Deterministic Random Bit Generators: A Comparison of NIST-Approved Techniques

Tomas Bedoya
Daniel Andres Jaimes Cárdenas
Eder Leandro Carbonero Baquero
Universidad de los Andes, Colombia

***Abstract*—The following paper centers around several methods of random number generation. It aims to comprehend the essential concepts behind RNG, including pseudo and true random generation. Additionally, it explores different approaches to randomness that are used today following the NIST SP 800-90 standards, both reliant on entropy sources as well as algorithmic techniques, and compares their randomness with a series of statistical tests to determine which operate better and suggest appropriate uses for the different methods.**

***Index Terms*—CTR DRBG, Counter mode Deterministic Random Bit Generator, NIST, National Institute of Standards and Technology, AES, Advanced Encryption Standard, RBG, Random Bit Generator, DRBG, Deterministic Random Bit Generator, RNG, Random Number Generator, PRNG, Pseudorandom Number Generator, TRNG, True Random Number Generator, DRNG, Digital Random Number Generator, ENRNG, Enhanced Random Number Generator, HSM, Hardware Security Module**

I. General Definitions

With the aim of providing a broader understanding of the various concepts addressed in this document, the following are definitions of some of the acronyms that will be mentioned throughout the text.

Acronym	**Meaning**
CTR_DRBG	Counter mode Deterministic Random Bit Generator
NIST	National Institute of Standards and Technology
AES	Advanced Encryption Standard
RBG	Random Bit Generator
DRBG	Deterministic Random Bit Generator
RNG	Random Number Generator
PRNG	Pseudorandom Number Generator
TRNG	True Random Number Generator
DRNG	Digital Random Number Generator
ENRNG	Enhanced Random Number Generator
HSM	Hardware Security Module

Table I
Acronyms definitions.

II. Introduction

Due to the deterministic nature of traditional computers, which is what most people and appliances use to this day and will continue being so for the foreseeable future, Random Number Generation (RNG) has proven to be a challenge. Due to the reliance of cryptographic algorithms, communication protocols and other security systems in random values, there's a constant race to find ever-better sources of randomness that follow the international standards of quality; the higher the randomness of a source, the greater the security it provides. This proves to be a challenge, however, for true randomness is difficult to find. Computers themselves cannot produce it and rely on Pseudo Random Number Generation (PRNG) to simulate random values. Other techniques, such as finding random measurements on certain like the time between user key strokes or mouse movements have proven to be unsatisfactory when put under statistical scrutiny (Mechalas, 2018). To guarantee that sensible processes utilize proper random sources, the NIST structured the SP 800-90 Standards for Random Number Generation (RNG) for safe RNG in cryptographic contexts. The following sections delve deep into several RNG methods, both of pseudo and true randomness, that follow these standards. They explore the different entropy sources they use (or don't), analyses how they extract random number chains from them, and evaluate the quality of their randomness through a series of statistical evaluations. Our aim is to understand better where RNG stands today, to evaluate how they fare against each other, and determine which of these methods work better for which applications.

As discussed in [2], the NIST standards provide a comprehensive framework for deterministic random bit generators.

III. Key Concepts

Most RNG methods can be classified into one of two categories. The first of these is Pseudo Random Number Generation, sometimes called Deterministic Random Bit Generation (DRBG) (Cao et al., 2022). A PRNG is a deterministic algorithm that produces numbers that appear to be random. It requires a seed value to generate a seemingly random number, and due to its deterministic nature, the same seed will produce the same value every time. PRNGs experience periodicity, for after exhausting all possible internal variations, it will repeat cycles that will reiterate on the sequences of produced numbers. Good PRNG algorithms, however, manage to display good statistical behaviors, with some having periods in order of magnitudes so large they become negligible. Due to their algorithmic nature, PRNGs are quick and scalable; their deterministic nature is also desirable in experiments where replicability is key. However, they are extremely unsafe key producers for cryptographic measures, for a backtrack of the algorithm or the knowledge of the seed reveal the output in its entirety (Mechalas, 2018). True Random Number Generators (TRNG), on the other hand, aim to produce true random values. To achieve this, TRNG relies on entropy sources, which extract true randomness by extracting information from physical phenomena. The two main types of entropy sources are dynamic entropy sources, which extract true random values from indeterminate physical processes like thermal noise or atmospheric noise, and static entropic sources, which extract randomness from randomly occurring properties in the hardware components of the computer as a result of the semiconductor manufacturing process, which become stable once the device is finished. These properties can be found in chip and are used for things like authentications (Cao et al., 2022). What is essential is that an entropy source extracts its randomness from the physical world, which means that no "randomness" generated by a procedural method can be considered an entropy source. TRNGs are desirable where safety is essential, and show constant distributions that guarantee unpredictability, but are usually slow to output a number due to their need to measure physical phenomena; this takes time and is computationally costly, which makes them not very scalable. Some methods can be classified into particular categories of RNG. Cascade Construction Random Number Generator (CCRNG), for example, relies on an entropy source to supply an "entropy buffer", which is then used to provide cryptographically secure PRNG. Digital Random Number Generators (DRNGs) is an approach that builds the RNG on the processor's hardware directly, and uses a combination of CCRNG and dynamic entropy sources to create random streams. Even so, this categories are usually some sort of combination of PRNG and TRNG to different degrees, and combining both methods is becoming more common in the industry to tackle the strengths and weaknesses of both methods.

IV. Studied Methods

The following RNG methods are the ones that will be seen in detail for the rest of this report.

A. Randomness Beacon Architecture

As defined by NIST, a **Randomness Beacon** is a service that provides periodic pulses of random numbers. These pulses are timestamped, hash-chained, and signed [7].

A beacon is composed of:

- An **engine**, a computer with clear physical boundaries where pulses are formed and the beacon app runs.
- A **frontend** providing access to the latest pulse and history of all pulses generated.
- An **HSM** (Hardware Security Module), independent from the engine, used for cryptographic operations.
- At least **two RNGs**, with at least one being independent from the beacon's engine.

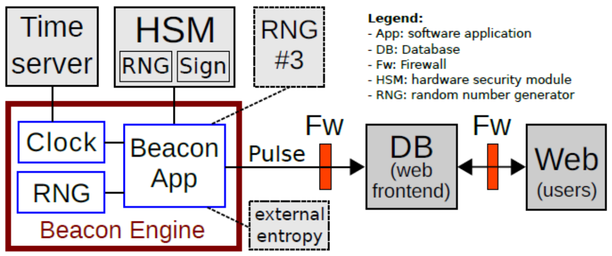

Figure 1. The general layout of a Randomness Beacon [7].

