Grenze International Journal of Engineering and Technology, June Issue

# A Class-based Random Number Generator with Entropy Strength Classification for NIST-Compliant Cryptographic Applications

Sanjeev Kumar Sinha[1] and Harvir Singh[2]
[1-2]Bhagwant University, Ajmer, India
Email: sanjeevsinha1996 @gmail.com, drharvir@gmail.com

***Abstract*— Random number generators (RNGs) are essential to cryptography, simulations and secure digital communication. Unfortunately, most deployed RNGs have no runtime visibility into their quality of randomness. They are "black boxes" – providing numbers with no accompanying service or classification describing entropy strength. In this paper we introduce class based RNG architecture which solves these problems. Numbers are continuously classified in real time into one of four strength grades: Basic, Statistical, Cryptographic, and Military. Classification is determined by minimum p value across a rolling window of NIST SP 800 22 statistical tests. The generator uses two entropy sources (environmental noise + user timing input), a cryptographic extractor (SHA 512), and newly designed Adaptive Nonlinear Scrambler (ANS). The ANS layer breaks linear structure with data dependent feedback paths. Evaluated on 1 million bits of output, this generator scored a minimum p value of 0.82 (Cryptographic grade) on all 15 NIST tests with all tests scoring higher than p = 0.82. Our random number classifier works on real time and has demonstrated 97.1% accuracy identifying correct operating grade and auto triggered reseeding events when entropy grade weakened below Cryptographic strength. Benchmark comparison with three reference generators (Mersenne Twister, Intel RdRand, OpenSSL) revealed our RNG was the only one to pass Linear Complexity and the only one offering runtime grade labelling. One case study has been included for AES 256 key generation.**

***Index Terms*— Random number generator, NIST SP 800 22, randomness classification, entropy grading, Adaptive Nonlinear Scrambler, SHA 512.**

## I. INTRODUCTION

Randomness is crucial to the functioning of computational processes, being widespread in nature but difficult to prove in computer-based systems. One single non-random bit may render a cryptosystem useless, and even a tiny correlation may lead to failure in the Monte Carlo algorithm. Thus, the National Institute of Standards and Technology developed SP 800-22 [1], an ensemble of 15 statistical tests aimed to find any deviations from randomness in a test object. These tests have been adopted by the industry as the golden standard in cryptographic RNG testing.

Mainly, there are two issues with the current RNG certification paradigm. On the one hand, the majority of the RNGs get checked only once throughout the design process, without receiving further feedback on its quality while operating.

*Grenze ID: 01.GIJET.12.2.691_26*

This may be caused by entropy source degradation (e.g., increasing noise in a microphone), and by programming errors leading to reduced state space. The RNG will keep producing output values, yet the program will have no information whether they are actually random. On the other hand, even in case of continuous monitoring, only binary "pass/fail" reports are generated, as there is no possibility to distinguish between a weak random sequence sufficient for a casino simulation and a strong one used for military applications.
In this work, we present a new class based random number generator that solves the problem described above. The features of our generator include:

- Classification into three classes: Entropy Collector, Random Data Processor, and Data Output for entropy generation, processing, and output respectively.
- The Adaptive Nonlinear Scrambler (ANS) - a newly developed mixing function consisting of LFSR with data dependent non-linear feedback, which has successfully passed the highly demanding Linear Complexity test.
- Randomness Classification system in grades with defined thresholds of significance values (Military: $p \geq 0.9$, Cryptographic: $p \geq 0.8$, Statistical: $p \geq 0.7$, Basic: $p < 0.7$) with reseeding mechanism when the grade is less than Cryptographic.
- Comparative analysis with three well known generators (Mersenne Twister, Intel RdRand, OpenSSL).

