently challenging. As a result, operating systems and applications rely on two general categories of random number generators:

- **True Random Number Generators (TRNGs)**, which collect entropy from unpredictable physical events such as thermal noise or quantum fluctuations;
- **Pseudo-Random Number Generators (PRNGs)**, which produce algorithmic sequences derived from an initial seed.

Modern systems often combine these approaches, using environmental entropy to seed cryptographically secure PRNGs (e.g., Hash_DRBG, HMAC_DRBG, CTR_DRBG). This hybrid model balances speed and security.

This paper compares RNG mechanisms implemented in mainstream platforms such as Windows, Linux, and macOS/iOS, focusing on entropy collection, cryptographic components, APIs, and related security properties.

In computing, randomness refers to both the unpredictability and uniform distribution of outputs. Two properties are essential:

1. Uniformity – all possible outcomes should occur with equal probability, preventing bias.

2. Independence – the result of one output must not reveal information about others.

Bias or correlation in generated sequences can compromise simulations or, more critically, expose cryptographic systems to attacks. Therefore, randomness is evaluated using well-established test suites such as NIST SP 800-22, Diehard/Dieharder, and TestU01, which assess statistical quality across various metrics.

2. Types of Random Number Generators

2.1 True Random Number Generators (TRNGs)

TRNGs derive randomness from inherently unpredictable physical processes, producing non-deterministic output. Though their entropy quality is superior, they typically require specialized hardware and may operate at slower speeds. Common examples include:

- Random.org – uses atmospheric noise as an entropy source; suitable for general tasks but not for cryptography due to network transmission risks.
- HotBits – relies on radioactive decay; reliable but significantly limited in throughput.
- Laser-based RNGs – exploit chaos in light oscillations, achieving extremely high speeds suitable for advanced cryptographic applications.
- Oscillator-based RNGs – utilize clock jitter and are commonly integrated into TPMs and HSMs.

Despite their strength, TRNGs may suffer from environmental interference or hardware degradation, requiring careful calibration and post-processing.

2.2 Pseudo-Random Number Generators (PRNGs)

PRNGs generate sequences using deterministic mathematical rules. Their output is reproducible when the same seed is used, which is beneficial for simulations but risky for security if the seed becomes known. Examples include:

- Linear Congruential Generator (LCG) – simple but predictable, unsuitable for cryptography.
- Lagged Fibonacci Generators – improved period length but still deterministic.
- Mersenne Twister – high statistical quality with a massive period, though not secure against state-recovery

attacks.

Table 1: Comparison of TRNGs and PRNGs

Feature	TRNG	PRNG
Source	Physical phenomena	Algorithmic, seed-based
Speed	Low	High
Determinism	None	Fully deterministic
Security risk	Hardware/environment attacks	Seed leakage, algorithm analysis
Use cases	Cryptography	Simulations, general computing

4. Security Considerations

RNGs are vital to the security of cryptographic systems. TRNGs provide high-entropy randomness but may be vulnerable to environmental manipulation or hardware failures. PRNGs depend heavily on seed secrecy—once exposed, all generated values become predictable. Historical issues such as the suspected backdoor in Dual_EC_DRBG underscore the need for transparent, peer-reviewed RNG designs. Security mechanisms such as regular reseeding, forward/backward secrecy, and hardware tamper-protection are essential for maintaining robust randomness.

Table 2. Comparison of Security Aspects of TRNGs and PRNGs

Aspect	TRNG	PRNG
Predictability	Very low	High if seed compromised
Environment dependence	High	Minimal
Hardware needs	Specialized	General-purpose
Cryptographic suitability	Excellent	Only with secure PRNG design
Attack vectors	Bias, hardware faults	Seed guessing, algorithm weaknesses

5. Conclusions

Random number generation is fundamental to secure and reliable computing. TRNGs offer genuine unpredictability needed for high-security scenarios, whereas PRNGs provide the performance and reproducibility required for large-scale and non-security tasks. Due to complementary strengths and weaknesses, hybrid RNG architectures—combining physical entropy with cryptographically secure algorithms—represent the most practical solution for modern systems. Future research must focus on improving entropy quality, protecting RNG subsystems against emerging attacks, and developing more resilient cryptographic PRNG frameworks.

References

1. Chan, W. K. (2009). Random Number Generation in Simulation.
2. Gutterman, Z., Pinkas, B., & Reinman, T. (2006). Analysis of the Linux Random Number Generator.
3. Haahr, M. (2011). Introduction to Randomness and Random Numbers.
4. Marsaglia, G. (2005). Random Number Generators.
5. Schneier, B. (2007). Dual_EC_DRBG: A Case Study in Backdoors.
6. Sunar, B., Martin, W., & Stinson, D. (2006). A Provably Secure True Random Number Generator.
7. 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
8. Eastlake, D., Schiller, J., & Crocker, S. (2005). Randomness Requirements for Security. RFC 4086. https://www.rfc-editor.org/rfc/rfc4086