The **NIST Randomness Beacon** produces a 512-bit pulse every minute and makes it publicly available with a timestamp. The beacon currently uses three RNGs:

- The Intel RNG included in the engine,
- A QRNG based on photon detection developed by NIST,
- The hardware RNG inside the HSM [5].

B. The Intel RNG

The Intel RNG is a DRNG integrated into many Intel CPUs. It utilizes the `RDRAND` and `RDSEED` instructions, which are part of the Intel 64 architecture, and implements them directly in hardware. Its architecture includes an entropy source, a conditioner to convert entropy into random numbers, and two parallel outputs: a DRBG and an ENRNG [8].

The entropy source measures thermal noise within the silicon of the CPU and outputs randomness as a binary stream at a rate of 3 GHz.

C. The QRNG Based on Photon Detection

The QRNG based on photon detection is described by Zhang et al. [10]. It operates as follows:

> At each trial, a horizontally polarized single photon is emitted from a source, and then measured randomly along either the X-basis (diagonal/anti-diagonal polarization basis) to generate a random bit or the Z-basis (horizontal/vertical polarization basis) to verify the prepared state.

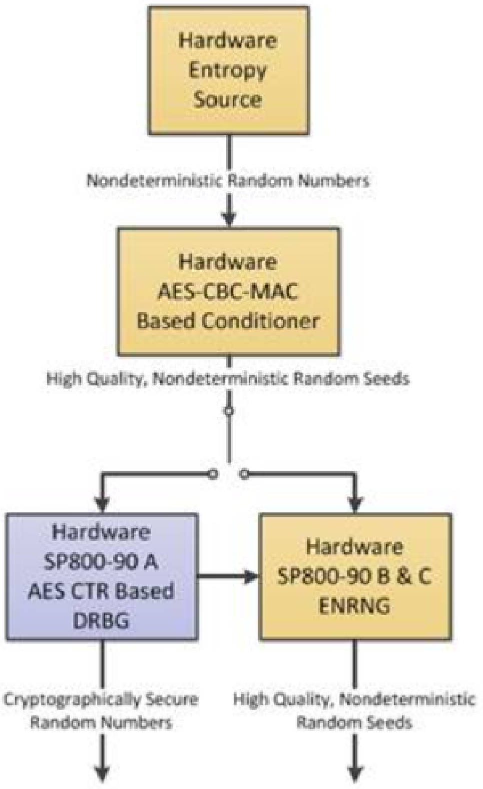
Hardware Entropy Source
Nondeterministic Random Numbers
Hardware AES-CBC-MAC Based Conditioner
High Quality, Nondeterministic Random Seeds
Hardware SP800-90 A AES CTR Based DRBG
Hardware SP800-90 B & C ENRNG
Cryptographically Secure Random Numbers
High Quality, Nondeterministic Random Seeds

Figure 2. Intel's DRNG Component Architecture [8].

In practice, the NIST QRNG measures photons prepared in a superposition of two time bins—early (t_e) and late (t_l)—using **time-bin encoding**. Photons pass through an unbalanced Mach-Zehnder interferometer (MZI), where the path-length difference matches the separation between t_e and t_l. Detection times correspond to:

- t_e or t_l: no interference (Z-basis measurement)
- t_m: interference (X-basis measurement)

The basis choice is passive and random, depending on the photon's path through the MZI. Outcomes are fundamentally unpredictable due to quantum mechanics, ensuring certifiable quantum randomness.

D. Hashing of RNG Output

The NIST Randomness Beacon concatenates outputs from its RNGs and applies a SHA-512 hash. The resulting hash is the *localRandomValue* of the pulse [6].

E. CTR_DRBG: Counter Mode Deterministic Random Bit Generator

CTR_DRBG is a standardized deterministic random bit generator built from a block cipher operating in counter (CTR) mode, as defined in NIST SP 800-90A [1]. It transforms a secure symmetric cipher (e.g., AES) into a cryptographically strong pseudorandom bit source. The counter mode ensures each block is unique by incrementing a counter for each output, preventing repetition under the same key and seed. This construction is widely used in cryptography for generating keys, IVs, and session tokens.

F. Random Number Generation

For this exercise, three different forms of random number generation were performed. The first was through the *CTR_DRBG* algorithm, which used random keys generated by the `random` function of the Python programming language version 3.9 as seeds and was executed on a MacBook Pro M3 Pro computer. This detail may lead to different results if the experiment is replicated. However, we assume that Python's `random` function generates a uniform randomness distribution, simulating an entropic system for generating random bits.

For the second option, a sequential range of seeds was chosen, which were formatted in 32 bits to ensure compatibility with AES encryption. The sequence started at 1 and increased by one until reaching 50 million seeds. These seeds were then used to execute the *CTR_DRBG* algorithm, which generates the pseudo-random bit values.

The third approach made use of the Beacon system. In this case, the official NIST API was consumed to retrieve the last 15,600 random bits, generated from 512-bit values, obtained from the historical records maintained by the system.

V. CTR_DRBG Counter mode Deterministic Random Bit Generator

CTR_DRBG (Counter mode Deterministic Random Bit Generator) is a standardized method for constructing a deterministic random bit generator (PRNG) using a block cipher operating in counter (CTR) mode. This technique is defined in NIST Special Publication 800-90A, titled *"Recommendation for Random Number Generation Using Deterministic Random Bit Generators"*. Essentially, CTR_DRBG transforms a secure symmetric cipher—such as AES—into a cryptographically strong source of pseudorandom bits. The counter mode ensures that each generated block is unique by systematically incrementing a counter value for each new data request, thus preventing the repetition of output sequences under the same key and seed. This approach is widely used in cryptographic applications requiring high security and reliability in random data generation, such as key generation, initialization vectors, and session tokens.

How It Works

The internal state of the CTR_DRBG consists of two components:

- **V**: A state vector that acts as a counter.
- **Key**: A symmetric encryption key (commonly 128 or 256 bits).

To generate random output, the algorithm encrypts sequential values derived from `V` using the current `Key`. For each block of output, the counter `V` is incremented.

A secure seed—typically composed of entropy input, a nonce, and optional personalization string—is required to properly initialize the internal state.