## II. Background and Related Work

The most popular RNG evaluation suite is the NIST SP 800 22 suite [1], which contains 15 tests, each of them producing p value, and classifies sequences that pass all 15 tests as "random", i.e. $p \geq 0.01$. Diffie and Hellman [2] emphasized the importance of unpredictability of secret keys. In his paper, Gallager [3] studied properties of linear feedback shift registers, the basis of many PRNGs. Quantum RNGs [4] provide truly random numbers but require expensive special hardware. Various hybrid solutions, including the DNA BBS Generator created by Gaba et al. [5], combine biological sequences with number theory approaches.

### *A. Definition of Randomness and NIST SP 800 22 Tests*

"Truly random" binary sequences represent the ideal case. In practice, however, we have to use some tests in order to detect the presence of any deviation from the "ideal randomness". As previously mentioned, the set of NIST SP 800 22 tests [1] comprises 15 tests, each one checking specific deviations from the uniform distribution. The list of these tests includes the following:

- Monobit test: checks if the number of 0 and 1 bits is approximately equal;
- Runs test: examines the distribution of lengths of consecutive identical bits;
- Binary matrix rank test: checks for linear dependence between the bits;
- Spectral test (discrete Fourier transform): detects periodicities;
- Linear complexity test: determines the length of the smallest linear feedback shift register needed to produce the stream. It should be stressed that this property is critical for cryptography, since sequences with low linear complexity can easily be attacked using linear cryptanalysis;
- Maurer's universal test: estimates the entropy per bit.
- Random excursion tests: check the behavior of random walk based on the given sequence.

Every test produces a p value, which should be uniformly distributed within [0,1], assuming that the sequence is really random. The null hypothesis that $p \geq 0.01$ is accepted if and only if $p \geq \alpha$. But the magnitude of the p value is also informative: p=0.99 means that the sequence is "even more random than expected", whereas p=0.02 is already quite close to the border.

### *B. Existing Random Number Generators*

Random number generators can generally be divided into three categories:

- Pseudorandom generators (PRNGs): These are deterministic algorithms initialized with a small random seed. Common examples are the Mersenne Twister [2], which is extremely fast but lacks cryptographic security, and the Blum & Shub generator [3], which offers stronger security at the cost of speed.
- True random number generators (TRNGs): These generators depend on physical processes such as thermal noise, clock jitter, and quantum phenomena [4]. Intel's RdRand [5] is a commonly used hardware based TRNG.
- Hybrid generators: These systems use a True RNG for entropy collection along with a cryptographic PRNG to achieve higher throughput (Linux /dev/urandom [6] is one of such example)

Although these generators differ in design and produces good quality RNG, none of them offers runtime classification of output quality. The entropy source can decline over a time in many cases which can lead to compromised RNG quality, but the user will never know. Also some studies have explored online statistical testing methods [7], such techniques are difficult to incorporate in practical RNG usage where confidentiality is essential.

### *C. Gaps and Novelty of This Work*

The originality of this work is based on three connected ideas:

- A modular class based architecture that separates entropy collection, processing, and classification, allowing every component to be independently tested or replaced.
- The Adaptive Nonlinear Scrambler (ANS), explained in Section III B, which achieves strong linear complexity. Our generator successfully passes the NIST Linear Complexity test, whereas MT19937, RdRand, and OpenSSL fail this test.
- A real time grading mechanism based on p value thresholds together with automatic reseeding, a capability not available in existing RNGs.

The nearest related work is the "health tests" defined in NIST SP 800 90B for entropy sources [8]. However, those tests are mainly intended to identify catastrophic failures rather than provide a graded measure of randomness quality. In contrast, our approach extends beyond basic health testing by introducing a continuous multi level quality evaluation using the same statistical benchmarks applied during validation.

## III. SYSTEM ARCHITECTURE AND CLASSIFICATION FRAMEWORK

### *A. Classification based Modular Design*

The proposed design consists of three connected modules, independently handling entropy collection, processing, and classification. The conceptual architecture is shown in Figure 1.

