International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Modern Pseudo-Random Number Generator Algorithms

Dr. Yatu Rani[1], Laxman Mukhiya[2]

[1,2]Artificial Intelligence and Data Science, Dr. Akhilesh Das Gupta Institute of Professional Studies Delhi, India

ABSTRACT

Pseudo-Random Number Generators (PRNGs) are a fundamental component of modern computing, underpinning applications that range from cryptographic protocols and scientific simulations to machine learning and gaming engines. As computing demands evolve, the need for PRNGs that can simultaneously offer long periods, statistical uniformity, computational efficiency, and security has become increasingly critical. Older algorithms such as Linear Congruential Generators (LCGs) are no longer considered adequate for most practical purposes due to their short periods and predictable output sequences. This paper presents a structured review and comparison of four widely used modern PRNG algorithms: Mersenne Twister (MT19937), Permuted Congruential Generator (PCG), the Xoshiro/Xoroshiro family, and Blum Blum Shub (BBS). Each algorithm is evaluated across key parameters including period length, speed, statistical quality, and security characteristics. The goal is to provide students and practitioners with a clear and practical understanding of when and why a particular PRNG should be chosen for a given application.

Keywords: Pseudo-Random Number Generator, Mersenne Twister, PCG, Xoshiro, Blum Blum Shub, Cryptographic Security, Statistical Randomness

I. INTRODUCTION

Randomness is a surprisingly difficult concept to implement on deterministic machines like computers. A Pseudo-Random Number Generator (PRNG) is an algorithm that produces a sequence of numbers which approximates the properties of truly random sequences — but is entirely determined by an initial value called the seed. Despite being deterministic, well-designed PRNGs generate output that is statistically indistinguishable from true randomness in most practical tests.

The role of PRNGs spans virtually every technical domain. In cryptography, they are used for key generation, nonce creation, and padding schemes — where any predictability can have catastrophic consequences. In scientific simulations such as Monte Carlo methods, PRNGs must produce statistically uniform distributions over enormous sample sizes. In machine learning, they govern weight initialization, dropout regularization, and data augmentation. In gaming, they are responsible for procedural generation, loot mechanics, and physics simulations — areas where speed matters most.

Early random number generators, such as the Linear Congruential Generator (LCG) popularized in the mid-20th century, served well for many years. However, their short periods, correlation between consecutive outputs, and susceptibility to reverse-engineering made them inadequate for modern demands. This triggered decades of research

that produced significantly more capable algorithms. Among the most important modern PRNGs are the Mersenne Twister (introduced in 1998), which became the default in Python and MATLAB; the PCG family (2014), which improved on statistical quality with a smaller footprint; the Xoshiro/Xoroshiro family (2018), which prioritized raw speed; and Blum Blum Shub (1986), which remains a standard reference for cryptographically secure pseudo-random generation.

This review paper systematically compares these four algorithms, examining their internal design, performance characteristics, and suitability for real-world use cases. The objective is not to declare a universal winner — rather, to help readers understand the trade-offs involved and make informed decisions based on their application requirements.

II. LITERATURE REVIEW

The development of PRNG algorithms over the past few decades reflects a continuous effort to balance competing goals: longer periods, better statistical properties, faster execution, reduced memory usage, and in some cases, cryptographic strength. Early work in the 1950s and 1960s laid the groundwork with simple congruential methods, but it was not until the late 1990s that truly robust general-purpose PRNGs began to emerge.

Matsumoto and Nishimura [1] introduced the Mersenne Twister in 1998, which quickly became one of the most widely used PRNGs in history. Its period of 2^19937 - 1 was unprecedented, and it passed the Diehard battery of statistical tests convincingly. However, it was never intended as a cryptographic tool, and later analyses showed it could fail more demanding test suites such as BigCrush under certain conditions [2].

O'Neill's 2014 work introduced the Permuted Congruential Generator (PCG) family [3], which built upon Linear Congruential Generators by applying output permutation functions to dramatically improve statistical quality. PCG generators are much smaller in state (128 bits for PCG64) and faster than MT in most benchmarks. However, they share the same fundamental limitation of non-cryptographic PRNGs — their output can be predicted if the internal state is compromised.