CTR_DRBG utilizes an approved block cipher algorithm in counter (CTR) mode, as described in [1]. Unlike the standard CTR mode, it allows the counter field to occupy only a portion of the cipher input block, as specified in [3].

For context, the block size depends on the cipher: 64 bits for TDEA and 128 bits for AES. The same block cipher algorithm and key length are used consistently throughout all encryption operations in the DRBG. These parameters must meet or exceed the required security level of the intended application.

The construction of *CTR_DRBG* revolves around an internal update function, `CTR_DRBG_Update`, illustrated in Figure 3. This function is invoked during instantiation, reseeding, and random bit generation to update the internal state using newly provided entropy or additional input. It also ensures that the internal state changes appropriately after every generation step.

Figure 4 outlines the full operation of *CTR_DRBG* in three stages: instantiation, generation, and reseeding.

To provide an overview of what can be found in the source article, we can say that *CTR_DRBG* is a deterministic random bit generator that uses an approved symmetric encryption algorithm, such as AES, in counter (CTR) mode to produce secure random bits. Its internal state consists of a counter vector called `V` and a secret key, `Key`, which are updated through an internal function named `CTR_DRBG_Update` during instantiation, generation, and reseeding, thus ensuring the cryptographic security of the generator. At each step, `V` is encrypted with `Key` and incremented to generate non-repeating pseudorandom blocks. It requires a secure initial seed with sufficient entropy to avoid predictability and can be periodically reseeded to maintain long-term security. This flexible design allows for the use of variable block sizes and key lengths depending on the algorithm (e.g., 128 bits for AES) and meets robust cryptographic standards, making it reliable for applications that require secure random number generation.

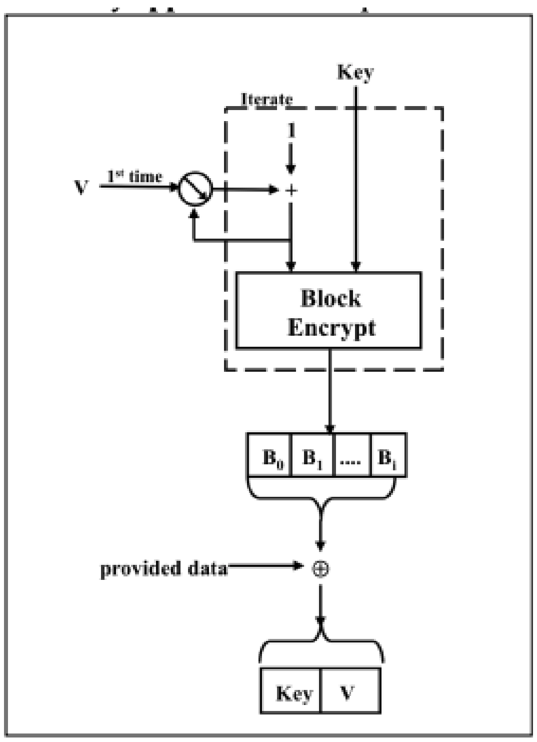

Figure 11: CTR_DRBG Update Function

Figure 3. CTR_DRBG Update function as shown in [9].

A. AES - Advanced Encryption Standard

AES (Advanced Encryption Standard) is a symmetric encryption algorithm widely used across various applications for securing data. It was established by the National Institute of Standards and Technology (NIST) in 2001 as a replacement for the older Data Encryption Standard (DES). AES operates on fixed-size blocks of data (128 bits) and supports key sizes of 128, 192, or 256 bits, providing a high level of security against brute-force attacks. The algorithm employs a series of transformations, including substitution, permutation, and mixing, to encrypt plaintext into ciphertext and vice versa. Its efficiency and robustness have made it the de facto standard for encryption in many cryptographic protocols and systems.

B. Seeds obtaining and random bits generation

C. Distribution of Random Bits Generated with Python Random Seeds

The distribution results of each technique are separated by the method used to obtain the seed versus

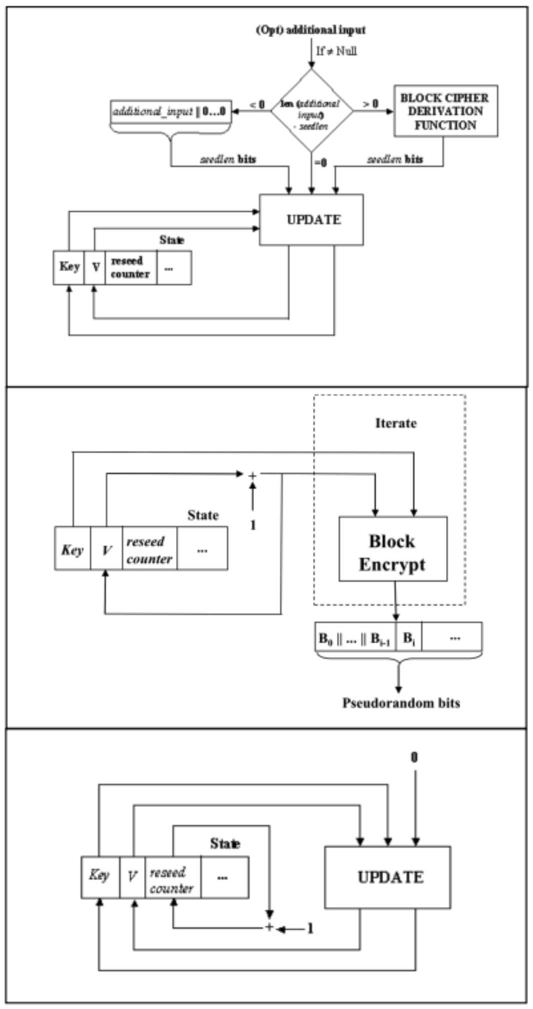

Figure 12: CTR-DRBG

Figure 4. The three stages of CTR_DRBG operation as illustrated in [9].

the technique used to distill and obtain the random big number, so we can observe different results for each case.

To view the distribution of values, distribution graphs were used, either standard or uniform, depending on the input data. In our experiments, uniform distributions were shown. 8-bit segments were taken, dividing the entire input data set into byte blocks, obtaining bytes from 0 to 255. This range of possibilities gives us a visual of how the results are distributed.

We begin with the first technique, which was the generation of 6.25 million 32-byte seeds that were used to generate 6.25 million random bytes, or 50 million random bits, using the *CTR_DRBG* technique. The graph we see below leads us to the conclusion that there is a uniform distribution of the results without pronounced peaks, which are a symptom of an effective system in generating random bits.