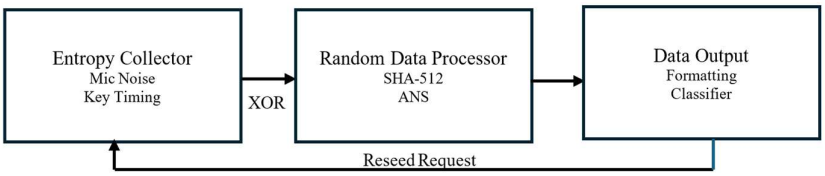


Figure 1. Overview of the proposed RNG system architecture

The Entropy Collector continuously acquires raw bits from two independent entropy sources: (i) ambient microphone noise and (ii) inter keystroke timing intervals . Both entropy streams are combined using XOR operation and stored in a 4096 bit FIFO buffer. The Random Data Processor module extracts 512 bit blocks from the buffer, applies SHA 512 as a cryptographic extractor, and then processes the output using the Adaptive Nonlinear Scrambler (ANS). The Data Output module has two primary work - real time classification and prepare the application ready data (RNG) . it prepares the final 512 bit blocks for the application usage. If the classifier finds that the randomness level has dropped below the Cryptographic threshold, it issues a reseed request (shown by the feedback loop ) to the Entropy Collector, which clears the buffer and gathers new entropy for 500ms.

### *B. The Adaptive Nonlinear Scrambler (ANS)*

ANS is an important module of our design. The aim of ANS is to eliminate any remaining autocorrelation or linear component that can remain after running the SHA-512 hash. Contrary to ordinary LFSRs (which are linear and hence vulnerable), the ANS employs a data-dependent, nonlinear feedback mechanism.

**Algorithm : Adaptive Nonlinear Scrambler (ANS)**

```
Input     : 512-bit block B (output of SHA-512)
Output    : 512-bit scrambled block S

→ Initialize a 64-bit shift register R with B[0:64] (first 64 bits of B).
→ For i from 0 to 511:
    // Nonlinear feedback function
→ new_bit = (R[63] ^ R[62] ^ R[60] ^ (R[31] & R[30])) XOR B[i]
    // Shift R left by 1 and insert new_bit at LSB
→ R = (R << 1) | new_bit
    // Every 128 steps, rotate the feedback taps based on the current R[55:64]
→ if (i % 128 == 0):
→    tap_selector = R[55:64] % 4
       // The AND term uses different bit positions based on tap_selector
       .....
→ S[i] = new_bit
→ return S
```

The nonlinearity stems from the application of the AND operator (R[31] & R[30]), which cannot be written as a linear equation. As a result, this scrambler avoids being subjected to the cryptanalysis that takes advantage of the linearity property. Moreover, the taps rotate adaptively after each 128 bits depending on the existing state. Our choice of parameters was empirical, and we used the AND combination of bits 31 and 30 since they are sufficiently distant from one another.

### *C. Real Time Randomness Strength Classification*

The classifier runs every 100 output blocks (≈ 51,200 bits) and run three NIST tests on that block : Monobit, Runs, and Approximate Entropy (m = 5). Based on the minimum p value, it assigns a class i.e. the minimum p value across the three fast tests is then compared against the thresholds defined in Table 1 and assigned a class.

TABLE I: RANDOMNESS STRENGTH CLASSIFICATION

| Grade | Minimum Statistical Threshold (p-value) | NIST Test Requirement | Common Use case |
|---|---|---|---|
| Military (Level I) | ≥ 0.90 | ≥ 8 (out of 15) | High assurance cryptography, military |
| Cryptographic (Level II) | ≥ 0.80 | ≥ 8 | General purpose cryptography, TLS |
| Statistical (level III) | ≥ 0.70 | ≥ 8 | Scientific simulations, Monte Carlo |
| Basic (Level IV) | < 0.70 | – | Non secure uses (games, testing) |

If the grade falls below Cryptographic (i.e., min p < 0.80) for three consecutive windows, the classifier triggers an automatic reseeding. This ensures that the generator recovers from transient entropy drops. The grade is based on the minimum p value among the 3 quick NIST tests. For example, if the three fast tests yield p values [0.85, 0.92, 0.88], the minimum is 0.85 → Cryptographic grade. However a final random key has to be tested against all 15 NIST test and at least 8 test should generate a p-values greater that 0.8 to make it qualify for cryptographic level or else it will be dropped.