Blackman and Vigna [4] proposed the Xoshiro and Xoroshiro families in 2018 as an evolution of xorshift generators, targeting the highest possible throughput for non-cryptographic applications. These generators excel in scenarios where billions of numbers are needed per second, such as real-time game physics. However, their linearity makes them unsuitable for security-sensitive contexts.

The Blum Blum Shub generator [5], while introduced back in 1986, remains relevant in academic and cryptographic discussions. Its security is based on the computational hardness of factoring large integers — the same foundation as RSA encryption. This gives it provably strong security guarantees, though its reliance on modular exponentiation makes it orders of magnitude slower than other PRNGs reviewed here.

More recent research has focused on hybrid approaches [6], combining fast generators with cryptographic post-processing, as well as the emerging potential of quantum random number generators (QRNGs) [7]. Table I summarizes the key contributions and limitations of relevant prior studies.

TABLE I: Summary of Key Related Studies

Author / Study	Focus Area	Key Contribution	Limitation
Matsumoto & Nishimura (1998)	Development of MT19937	Very long period (2^19937-1), passes	Not suitable for cryptographic use

		Diehard tests	
O'Neill (2014)	Introduction of PCG family	Better statistical output, small state, fast performance	Not cryptographically secure by default
Blackman & Vigna (2018)	Xoshiro/Xoroshiro algorithms	Extremely fast, excellent for simulations and games	Short period for some variants, weak in high dimensions
Blum, Blum & Shub (1986)	Blum Blum Shub CSPRNG	Strong cryptographic security based on factoring hardness	Very slow due to modular exponentiation; impractical for general use
L'Ecuyer (1999)	MRG and combined generators	Improved uniformity and portability across systems	More complex implementation compared to single generators
Vigna (2016)	xorshift128+ analysis	Faster alternative to MT with decent statistical properties	Fails some linear complexity tests in certain configurations

III. METHODOLOGY

This study follows a qualitative and descriptive comparison approach, drawing on published algorithm specifications, empirical benchmarks reported in the literature, and standardized test suite results. The four selected algorithms — MT19937, PCG64, Xoshiro256**, and BBS — were chosen because they collectively represent the major design paradigms in modern PRNG development: state-space twisting, permutation-based transformation, xorshift linear operations, and number-theoretic security.

Each algorithm is evaluated across the following five criteria:

- Period Length: The total number of values an algorithm can produce before repeating its sequence. Longer periods are essential for large-scale simulations and any application that generates millions or billions of random values.
- Speed / Computational Efficiency: Measured in terms of approximate throughput (numbers generated per second) and relative CPU overhead. This includes both single-threaded performance and behavior on modern hardware with SIMD instructions.
- Statistical Randomness Quality: Evaluated using well-established test batteries including the NIST Statistical Test Suite, TestU01's BigCrush, and the Diehard tests. These tests assess properties such as uniformity, independence, and absence of detectable patterns.
- Cryptographic Security: Determined by whether an algorithm satisfies the definition of a Cryptographically Secure PRNG (CSPRNG). A CSPRNG must be computationally infeasible to distinguish from true randomness and must not allow state recovery from observed output.
- Computational Complexity: The time and space complexity of generating each number, including the memory footprint of the generator state. This is particularly relevant when embedding PRNGs in resource-constrained environments.

It is important to note that this review does not conduct original experiments. Instead, it synthesizes findings from existing literature and standardized benchmarks to offer a consolidated comparison that is

accessible to readers with an undergraduate-level background in computer science.

IV. COMPARATIVE ANALYSIS OF PRNG ALGORITHMS

The four algorithms under review represent meaningfully different design philosophies. Below is a detailed discussion of each, followed by the comparative tables.

A. Mersenne Twister (MT19937)

MT19937 is arguably the most recognized PRNG in use today. It maintains a state array of 624 32-bit words and applies a series of twist and tempering operations at each step. Its period of 2^19937 - 1 makes it practically inexhaustible for most simulation workloads. It initializes well and has been extensively validated. However, it consumes 2.5 KB of memory for its state, which is significant in embedded systems. More critically, it is not suitable for any security application since its state can be fully reconstructed after observing just 624 consecutive outputs.