D. *Distribution of Random Bits Generated with Sequential Seeds*

For the following test, sequential seeds were used starting at 1 and reaching 6.5 million in the set of natural numbers. This number was converted to binary and turned into a 32-bit number, then it was used as a seed to generate the distribution with the *CTR_DRBG* method, showing a uniform distribution without much variation. Unexpectedly, it is seen that the distribution behaves very similar to obtaining reliable seeds with true entropy.

E. *Distribution from the NIST Beacon*

Finally, we reported the Beacon, and this time we used the official NIST API, where we downloaded a history of just over 15,600 pulses. Each pulse corresponds to a hexadecimal message that corresponds to a 512-bit random number, which allowed us to generate the histogram and create the tests.

F. *Quantum Random Number Generator*

Quantum Random Number Generation (QRNG) harnesses the inherent indeterminacy of quantum mechanics to produce random sequences theoretically unpredictable by classical means. At its core, QRNG utilizes qubits which, through the principle of superposition, can represent multiple states simultaneously until a measurement is performed. This act of measurement is pivotal, compelling a qubit to collapse from its superposition into a definite classical state (0 or 1), with the specific outcome governed by quantum probabilities, forming the basis of true randomness. The translation of these quantum mechanical principles into a practical random bit stream involves a carefully orchestrated process. It begins with the precise preparation of quantum states specifically designed to maximize outcome uncertainty. Subsequent interactions with these prepared states, followed by their measurement, yield a sequence where each outcome, dictated by the probabilistic nature of quantum collapse, contributes a bit to the forming random sequence. The quantum random number generation (QRNG) methodology implemented, which makes use of code developed by Gehad Salem Fekry, software engineer and quantum researcher and publicly available on GitHub [4], begins with the construction of a quantum circuit comprising N qubits, where N dictates the length of the random bit string to be generated. Each qubit is individually subjected to a Hadamard (H) gate operation. This crucial step transforms the qubit from a definite initial state into an equal superposition of the computational basis states $|0\rangle$ and $|1\rangle$, described by the state vector $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Consequently, upon measurement, each qubit has an equal probability of collapsing to either 0 or 1, providing the fundamental source of randomness.

The collective measurement of all N qubits yields a single N-bit string derived from these inherently probabilistic quantum events. For the practical realization of these random bits, the constructed quantum circuit is executed on a quantum backend, accessed via the Qiskit Runtime Service which can select an operational quantum processor or a simulator. Prior to execution, the circuit undergoes a transpilation process, adapting it to the specific gate set and connectivity constraints of the target backend. The transpiled circuit is then run for a significant number of shots (e.g., 5000 as specified in the implementation), with the `memory=True` option enabling the retrieval of the exact bit string outcome from each individual shot. These collected bit strings constitute the raw output of the QRNG process.

VI. Statistical Testing

Due to their protagonic relevance in cryptography and security, as mentioned before, the evaluation of random and pseudorandom number generators (RNGs/PRNGs) is a critical aspect of their deployment. The statistical properties of the generated sequences must approximate those of true random sequences to ensure unpredictability and reliability. The National Institute of Standards and Technology (NIST) Special Publication 800-22 Rev 1a provides a widely recognized suite of statistical tests for this purpose, offering tools to detect deviations from randomness in binary sequences. While no finite set of tests can provide absolute certification of randomness, these tests serve as an essential initial step in assessing a generator's suitability.

The five tests selected from the NIST suite for this analysis—Frequency (Monobit) Test, Frequency Test within a Block, Runs Test, Maurer's "Universal Statistical" Test, and the Cumulative Sums (Cusum) Test—were chosen to provide a balanced and comprehensive assessment. This selection emphasizes tests that are both fundamental in their approach and offer insights into different potential weaknesses of an RNG, ranging from basic distributional properties and sequential dependencies to more complex characteristics like data compressibility, which relates to the sequence's entropy. The aim is to cover a diverse range of statistical attributes with tests that are also relatively accessible in their conceptual underpinnings, facilitating a clear interpretation of a generator's performance.

Selected Statistical Tests: Description and Interpretation

A. Frequency (Monobit) Test

The **Frequency (Monobit) Test** is the most fundamental test, examining the overall proportion of zeros and ones in the entire binary sequence. Its purpose is to determine if the number of ones and zeros in a sequence are approximately equal, as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to $1/2$ by converting the bits into a sequence of $+1$ and -1 values, summing these values to obtain S_n, and then computing a test statistic

$$s_{\text{obs}} = \frac{|S_n|}{\sqrt{n}}.$$

A P-value is subsequently calculated using the complementary error function,

$$P\text{-value} = \operatorname{erfc}\left(\frac{s_{\text{obs}}}{\sqrt{2}}\right).$$

If the computed P-value is less than the chosen significance level α (typically 0.01), the sequence is considered non-random; otherwise, it is considered random with respect to this test. A small P-value suggests that the sum S_n is too large, indicating an excess of ones or zeros in the sequence. NIST recommends a minimum sequence length of 100 bits for this test.

B. Frequency Test within a Block

The **Frequency Test within a Block** focuses on the proportion of ones within M-bit blocks of the sequence. Its objective is to determine whether the frequency of ones in an M-bit block is approximately $M/2$, as would be expected under an assumption of randomness. The input sequence is partitioned into $N = \lfloor n/M \rfloor$ non-overlapping blocks. For each block, the proportion of ones (π_i) is calculated, and a χ^2 statistic is computed as

$$\chi^2(\text{obs}) = 4M \sum_{i=1}^{N} \left(\pi_i - \frac{1}{2}\right)^2.$$

The P-value is then determined using the incomplete gamma function,

$$P\text{-value} = \operatorname{igamc}\left(\frac{N}{2}, \frac{\chi^2(\text{obs})}{2}\right).$$

If the P-value is less than α (e.g., 0.01), it suggests non-randomness, indicating that at least one block deviates significantly from the expected equal proportion of ones and zeros. Recommended input parameters include $n \geq 100$, $M \geq 20$, $M > 0.01n$, and $N < 100$.