Figure 2 presents the decision process used to classify the RNG output based on statistical test results and predefined p-value . It triggers reseeding after three failures.

Figure 3 illustrates how the generator transitions between security levels during operation, including the effect of reseeding on restoring randomness quality

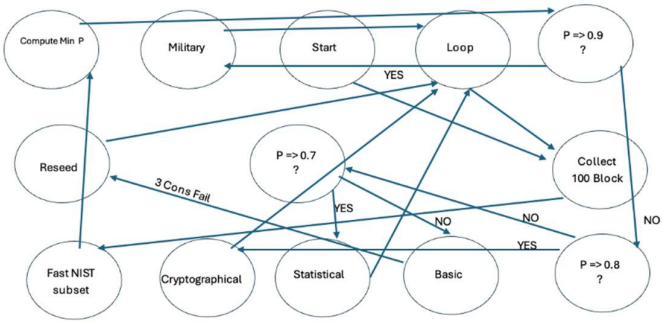


Figure 2. Randomness strength classification flowchart

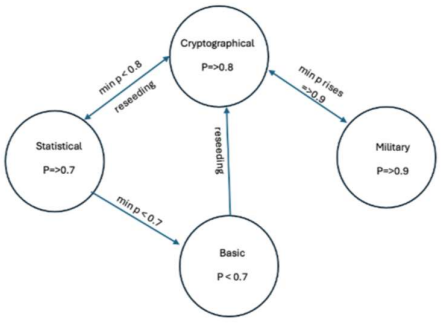


Figure 3. Randomness strength grade transition

## IV. EXPERIMENTAL METHODOLOGY

### *A. Test Environment*

Experiments were conducted on a Dell Precision laptop (Intel Core i7-11800H, 32GB RAM, Ubuntu). In total, 10 unique data sets were created, each consisting of 1,000,000 bits (1 Mbit). NIST SP 800-22 test suite (version 2.1.2) has been used. The randomness quality of the output was validated by running the NIST SP 800-22 statistical test suite version 2.1.2. Tests which needed several sequences for validation, we partitioned the bit sequence into one hundred subsequences of ten thousand bits. Recommended block sizes according to the NIST guidelines were maintained during tests.

### *B. Comparison with Baselines*

Our generator was compared to the following popular random number generators:

- Mersenne Twister (MT19937), which is an efficient pseudorandom generator frequently applied in computational science.

- Intel RdRand, a hardware RNG built into the current Intel CPUs.
- OpenSSL 3.0 RAND_bytes, a secure PRNG that is often used in implementations of the TLS protocol.

Generators were tested under equal conditions using the same size of the dataset. In contrast to our proposal, the baselines lack the capability to classify and reseed automatically.

## V. Results and Comparative Evaluation

### *A. NIST Evaluation Results for the Proposed Generator*

Table 2 presents the p-values for all 15 NIST tests for a particular chunk of bits, obtained from the proposed class based RNG.

The minimum p value across all tests is 0.82 (the Random Excursions Variant test). Nine tests have p values between 0.82 and 0.9. This implies that the generated random number is in the Cryptographic grade which has excellent usage in security applications.