B. PCG (Permuted Congruential Generator)

PCG64 operates by running a 128-bit Linear Congruential Generator internally and then applying an output permutation function — specifically a rotation combined with XOR — to scramble the output. This design is clever because it retains the mathematical tractability of LCGs while eliminating their statistical weaknesses. PCG is faster than MT on most modern architectures and has a much smaller memory footprint. It also supports multiple independent streams easily, which is useful in parallel computing. Like MT, it is not cryptographically secure.

C. Xoshiro / Xoroshiro Family

The Xoshiro256** ("xor-shift-rotate") generator uses a 256-bit state and performs a fixed sequence of XOR, shift, and rotation operations. It is among the fastest PRNGs available and produces excellent statistical quality for its speed. The 256-bit state provides a very long period of 2^256 - 1. However, because all operations are linear over GF(2), the generator fails linear complexity tests and should not be used in cryptographic contexts. The Xoroshiro variant uses a 128-bit state for applications where memory is tight and the slightly shorter period is acceptable.

D. Blum Blum Shub (BBS)

BBS computes x_{n+1} = x_n^2 mod M, where M = p * q is the product of two large primes satisfying p ≡ q ≡ 3 (mod 4). Each iteration yields one or a few bits of the next output. The security of BBS is reducible to the hardness of the Quadratic Residuosity Problem, making it provably secure under well-studied computational assumptions. However, each iteration requires a modular squaring operation with a large modulus (typically 512–1024 bits), making it several orders of magnitude slower than the other algorithms in this comparison. It is primarily used where security is the only requirement — such as key generation — and throughput is not a concern.

TABLE II: Detailed Comparison of PRNG Algorithms

Algorithm	Period Length	Speed	Security	Primary Use Case	Key Limitation
Mersenne Twister (MT19937)	2^19937 − 1	Moderate	Not secure	Simulations, ML, gaming	Large state (2.5 KB); fails some modern tests
PCG (PCG64)	2^128	Very fast	Not secure	General purpose, gaming, stats	No cryptographic guarantees

Xoshiro256**	2^256 − 1	Fastest	Not secure	Games, simulations, benchmarks	Fails BigCrush in some variants
Blum Blum Shub (BBS)	Exponential	Very slow	Cryptographically secure	Key generation, cryptography	Impractical for high-speed use

Note: Speed ratings are relative. Xoshiro256** typically achieves ~1.5 billion values/second on modern hardware, MT ~600 million, PCG ~900 million, and BBS under 1 million.

TABLE III: Statistical Test Performance Summary

Algorithm	NIST Tests	Diehard	BigCrush	Dimensionality	Practical Speed
Mersenne Twister	Passes	Passes	Fails some	623-dimensional	Moderate
PCG64	Passes	Passes	Passes	High	High
Xoshiro256**	Passes	Passes	Mostly passes	High	Very High
Blum Blum Shub	Passes	Passes	Passes	Moderate	Low (slow)

Note: "Passes" indicates the algorithm meets the test suite requirements under standard configurations. Individual edge cases may produce different results.

V. APPLICATIONS OF PRNG ALGORITHMS

The choice of PRNG algorithm is not a one-size-fits-all decision — it depends heavily on the requirements of the application. Below are the primary domains where these algorithms are commonly deployed.

A. Cryptography and Security Systems

In cryptographic applications — including SSL/TLS handshakes, password generation, session token creation, and nonce generation — only CSPRNGs are acceptable. Among the four algorithms reviewed, only BBS satisfies this requirement. Most cryptographic libraries in practice use algorithms like ChaCha20 or AES-CTR in CSPRNG mode, which are faster than BBS while maintaining security. MT, PCG, and Xoshiro should never be used in security-sensitive contexts.

B. Scientific Simulations

Monte Carlo simulations, numerical integration, and stochastic modeling require generators with long periods, high uniformity, and good behavior in high-dimensional spaces. MT19937 has historically been the standard choice here, though PCG and Xoshiro256** are increasingly preferred due to better statistical results in TestU01 and higher speed. Multiple independent streams (supported natively by PCG) are particularly valuable in parallel Monte Carlo simulations.