C. Runs Test

The **Runs Test** examines the total number of runs in the sequence, where a run is defined as an uninterrupted sequence of identical bits. The purpose is to determine if the oscillation between zeros and ones is too fast or too slow, which would indicate a deviation from randomness. A prerequisite is that the sequence passes a basic frequency check: if the proportion of ones, π, satisfies

$$|\pi - 1/2| \geq \tau, \quad \text{where} \quad \tau = \frac{2}{\sqrt{n}},$$

the Runs Test is not performed. If applicable, the test statistic $V_n(\text{obs})$, the total observed number of runs, is calculated. The P-value is then

$$P\text{-value} = \operatorname{erfc}\left(\frac{|V_n(\text{obs}) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right).$$

A P-value less than α (e.g., 0.01) leads to the conclusion that the sequence is non-random. A large value of $V_n(\text{obs})$ suggests the oscillation is too fast, while a small value indicates it is too slow. A minimum sequence length of 100 bits is recommended.

D. Maurer's "Universal Statistical" Test

Maurer's "Universal Statistical" Test evaluates the compressibility of the sequence, focusing on the number of bits between matching L-bit patterns. The test aims to detect whether a sequence can be significantly compressed without loss of information; a significantly compressible sequence is considered non-random. The n-bit sequence is divided into an initialization segment of Q L-bit blocks and a test segment of K L-bit blocks. For each L-bit block in the test segment, the algorithm identifies the distance (in blocks) to its most recent previous occurrence found in the sequence processed so far. The test statistic, f_n, is the average of the base-2 logarithm of these distances over all K test blocks. This observed f_n is compared to a theoretical expected value for the given L, $\text{E}(L)$, and a P-value is computed as

$$P\text{-value} = \operatorname{erfc}\left(\frac{|f_n - \text{E}(L)|}{\sqrt{2}\sigma}\right),$$

where σ is a pre-calculated standard deviation. If the P-value is less than α (e.g., 0.01), the sequence is deemed non-random, suggesting it is compressible. This test requires long sequences, with L typically between 6 and 16, and n chosen such that $K \approx 1000 \cdot 2^L$ and $Q = 10 \cdot 2^L$.

E. Cumulative Sums (Cusum) Test

The **Cumulative Sums (Cusum) Test** focuses on the maximal excursion (from zero) of a random walk defined by the cumulative sum of adjusted $(-1, +1)$ digits in the sequence. The purpose is to determine if this cumulative sum is too large or too small relative to the expected behavior for random sequences. The (0,1) sequence is converted to a sequence of X_i values of -1 and $+1$. Partial sums S_k are computed. The test statistic z is the maximum absolute value among these partial sums:

$$z = \max_{1 \le k \le n} |S_k|.$$

The test can be applied in a forward direction (mode=0) or a backward direction (mode=1). The P-value is calculated based on z and n, involving sums of normal cumulative distribution function evaluations. If the P-value is less than α (e.g., 0.01), the sequence is considered non-random. Large excursions suggest an excess of ones or zeros at the beginning (forward mode) or end (backward mode) of the sequence, while very small excursions might indicate that ones and zeros are intermixed too evenly. A minimum sequence length of 100 bits is recommended.

F. Visual Test: Histogram of Byte Frequencies

Additionally, a visual test assesses byte sequence randomness by plotting a histogram of byte value frequencies (0-255). Ideally random data yields a flat, uniform histogram, as each byte value should occur with similar probability. Overlays like a normal distribution (using sample mean μ and standard deviation σ) and highlighting extreme frequencies help visualize this. Significant deviations from uniformity, such as pronounced peaks or valleys, suggest potential non-randomness in the sequence.

VII. Results

The statistical evaluation of the random number generators discussed here employs a selection of tests primarily based on the NIST SP 800-22 standard. The specific software implementation of these statistical tests utilized for this research is adapted from the open-source Python suite developed by Steven Kho Ang, an IT Professional. This foundational codebase is publicly available for reference on GitHub: https://github.com/stevenang/randomness_testsuite. The subsequent sections detail the outcomes and interpretations of these tests as applied to the evaluated random number generation methods.

For comparative analysis, the following results of different generators will be compared with a classical pseudo-random bit (CRNG) strings. These sequences were produced using Python's standard `random` module, where each bit was independently determined by the `random.choice(['0', '1'])` function to form strings of the required length.

*A. **Histogram***

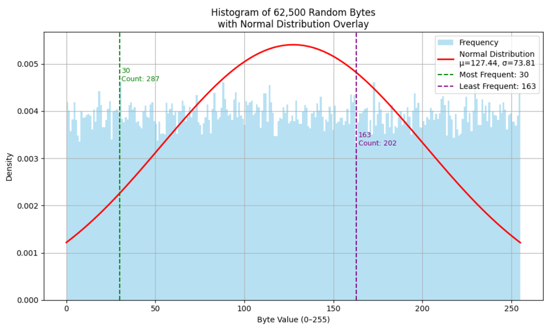

Figure 5. Histogram: CRNG

Table II
SUMMARY OF NIST SP 800-22 REV 1A TEST RESULTS FOR VARIOUS RNGS ($\alpha = 0.01$)

NIST Test	CRNG		CTR DRBG (no seq)		CTR DRBG (seq)		BEACON		QRNG	
	P-value	Result	P-value	Result	P-value	Result	P-value	Result	P-value	Result
Frequency (Monobit)	0.779467	Pass	0.671566	Pass	0.499041	Pass	0.852445	Pass	0.000000	Fail
Frequency within a Block (M=128)	0.133136	Pass	0.774426	Pass	0.600071	Pass	0.627412	Pass	0.888434	Pass
Runs Test	0.362484	Pass	0.821062	Pass	0.490481	Pass	0.723313	Pass	0.000000	Fail
Maurer's Universal (L=see note)	0.904546 (L=6)	Pass	0.212853 (L=7)	Pass	0.098342 (L=7)	Pass	0.799200 (L=7)	Pass	0.547755 (L=6)	Pass
Cumulative Sums (Forward)	0.725570	Pass	0.632885	Pass	0.708291	Pass	0.433978	Pass	0.000000	Fail
Cumulative Sums (Backward)	0.950381	Pass	0.970669	Pass	0.665229	Pass	0.585944	Pass	0.000000	Fail

Note: "Result" indicates Pass (P-value ≥ 0.01) or Fail (P-value < 0.01). M and L values for specific tests are based on implementation defaults or sequence length (M=128 for Block Frequency). For Maurer's Universal Test, L was determined by sequence length: L=6 for n=500,000 (CRNG, QRNG); L=7 for n=1,000,000 (CTR DRBG (no seq),CTR DRBG (seq), BEACON). CRNG and QRNG tested on 500,000-bit sequences. CTR DRBG (no seq), CTR DRBG (seq) and BEACON tests performed on a 1,000,000-bit segment

The histogram corresponding to 62,500 random bytes generated by the Classical Random Number Generator (CRNG) exhibits a distribution that appears reasonably uniform, indicating a satisfactory degree of randomness. The sample mean ($\mu = 127.44$) and standard deviation ($\sigma = 73.81$) are closely aligned with the theoretical values for a perfectly uniform distribution of bytes in the range 0–255, namely $\mu_{\text{th}} = 127.5$ and $\sigma_{\text{th}} \approx 73.90$.