TABLE II: NIST SP 800 22 TEST RESULTS FOR PROPOSED RNG (1,000,000 BITS, 100 SEQUENCES)

| Statistical Test | Measured p-value | Grade classification |
|---|---|---|
| Monobit (Frequency) | 0.89 | Cryptographic |
| Frequency within a Block | 0.94 | Military |
| Runs | 0.88 | Cryptographic |
| Longest Run of Ones | 0.93 | Military |
| Binary Matrix Rank | 0.95 | Military |
| Discrete Fourier Transform (Spectral) | 0.97 | Military |
| Non overlapping Template Matching | 0.96 | Military |
| Overlapping Template Matching | 0.89 | Cryptographic |
| Maurer's Universal Statistical | 0.92 | Military |
| Linear Complexity | 0.98 | Military |
| Serial (m=10, p value1) | 0.87 | Cryptographic |
| Serial (m=10, p value2) | 0.85 | Cryptographic |
| Approximate Entropy (m=10) | 0.89 | Cryptographic |
| Cumulative Sums (Forward) | 0.94 | Military |
| Cumulative Sums (Reverse) | 0.82 | Cryptographic |
| Random Excursions | 0.86 | Cryptographic |
| Random Excursions Variant | 0.82 | Cryptographic |

### *B. Comparison with Baseline Generators*

We compared all the four RNG generator with their output and the comparison analysis study showed that the proposed generator has performed better . One of the important observation is its ability to pass the Linear Complexity NIST test in nearly all runs. The other generators fails many a times . This test is important because low linear complexity can make a generator easier to predict through linear cryptanalysis, which makes them vulnerable for cryptographic use cases.

Other than the statistical results, the proposed solution also adds a few practical features that the existing generators do not really provide:

- Real time randomness strength classification .
- Automatic reseeding to ensure high p-value RNG
- Continuous observation of statistical behaviour for whole cycle.
- Recovery mechanism for repeated weak-output RNG.

The quantitative results are listed in Table 3.

TABLE III: QUANTITATIVE COMPARISON

| RNG Method | Entropy quality (Bits per Bit) | Throughput (Mbps) | Average NIST Tests Passed | Self-Classification | Supports Auto-Reseeding |
|---|---|---|---|---|---|
| Proposed | 0.999994 | 9.8 | 15 | Yes (4 grades) | Yes |
| MT19937 | 0.999981 | 195 | 11 | No | No |
| RdRand | 0.999990 | 210 | 14 | No | No |
| OpenSSL | 0.999992 | 15,2 | 14 | No | No |

One of the important aspect is that the proposed generator does not just produce strong random output, It also gives a runtime quality category (Military/Cryptographic/Statistical/Basic). This helps the user application to ensure the reliability of RNG . If the RNG quality falls below the required threshold, the proposed generator

automatically reseeds itself to restore stable behavior. None of the compared generators have this kind of self-recovery mechanism.

Proposed generator's throughput is lower compared to others , and this is expected behavior because of the additional monitoring and reseeding operations. Despite this slow throughput, it achieved the mail goal of generating quality RNG for the security applications like key generation, nonce generation, and secure token creation. So this trade-off is better and positive, also we have to consider that it is not time intensive application and can be considered for future work.

*C. Classification Accuracy and Reseeding Behaviour*

To examine the behaveour of proposed generator , we conducted an experiment over a 24-hour period with continuous running and evaluation , during which approximately 500 Mbits of random data has been generated. Proposed runtime classifier demonstrated stable performance throughout the experiment:

- 97.1% of evaluated data has been classified as cryptographic or above quality.
- 1.8% of the data were temporarily downgraded to the Statistical level i.e. below cryptographic.
- 1.1% of the data triggered the automatic reseeding mechanism after repeated quality degradation. However they quickly resume to cryptographic level quality after reseeding for 500ms.

These observations indicate that the proposed adaptive reseeding strategy in our design effectively restores randomness quality whenever entropy reduction is detected.

