ANALYZING TRUE AND PSEUDO-RANDOM NUMBER GENERATORS: IMPLICATIONS FOR CYBERSECURITY AND CYBER THREAT RESISTANCE

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

Abstract. *Random number generation is a critical building block in computer science, underpinning applications in cryptography, simulations, secure communications, and statistical modeling. This paper presents a comprehensive overview of both true random number generators (TRNGs) and pseudo-random number generators (PRNGs)—two fundamentally different approaches to generating randomness. TRNGs rely on inherently unpredictable physical processes, such as thermal noise, radioactive decay, or quantum phenomena, to produce non-deterministic output. In contrast, PRNGs employ algorithmic methods that, while deterministic and reproducible, aim to simulate randomness from an initial seed value.*

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

1. Introduction

Randomness serves as a foundational element in numerous computational domains, including cryptography, simulations, data sampling, randomized algorithms, and statistical modeling. A sequence of numbers is considered random if it exhibits both **uniform distribution** and **statistical independence**—ensuring each value is equally likely and uninfluenced by prior outputs (Marsaglia, 2005).

In the realm of **cryptography**, the **unpredictability** of such sequences directly impacts the strength of security protocols. For instance, cryptographic keys, initialization vectors (IVs), and nonces must be random to prevent prediction-based attacks. However, generating **true randomness** within inherently deterministic computing environments is a non-trivial challenge. Digital systems must therefore rely on two broad categories of random number generators (RNGs):

1. **True Random Number Generators (TRNGs):** These utilize inherently unpredictable physical processes—such as electronic thermal noise, radioactive decay, or quantum vacuum fluctuations—to generate non-deterministic outputs. Though typically slower and more resource-intensive, TRNGs offer high entropy and are suitable for applications demanding strong security guarantees.
2. **Pseudo-Random Number Generators (PRNGs):** These are deterministic algorithms that generate sequences which mimic randomness, using an initial

seed as the entropy source. PRNGs are fast and efficient but susceptible to state-recovery or prediction attacks if the seed or algorithm becomes known or compromised.

To address the limitations of both approaches, **modern operating systems implement hybrid random number subsystems**. These systems gather environmental entropy (e.g., keyboard timing, mouse movements, interrupt timing, hardware noise) and use it to seed cryptographically secure PRNGs—such as those compliant with **NIST SP 800-90A** (e.g., Hash_DRBG, HMAC_DRBG, and CTR_DRBG). This layered architecture enables both high throughput and adequate security across applications and system services.

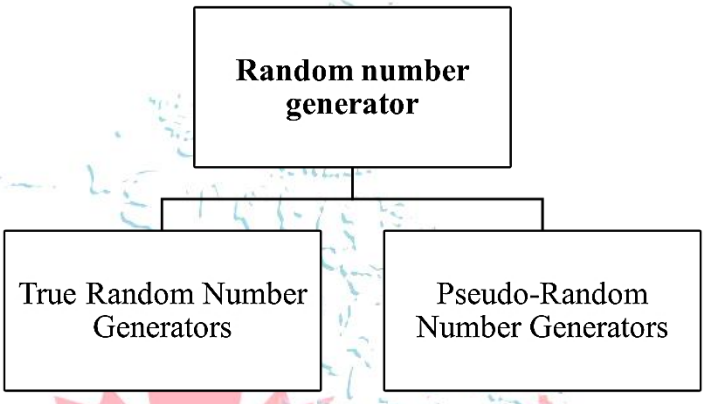

Figure 1. categories of random number generators

This paper conducts a **comparative analysis of random number generation architectures** across three major platforms—**Windows**, **Linux**, and **macOS/iOS**—by examining their entropy collection mechanisms, cryptographic components, application programming interfaces (APIs), and the security implications of their designs.

2. Defining Randomness

In theoretical computer science and practical system design, **randomness** is characterized by the **unpredictability** and **uniformity** of outcomes. A sequence is said to be truly random if it satisfies two fundamental statistical properties:

1. **Uniform Distribution**
   Each possible value in the output domain should have the same probability of occurring. This ensures the absence of bias—no value is favored over another, maintaining fairness across outcomes.
2. **Independence**
   The generation of any one value should not influence or provide information about others. This property guarantees that each output is statistically independent of the preceding and succeeding values (Kenny, 2005).

A common real-world analogy is the roll of a fair six-sided die. Ideally, every face (1–6) has a 1/6 probability of appearing, and the result of one roll does not influence the next. Any deviation from these properties introduces **bias** or **predictability**, which can be **exploited by adversaries**, especially in cryptographic applications.