The most frequently occurring byte appears 287 times, while the least frequent appears 202 times. These values are centered around the expected average count of approximately 244.14 (computed as $62500/256$). Taken together, these statistical properties support the conclusion that the CRNG produces high-quality random byte sequences.

B. Statistical Test Results for the CTR DRBG Implementation

The output sequence from the Counter Mode Deterministic Random Bit Generator (CTR DRBG was evaluated. While a total of 50,000,000 bits were generated, the statistical tests detailed herein were performed on a 1,000,000-bit segment, confirmed to be purely binary.

*1) **Frequency (Monobit) Test**:* This test assesses the overall balance of zeros and ones. The CTR DRBG yielded a P-value of 0.671566, resulting in a 'Pass' conclusion. This indicates that the overall proportion of zeros and ones in the tested sequence is statistically balanced and consistent with random data.

*2) **Frequency Test within a Block**:* This test examines the proportion of ones within M-bit blocks for local uniformity. The CTR DRBG sequence produced a P-value of 0.774426, leading to a 'Pass' status. This suggests that the distribution of ones and zeros is statistically uniform when examined across smaller, localized blocks (using M=128, the default from the implementation if not otherwise specified for this run).

*3) **Runs Test**:* The Runs Test evaluates the rate of oscillation between zeros and ones. The CTR DRBG achieved a P-value of 0.821062, corresponding to a 'Pass'. This implies that the number of runs of zeros and ones is as expected for a random sequence, indicating healthy oscillation patterns.

*4) **Maurer's "Universal Statistical" Test**:* This test assesses the sequence's compressibility. The CTR DRBG data resulted in a P-value of 0.212853, and a 'Pass' conclusion. This indicates that, according to this test, the sequence does not exhibit significant compressibility, a characteristic consistent with random data (using L=12, as determined by the implementation for a 50,000,000-bit sequence).

*5) **Cumulative Sums (Cusum) Test**:* This test checks for maximal excursions in a random walk derived from the sequence. For the forward test, the CTR DRBG yielded a P-value of 0.632885, resulting in a 'Pass'. Similarly, for the backward test, the P-value was 0.970669, also a 'Pass'. Collectively, these 'Pass' results suggest that the cumulative sum of the sequence does not drift excessively from zero in either direction, indicating no significant overall bias or trends in the data.

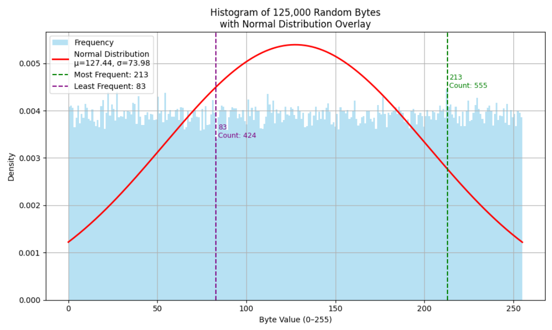

Figure 6. Histogram: CTR-DRBG

*6) **Histogram**:* This histogram represents 125,000 random bytes from the CTR generator. The visual distribution of byte frequencies appears largely uniform, which is indicative of a high degree of randomness. The sample mean extracted from the data is $\mu = 127.44$, and the sample standard deviation is $\sigma = 73.98$. These values align closely with the theoretical figures for a perfect uniform distribution over the 0-255 byte range ($\mu_{th} = 127.5$ and $\sigma_{th} \approx 73.90$). The most frequent byte value was observed 555 times, and the least frequent was observed 424 times. For this sample size, the expected average count per byte is approximately 488.28 $(125000/256)$. The proximity of the observed mean and standard deviation to the theoretical values, along with the reasonable concentration of extreme counts around the expected average, supports the conclusion that the CTR generator produces random bytes of good quality.

In summary, the statistical evaluation of this 1,000,000-bit sequence from the CTR DRBG implementation demonstrates excellent random characteristics across all five selected NIST tests. The sequence successfully passed tests for bit frequency (both global and local), run properties, compressibility, and cumulative sum trends, indicating a high degree of conformity with the expected properties of a random sequence at the chosen significance level.

C. *Statistical Test Results for the CTR DRBG (Sequential Seed) Implementation*

The output sequence from the Counter Mode Deterministic Random Bit Generator with sequential seeding (CTR DRBG (seq)) was evaluated. While a total of 400,000,000 bits were generated, the statistical tests detailed herein were performed on a 1,000,000-bit segment, confirmed to be purely binary.

*1) **Frequency (Monobit) Test**:* This test assesses the overall balance of zeros and ones. The CTR DRBG (seq) yielded a P-value of 0.671566, resulting in a 'Pass' conclusion. This indicates that the overall proportion of zeros and ones in the tested 1,000,000-bit sequence is statistically balanced and consistent with random data.

*2) **Frequency Test within a Block**:* This test examines the proportion of ones within M-bit blocks for local uniformity. The CTR DRBG (seq) sequence produced a P-value of 0.600071, leading to a 'Pass' status. This suggests that the distribution of ones and zeros is statistically uniform when examined across smaller, localized blocks (using M=`128`, the default from the implementation if not otherwise specified for this run).

*3) **Runs Test**:* The Runs Test evaluates the rate of oscillation between zeros and ones. The CTR DRBG (seq) achieved a P-value of 0.490481, corresponding to a 'Pass'. This implies that the number of runs of zeros and ones is as expected for a random sequence, indicating healthy oscillation patterns.

*4) **Maurer's "Universal Statistical" Test**:* This test assesses the sequence's compressibility. The CTR DRBG (seq) data resulted in a P-value of 0.098342, and a 'Pass' conclusion. This indicates that, according to this test, the 1,000,000-bit sequence does not exhibit significant compressibility, a characteristic consistent with random data (using L=`7`, as determined by the implementation for this sequence length).