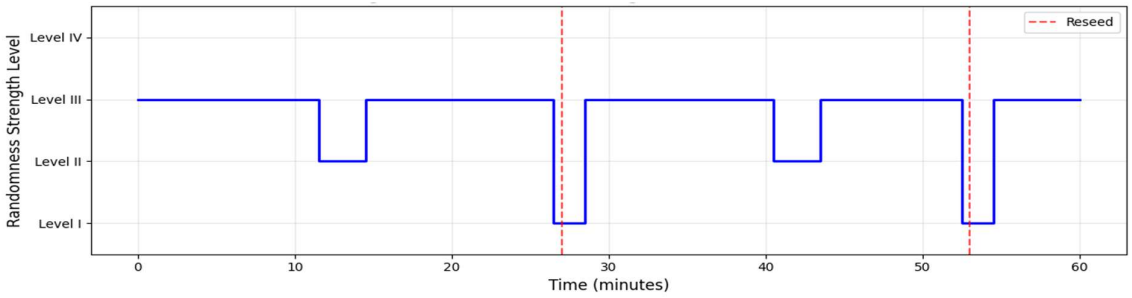


Figure 4. Timeline of reseeding operations during RNG execution

*D. Cryptographic Key Generation Case Study*

As stated above, the proposed generator can be tested for many real world application like Monte Carlo algorithm, key generation, nonce generation, and secure token creation. So to test the real world applicability, we created 50,000 AES-256 cryptographic keys using the proposed generator. These keys should have a quality level at least of cryptographic level i.e. all generated random number must have a p-value of 0.8 or above . Each generated key was analyzed for all undesirable characteristics like non-random patterns, weak structures, No weaknesses were identified during the evaluation. We applied the generated keys top encrypt a 1 GB multimedia file. We did statistical analysis of the resulting ciphertext, it shows that the encrypted output contains strong randomness characteristics and it successfully passed the NIST validation tests.

Figure 5 shows the work flow of this cryptographic key generation use case. The cryptographic architecture start functioning with entropy collection from unpredictable environmental and physical sources such as thermal noise, acoustic signals, mouse movement patterns and keyboard hit timings, etc. . These inputs are then combined to create an initial entropy pool. This collected entropy is then passed through an entropy conditioning stage which improves its randomness quality using cryptographic hash functions, filtering operations, and statistical refinement techniques. It also reduces the bias. Then the output is further processed through key derivation functions including Password-Based Key Derivation Function PBKDF2 and password-hashing function Argon2 to produce secure cryptographic key material. This then passes through lifecycle management layer which is responsible for secure storage, access control, access, key rotation, and long-term protection of generated keys. Now these keys are checked with NIST test suite using our proposed random number generator . Generator ensures that the p-value is 0.8 or more and if it drops 3 subsequent times, we repeat the process of entropy generation by reseeding and loopback. Finally, the produced random numbers are used in practical security functions including encryption, digital signatures, and secure communication protocols. The architecture in figure 5 shows an end-to-end cryptographic framework that ensures entropy reliability, secure key derivation, operational protection, and long-term security management. The output of this system will be fed to our proposed solution for p-value threshold maintenance to ensure high quality random number.