C. Machine Learning and AI

Randomness plays a central role in machine learning — from initializing weights in neural networks to applying dropout, shuffling training data, and generating synthetic datasets. The default PRNG in NumPy (since version 1.17) is PCG64, which replaced MT19937. This shift reflects the research community's recognition that PCG provides better statistical properties with improved performance. Xoshiro variants are also used in frameworks that prioritize throughput.

D. Gaming and Procedural Generation

In video games, PRNGs are used for procedural world generation, enemy AI decision-making, loot drop systems, and physics simulations. Speed is critical in real-time environments. Xoshiro256** and PCG are the most popular choices in this domain. MT19937 is sometimes used in older game engines.

Security is not a concern here, but reproducibility and speed are paramount.

E. Scientific Computing and Statistical Packages

Languages and packages such as Python (random module), NumPy, R, and Julia have historically used MT19937 as their default PRNG. More recent versions have moved toward PCG variants, driven by its performance advantages and cleaner statistical profiles. BBS has no practical role in high-throughput scientific computing.

VI. CHALLENGES AND OPEN ISSUES

Despite significant advances in PRNG design, several fundamental challenges remain unresolved or represent ongoing trade-offs for developers and researchers.

A. Predictability and State Recovery

All non-cryptographic PRNGs, including MT, PCG, and Xoshiro, are fundamentally predictable once a sufficient number of outputs are observed. For MT19937, as few as 624 consecutive 32-bit outputs are enough to reconstruct the entire internal state. This predictability is not always obvious to developers, leading to security vulnerabilities when these generators are misused in security-sensitive applications.

B. Speed vs. Security Trade-off

There is a well-known inverse relationship between speed and cryptographic security in PRNG design. BBS is secure but slow. Xoshiro is fast but insecure. There is no algorithm among those reviewed that fully satisfies both requirements. Hybrid generators — which use a fast PRNG seeded by a CSPRNG — partially address this, but add architectural complexity.

C. Seeding and Initialization Sensitivity

Poor seeding is a common source of failures in practice. A PRNG is only as good as its seed — if the seed is predictable (e.g., based on the system time with low resolution), the entire output sequence becomes predictable. This issue affects all four algorithms equally. MT19937, for instance, produces known poor outputs from certain seed values unless a specific initialization procedure is followed.

D. True Randomness vs. Pseudo-Randomness

No matter how sophisticated the algorithm, PRNGs produce deterministic output that only approximates randomness. For applications requiring genuine non-determinism — such as high-value key generation or casino gaming systems — hardware random number generators (HRNGs) or entropy-harvesting methods (like
/dev/random on Linux) are necessary. The gap between true randomness and pseudo-randomness remains a conceptual and practical challenge.

E. Hardware and Portability Constraints

Xoshiro and PCG are optimized for 64-bit architectures. On 32-bit or embedded systems, their performance may degrade or require adaptation. BBS requires multi-precision arithmetic support, making it unsuitable for constrained environments. MT's 2.5 KB state size can also be problematic in IoT or microcontroller applications.

VII. FUTURE SCOPE

The field of random number generation is evolving in several interesting directions that are likely to reshape how PRNGs are selected and designed in the coming years.

A. Hybrid PRNG Architectures

One of the most promising directions is the development of hybrid systems that combine the speed of

statistical PRNGs with the unpredictability of CSPRNGs. For example, a fast Xoshiro generator can be periodically reseeded using entropy from a cryptographic source, providing high throughput while limiting the window of predictability. These designs are already appearing in system-level random number APIs.

B. Quantum Random Number Generators (QRNGs)

Quantum mechanics offers a fundamentally different approach to randomness — one that is truly non-deterministic rather than pseudo-random. Commercial QRNG devices are already available, and cloud-based quantum randomness APIs are accessible through services like the Australian National University's quantum server. As quantum computing matures, QRNGs could become the standard entropy source for seeding software PRNGs in high-security environments.

C. AI-Assisted Randomness and Testing

Machine learning techniques, particularly generative models and adversarial testing, are beginning to be applied to evaluate and improve PRNG quality. Adversarial attacks using neural networks can sometimes identify subtle biases in PRNG output that standard test batteries miss. Conversely, AI may also assist in designing new PRNG architectures that generalize better across diverse application requirements.