*5) **Cumulative Sums (Cusum) Test**:* This test checks for maximal excursions in a random walk derived from the sequence. For the forward test, the CTR DRBG (seq) yielded a P-value of 0.708291, resulting in a 'Pass'. Similarly, for the backward test, the P-value was 0.665229, also a 'Pass'. Collectively, these 'Pass' results suggest that the cumulative sum of the sequence does not drift excessively from zero in either direction, indicating no significant overall bias or trends in the data.

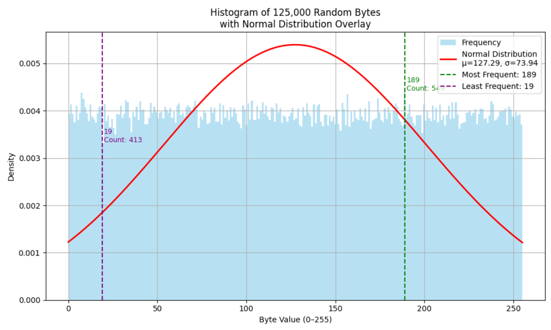

Figure 7. Histogram: CTR-DRBG(seq)

*6) **Histogram**:* This histogram represents 125,000 random bytes from the SEQ generator. The visual distribution of byte frequencies appears largely uniform, which is indicative of a high degree of randomness. The sample mean extracted from the data is $\mu = 127.29$, and the sample standard deviation is $\sigma = 73.94$. These values align closely with the theoretical figures for a perfect uniform distribution over the 0-255 byte range ($\mu_{th} = 127.5$ and $\sigma_{th} \approx 73.90$). The most frequent byte value was observed 541 times, and the least frequent was observed 413 times. For this sample size, the expected average count per byte is approximately 488.28 $(125000/256)$. The proximity of the observed mean and standard deviation to the theoretical values, along with the reasonable concentration of extreme counts around the expected average, supports the conclusion that the SEQ generator produces random bytes of good quality.

In summary, the statistical evaluation of the 1,000,000-bit segment from the CTR DRBG (seq) implementation demonstrates robust random characteristics across all five selected NIST tests. The sequence successfully passed tests for bit frequency (both global and local), run properties, compressibility, and cumulative sum trends, indicating a high degree of conformity with the expected properties of a random sequence at the chosen significance level.

D. *Statistical Test Results for the BEACON Generator*

The output sequence from the BEACON generator was evaluated using a 1,000,000-bit segment of entire sample of 8,681,984, confirmed to be purely binary.

*1) **Frequency (Monobit) Test**:* This test assesses the overall balance of zeros and ones. The BEACON generator yielded a P-value of 0.852445, resulting in a 'Pass' conclusion. This indicates that the overall proportion of zeros and ones in the tested sequence is statistically balanced.

*2) **Frequency Test within a Block**:* This test examines the proportion of ones within M-bit blocks for local uniformity. The BEACON sequence produced a P-value of 0.627412, leading to a 'Pass' status. This suggests that the distribution of ones and zeros is statistically uniform when examined across smaller, localized blocks (using M=128, the default from the implementation).

*3) **Runs Test**:* The Runs Test evaluates the rate of oscillation between zeros and ones. The BEACON generator achieved a P-value of 0.723313, corresponding to a 'Pass'. This implies that the number of runs of zeros and ones is as expected for a random sequence.

*4) **Maurer's "Universal Statistical" Test**:* This test assesses the sequence's compressibility. The BEACON data resulted in a P-value of 0.799200, and a 'Pass' conclusion. This indicates that, according to this test, the sequence does not exhibit significant compressibility, a characteristic consistent with random data (using L=7, as determined by the implementation for a 1,000,000-bit sequence).

*5) **Cumulative Sums (Cusum) Test**:* This test checks for maximal excursions in a random walk derived from the sequence. For the forward test, the BEACON generator yielded a P-value of 0.433978, resulting in a 'Pass'. Similarly, for the backward test, the P-value was 0.585944, also a 'Pass'. Collectively, these 'Pass' results suggest that the cumulative sum of the sequence does not drift excessively from zero in either direction, indicating no significant overall bias or trends in the data according to this specific test.

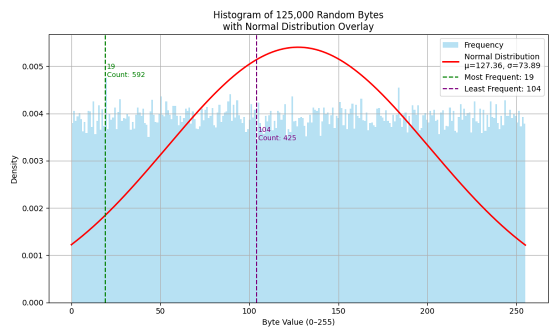

Figure 8. Histogram: BEACON

*6) **Histogram**:* This histogram represents 125,000 random bytes from the Beacon generator. The visual distribution of byte frequencies appears largely uniform, which is indicative of a high degree of randomness. The sample mean extracted from the data is $\mu = 127.36$, and the sample standard deviation is $\sigma = 73.89$. These values align closely with the theoretical figures for a perfect uniform distribution over the 0-255 byte range ($\mu_{th} = 127.5$ and $\sigma_{th} \approx 73.90$). The most frequent byte value was observed 592 times, and the least frequent was observed 425 times. For this sample size, the expected average count per byte is approximately 488.28 $(125000/256)$. The proximity of the observed mean and standard deviation to the theoretical values, along with the reasonable concentration of extreme counts around the expected average, supports the conclusion that the Beacon generator produces random bytes of good quality.

Based on these five selected NIST tests, the 1,000,000-bit segment from the BEACON generator passed tests for bit frequency (global and local), run properties, compressibility, and cumulative sum trends. Therefore the BEACON generator exhibits random characteristics.

7) Statistical Test Results for the Quantum RNG (QRNG): The output sequence from the quantum random number generator (QRNG), consisting of 500,000 bits confirmed to be purely binary, was evaluated using five selected statistical tests from the NIST SP 800-22 Rev 1a suite.

*8) **Frequency (Monobit) Test**:* This test assesses the overall balance of zeros and ones. The QRNG yielded a P-value of 0.000000, resulting in a 'Fail' conclusion. This indicates a critical imbalance in the global proportion of zeros and ones within the tested sequence.