In computational settings, ensuring these properties is **critical**. Even minor statistical biases or correlations in random outputs can lead to serious vulnerabilities, such as:

- Predictable session keys in secure communication
- Repeatable IVs causing ciphertext leakage
- Bias in simulations producing skewed results

As a result, RNGs are subject to **rigorous statistical evaluation**, including:

- **NIST SP 800-22**: A suite of 15 tests for binary sequences
- **Diehard/Dieharder tests**: Focused on PRNG evaluation
- **TestU01**: Comprehensive statistical analysis for RNGs

These tests assess both **short-term unpredictability** and **long-term distributional properties**, ensuring that the generated sequences meet the standards for randomness required by modern applications.

3. Types of Random Number Generators

3.1. True Random Number Generators (TRNGs)

True Random Number Generators (TRNGs) use unpredictable physical processes—such as thermal noise, radioactive decay, quantum effects, or electronic jitter—to generate random numbers. Because they are based on non-deterministic sources, TRNGs do not require a seed and produce results that are not reproducible, making them well-suited for high-security applications.

However, TRNGs come with several limitations:

- **Slower output speed** compared to algorithmic methods
- **Dependence on specialized hardware**
- **Need for complex post-processing** to remove bias and ensure uniform randomness

Notable TRNG Implementations

- **Random.org:** This service harnesses **atmospheric noise** as an entropy source, offering high-quality randomness validated by third-party statistical tests (Haahr, 2011). While suitable for non-critical tasks (e.g., gaming, lottery draws), it is not recommended for cryptographic purposes due to transmission over public networks, introducing risks such as interception and man-in-the-middle attacks.
- **HotBits:** Developed by John Walker, HotBits uses **radioactive decay**—a fundamentally quantum process—to generate randomness. It is one of the earliest public TRNGs, but its output speed is limited to approximately **100**

bytes/second (HotBits, 2012), restricting its usability for modern applications requiring high data throughput.

- **Laser-Based RNGs:** These devices utilize the **chaotic behavior of laser intensity fluctuations** to achieve high-speed entropy generation—often exceeding **10 Gbps** (Li, Wang, & Zhang, 2010). Such systems are used in quantum cryptography and high-assurance randomness sources but require **complex bias removal algorithms** and **expensive hardware**.
- **Oscillator-Based RNGs:** A practical and widely implemented approach uses **clock jitter**—tiny variations in oscillation timing—as an entropy source. Found in many hardware security modules (HSMs) and TPMs (Trusted Platform Modules), they offer a balance between cost and performance. Nevertheless, they are **vulnerable to environmental influences** (e.g., temperature, voltage manipulation) and often require cryptographic post-processing (Sunar et al., 2006).

3.2. Pseudo-Random Number Generators (PRNGs)

PRNGs use deterministic mathematical algorithms to generate sequences that simulate randomness. They are initialized with an internal seed and follow a predictable path. While efficient and reproducible—features useful in simulations, procedural generation, and software testing—PRNGs are not inherently secure, especially if:

- The seed is weak or guessable
- The algorithm's internal state becomes exposed

PRNGs are generally unsuitable for cryptographic tasks unless explicitly designed to meet cryptographic security criteria. Popular PRNG Algorithms:

- **Linear Congruential Generator (LCG)**: One of the oldest and simplest PRNGs, the LCG uses the formula $\boldsymbol{s_{i+1} = (a \cdot s_i + c) mod\ m}$ 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: $\boldsymbol{s_{i+1} = (s_{i-p} \pm s_{i-q}) mod\ m}$ where $\boldsymbol{p > q}$. 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.

Table 1: Comparison of TRNGs and PRNGs

Feature	TRNGs (True RNGs)	PRNGs (Pseudo-RNGs)
Entropy Source	Physical (e.g., quantum effects, thermal noise)	Algorithmic (seed-based computation)
Speed	Slow (e.g., ~100 B/s for HotBits)	Fast (e.g., Gbps with ChaCha algorithm)
Determinism	Non-deterministic	Deterministic
Reproducibility	Not reproducible	Reproducible with the same seed
Security Risks	Hardware attacks, environmental bias	Seed leakage, algorithm weaknesses
Applications	Cryptography, secure key generation	Simulations, games, non-critical computations
Examples	Random.org, ID Quantique, HotBits	LCG, Xoshiro256++, ChaCha

4. Security Considerations of Random Number Generators

Random Number Generators (RNGs) play a critical role in modern cryptographic systems, underpinning key generation, secure communication, authentication, and session management. They are generally divided into two categories: **True Random Number Generators (TRNGs)** and **Pseudo-Random Number Generators (PRNGs)**. Both types introduce distinct security risks that must be carefully managed to avoid undermining the overall security of a system.