D. Standardization and Certification

As PRNGs become embedded in increasingly critical systems — from autonomous vehicles to financial algorithms

— the need for standardized certification frameworks is growing. NIST already provides guidelines for CSPRNGs [8], but general-purpose PRNGs lack equivalent formal standards. Future work in this area is likely to produce clearer certification benchmarks for different use-case categories.

VIII. CONCLUSION

This paper has reviewed and compared four representative modern PRNG algorithms — Mersenne Twister (MT19937), PCG64, Xoshiro256**, and Blum Blum Shub — across dimensions of period length, speed, statistical quality, and cryptographic security. Each algorithm reflects a distinct design philosophy and is best suited to a specific class of applications.

MT19937 remains a solid choice for legacy systems and scientific software that requires a well-tested, widely supported generator. PCG64 offers a compelling improvement for general-purpose use, combining speed, small state size, and strong statistical properties — making it the better default for new simulation and machine learning projects. Xoshiro256** is the top recommendation when raw throughput is the primary concern, such as in real-time games or benchmarking tools. Blum Blum Shub, despite its impracticality in most contexts, remains the reference standard for applications where cryptographic security is the sole criterion and performance is secondary.

No single algorithm dominates all scenarios. The appropriate choice depends on the specific requirements of the application, and developers should resist the temptation to use a single PRNG everywhere. As quantum computing and hybrid architectures mature, the landscape of random number generation is set to shift further, offering opportunities that current algorithms cannot yet fulfill.

IX. REFERENCES

1. M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," ACM Transactions on Modeling and Computer Simulation, vol. 8, no. 1, pp. 3–30, Jan. 1998.

2. P. L'Ecuyer and R. Simard, "TestU01: A C library for empirical testing of random number generators," ACM Transactions on Mathematical Software, vol. 33, no. 4, Article 22, Aug. 2007.
3. M. E. O'Neill, "PCG: A family of simple fast space-efficient statistically good algorithms for random number generation," Harvey Mudd College, Claremont, CA, Tech. Rep. HMC-CS-2014-0905, Sep. 2014.
4. D. Blackman and S. Vigna, "Scrambled linear pseudorandom number generators," ACM Transactions on Mathematical Software, vol. 47, no. 4, pp. 1–32, 2021.
5. L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudo-random number generator," SIAM Journal on Computing, vol. 15, no. 2, pp. 364–383, 1986.
6. F. Panneton, P. L'Ecuyer, and M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2," ACM Transactions on Mathematical Software, vol. 32, no. 1, pp. 1–16, 2006.
7. M. Herrero-Collantes and J. C. Garcia-Escartin, "Quantum random number generators," Reviews of Modern Physics, vol. 89, no. 1, p. 015004, Feb. 2017.
8. E. Barker and J. Kelsey, "Recommendation for random number generation using deterministic random bit generators," NIST Special Publication 800-90A, Rev. 1, National Institute of Standards and Technology, Gaithersburg, MD, Jun. 2015.
9. S. Vigna, "An experimental exploration of Marsaglia's xorshift generators, scrambled," ACM Transactions on Mathematical Software, vol. 42, no. 4, Article 30, Jun. 2016.
10. G. Marsaglia, "Xorshift RNGs," Journal of Statistical Software, vol. 8, no. 14, pp. 1–6, Jul. 2003.
11. P. L'Ecuyer, "Tables of linear congruential generators of different sizes and good lattice structure," Mathematics of Computation, vol. 68, no. 225, pp. 249–260, 1999.
12. D. E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd ed. Reading, MA: Addison-Wesley, 1997.
13. National Institute of Standards and Technology, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication 800-22, Rev. 1a, Apr. 2010.
14. R. Brent, "Uniform random number generators for supercomputers," in Proc. Fifth Australian Supercomputer Conference, Melbourne, Australia, 1992, pp. 95–104.
15. J. Salmon, M. Moraes, R. Dror, and D. Shaw, "Parallel random numbers: As easy as 1, 2, 3," in Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2011, pp. 1–12.