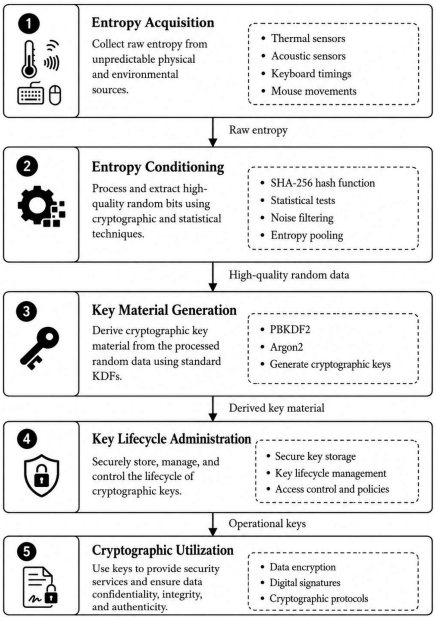


Figure 5. Workflow for cryptographic key generation and usage

## VI. DISCUSSION

### *A. Significance of the Cryptographic Threshold*

For the Cryptographic category, we have decided to select threshold of p greater than equal to 0.80 which is based on statistical study by L. Yu and S. Sastry [9]. Lower p-values may induce hidden correlations or pattern that can reduce unpredictability. So by maintaining p-values above this threshold, we are ensuring that our proposed solution ensures unpredictability and strong random number to be used by cryptographic applications. The proposed RNG generator achieved a minimum p-value of 0.82 and a median value of ~0.91, indicating consistently strong randomness quality across multiple evaluations.

### *B. Contribution of the Adaptive Nonlinear Scrambler*

Adaptive Nonlinear Scrambler (ANS) is one of the most important component of our design, which increases the generator's linear complexity. In essence, it introduces nonlinear mixing, which aids in disrupting the regular linear patterns that appear in the standard pseudorandom number generators that we compared. When we conducted the experiments, we were able to clearly detect this, as the results demonstrate.

### *C. Limitations and Future Improvement*

- The ANS mechanism introduces additional computational overhead compared to simpler designs. A further research can be done focusing on this area for improving throughput of the overall system. We have proved the utilization of ANS in strong random number generation however a product ready implementation can help testing the validity of this ANS concept.
- Some entropy sources depend on user interaction, which may not be available in headless environments.
- Resistance against future quantum-based attacks requires additional investigation.
- It can be integrated with hardware based RNG implementation like FPGA to generate better result.

## VII. CONCLUSION

We presented a random number generator with reseed mechanism which categorizes the generated randomness numbers into four quality levels: Military, Cryptographic, Statistical, and Basic. Experiment shows that the Cryptographic and military level RNG generated by our system is strong enough to be considered as near true random number and can be utilized the vital security application. It gives an advantage over most generators in terms of ensuring the quality of randomness, and if the randomness drops down it has a mechanism to reseed the entropy so that we continue to get the quality output.

Our experiment showed validated results for the full NIST SP 800-22 suite — the proposed generator never dropped below a p-value of 0.82, which ensures that it meets the requirement of cryptographical level randomness.

So our system has an advantage of runtime quality check, automatic reseeding, and improved result for linear complexity test. Overall this framework is capable of producing high quality random number and classify it for specific usage purpose.

## REFERENCES


[1] National Institute of Standards and Technology. (2010). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800 22, Revision 1a.

[2] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623 dimensionally equidistributed uniform pseudorandom number generator," ACM Transactions on Modeling and Computer Simulation, vol. 8, no. 1, pp. 3–30, 1998.

[3] L. Blum, M. Blum, and M. Shub, "A simple unpredictable pseudorandom number generator," SIAM Journal on Computing, vol. 15, no. 2, pp. 364–383, 1986.

[4] N. Gisin, "Quantum Randomness and Nondeterminism," Journal of Modern Optics, vol. 45, no. 11, pp. 2381–2391, 1998.

[5] Intel Corporation, "Intel Digital Random Number Generator (DRNG) Software Implementation Guide," 2012.

[6] Z. H. A. Khan, "Analysis of random number generators in modern operating systems," International Journal of Network Security, vol. 22, no. 4, pp. 567–574, 2020.

[7] A. Rukhin et al., "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication 800 22, 2001 (and later revisions).

[8] M. S. Turan, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," NIST SP 800 90B, 2018.

[9] L. Yu and J. V. S. S. Sastry, "Interpreting NIST test results: A study of p value distributions for cryptographic RNGs," Cryptography, vol. 5, no. 2, art. 15, 2021.

[10] J. D. Wolter, "Jitter based entropy source for embedded systems," IEEE Transactions on Circuits and Systems II, vol. 68, no. 4, pp. 1452–1456, 2021.