TRNGs derive entropy from unpredictable physical phenomena, such as electronic noise, radioactive decay, or clock jitter. While they are theoretically non-deterministic and provide high-quality randomness, they are not immune to threats. Environmental factors like temperature fluctuations, voltage instability, or electromagnetic interference can bias their outputs. Additionally, long-term hardware degradation may affect entropy quality over time, and attackers may exploit these weaknesses in side-channel attacks.

PRNGs, on the other hand, rely on deterministic algorithms initialized with a **seed** value. If this seed is poorly chosen, predictable, or leaked, attackers can reproduce the entire sequence of outputs—effectively compromising any system relying on it. Historically, the cryptographic community has witnessed examples like **Dual_EC_DRBG**, which raised concerns over the presence of an intentional backdoor due to its mathematical design. This emphasizes the importance of transparency, peer review, and avoidance of obscure or proprietary RNG implementations in security-sensitive contexts.

To mitigate these risks, modern secure RNG systems incorporate mechanisms like **forward secrecy**—ensuring that past outputs remain secure even if the current internal state is exposed—and **backward secrecy**—preventing future outputs from being predicted using previously known data. Moreover, RNGs must be regularly reseeded with fresh, high-entropy input and protected from tampering at both hardware and software levels.

The following Table 2 summarizes and contrasts the security-related characteristics of TRNGs and PRNGs:

Table 2. Comparison of Security Aspects of TRNGs and PRNGs

Feature / Risk	**TRNG (True RNG)**	**PRNG (Pseudo-RNG)**
Entropy Source	Physical phenomena (e.g., noise, decay, jitter)	Deterministic algorithm with a seed
Predictability	Very low (but sensitive to environmental factors)	High if seed is known or poorly chosen
Environmental Dependence	High (can be influenced by temperature, EMI, etc.)	Low
Output Speed	Slow (often < 1 MB/s)	Fast (up to GB/s range)
Forward / Backward Secrecy	Requires explicit mechanisms	Depends on algorithm and implementation
Hardware Dependency	Requires dedicated hardware	Software-based, runs on general-purpose hardware
Cryptographic Suitability	Ideal for generating keys and nonces	Suitable with secure design and good seed management
Potential Attack Vectors	Hardware failure, environmental interference	Seed guessing, algorithm compromise or backdoors

5. Conclusions

Random number generation is a cornerstone of modern computing and cryptography. True Random Number Generators (TRNGs) provide genuine unpredictability by harnessing physical phenomena, making them ideal for high-security applications—albeit at the cost of slower speeds and hardware complexity. In contrast, Pseudo-Random Number Generators (PRNGs) offer efficiency and reproducibility, but their security hinges on robust algorithm design and secure seeding. As technology and threats evolve, ongoing research should aim to enhance entropy sources, strengthen resistance to attacks, and develop hybrid RNG models that combine the strengths of both TRNGs and PRNGs to achieve greater security, scalability, and practicality.

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
9. Microsoft. (2023). Cryptography API: Next Generation. Microsoft Docs. https://learn.microsoft.com/en-us/windows/win32/seccng/cng-portal
10. Microsoft. (2023). BCryptGenRandom function (bcrypt.h). Microsoft Docs. https://learn.microsoft.com/en-us/windows/win32/api/bcrypt/nf-bcrypt-bcryptgenrandom
11. Linux Kernel Documentation. (2023). Random Number Generator. https://www.kernel.org/doc/html/latest/admin-guide/dev-random.html
12. Linux man-pages project. (2023). getrandom(2) – Linux manual page. https://man7.org/linux/man-pages/man2/getrandom.2.html
13. Apple Developer Documentation. (2023). SecRandomCopyBytes. https://developer.apple.com/documentation/security/1399291-secrandomcopybytes
14. Apple. (2020). Platform Security Guide. https://support.apple.com/guide/security/welcome/web
15. 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
16. 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
17. Lacharme, P. (2012). Security flaws in Linux's /dev/random. https://eprint.iacr.org/2012/251
18. BSD Unix. (2022). arc4random and related APIs. https://man.openbsd.org/arc4random

19. 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
20. 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
21. 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
22. 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
23. 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
24. Debian Security Advisory. (2008). Debian OpenSSL Predictable PRNG Vulnerability (DSA-1571). https://www.debian.org/security/2008/dsa-1571