*9) **Frequency Test within a Block**:* This test examines the proportion of ones within M-bit blocks for local uniformity. The QRNG sequence produced a P-value of 0.888434, leading to a 'Pass' status. This suggests that while a global imbalance exists, the distribution of ones and zeros appears statistically uniform when analyzed across smaller, localized blocks (e.g., using M=128).

*10) **Runs Test**:* The Runs Test evaluates the rate of oscillation between zeros and ones. The QRNG achieved a P-value of 0.000000, corresponding to a 'Fail'. This implies that the total number of runs of zeros and ones deviates significantly from what is expected for a random sequence, indicating problematic oscillation patterns.

*11) **Maurer's "Universal Statistical" Test**:* This test assesses the sequence's compressibility. The QRNG data resulted in a P-value of 0.547755, and a 'Pass' conclusion. This indicates that, according to this test, the sequence does not exhibit significant compressibility, a characteristic consistent with random data (using L=6, as determined by the implementation for a 500,000-bit sequence).

*12) **Cumulative Sums (Cusum) Test**:* This test checks for maximal excursions in a random walk derived from the sequence. For the forward test, the QRNG yielded a P-value of 0.000000, resulting in a 'Fail'. Similarly, for the backward test, the P-value was 0.000000, also a 'Fail'. These 'Fail' results across both modes strongly suggest that the cumulative sum of the sequence deviates excessively from zero, pointing to substantial trends or biases within the data.

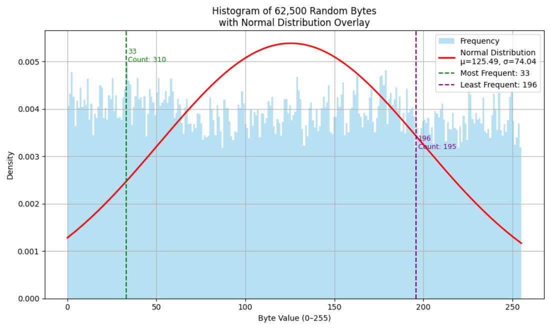

Figure 9. Histogram: QRNG

13) ***Histogram:*** The histogram displays data for 62,500 random bytes from the QRNG (Quantum Random Number Generator). The distribution of byte frequencies approximates uniformity, indicating some degree of randomness. The sample mean is $\mu = 125.49$, which is slightly lower than the theoretical mean of 127.5 for an ideal uniform distribution. The sample standard deviation is $\sigma = 74.04$, closely matching the theoretical standard deviation of ≈ 73.90. The most frequent byte occurred 310 times, while the least frequent byte occurred 195 times. The expected average count per byte value for this sample size is approximately 244.14 $(62500/256)$. While the standard deviation is good, the slight deviation in the mean and the spread of frequency counts (from 195 to 310) suggest that while generally random, the output might benefit from further statistical tests to confirm ideal uniformity.

In summary, the evaluation of this 500,000-bit sequence from the QRNG reveals significant statistical weaknesses. Despite passing tests for local frequency distribution (Block Frequency) and compressibility (Maurer's Universal), the QRNG critically fails the fundamental Frequency (Monobit) test, the Runs Test, and both Cumulative Sums tests. These failures point to considerable issues with overall bit bias, the nature of bit alternations, and pervasive trends within the generated sequence, suggesting notable deviations from ideal randomness.

VIII. Conclusions

The comparative analysis of various random number generators, including classical pseudo-random (CRNG), Counter Mode Deterministic Random Bit Generators (CTR DRBG and CTR DRBG with sequential seeding), and a Quantum Random Number Generator (QRNG), was conducted using a selection of five statistical tests from the NIST SP 800-22 Rev 1a suite. These tests, namely the Frequency (Monobit) Test, Frequency Test within a Block, Runs Test, Maurer's "Universal Statistical" Test, and the Cumulative Sums (Cusum) Test, proved invaluable in assessing different aspects of randomness, from basic bit balance to compressibility and trend analysis . The results indicated that both CTR DRBG implementations exhibited excellent statistical properties, passing all applied tests and demonstrating characteristics consistent with high-quality random data suitable for cryptographic applications. In contrast, the evaluated QRNG, despite its theoretical underpinnings for true randomness, showed significant deviations, failing critical tests such as the Monobit, Runs, and Cusum tests, suggesting underlying biases or issues in the generation or collection process that compromise its statistical randomness for the tested 500,000-bit sequence. The classical RNG served as a baseline and generally performed well.

References

[1] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators (revised). Technical Report Special Publication 800-90A Revision 1, National Institute of Standards and Technology (NIST), June 2015. URL https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf.

[2] J. Schiller D. Eastlake, S. Crocker. Recommendation for random number generation using deterministic random bit generators. Special Publication 800-90A Revision 1 SP 800-90A Rev. 1, National Institute of Standards and Technology (NIST), Gaithersburg, MD, 2015. URL https://doi.org/10.6028/NIST.SP.800-90Ar1.

[3] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. Technical Report Special Publication 800-38D, National Institute of Standards and Technology (NIST), November 2007. URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf.

[4] Gehad Salem Fekry. qrng: Quantum random number generator code repository. https://github.com/GehadSalemFekry/qrng/, 2023. Accessed: 2025-05-24.

[5] John Kelsey. Randomness beacon background and overview. Technical report, National Institute of Standards and Technology (NIST), 2018. URL https://csrc.nist.gov/publications/detail/nistir/8219/final.

[6] John Kelsey. Nist randomness beacon: Design and implementation. Technical report, National Institute of Standards and Technology (NIST), 2021. URL https://csrc.nist.gov/publications/detail/nistir/8380/final.

[7] John Kelsey, Lucas Brandão T. A., Rafael Peralta, and Joseph Booth. A proposal for a beacon design. Technical report, National Institute of Standards and Technology (NIST), 2019. URL https://csrc.nist.gov/CSRC/media/Projects/Random-Number-Generation/documents/beacon/NIST-Beacon-Proposal-v2.0.pdf.

[8] David Mechalas. Intel digital random number generator (drng) software implementation guide. https://software.intel.com/content/www/us/en/develop/articles/intel-digital-random-number-generator-drng-software-implementation-guide.html, 2018. Accessed: 2025-05-24.

[9] National Institute of Standards and Technology (NIST). Recommendation for random number generation using deterministic random bit generators. NIST Special Publication 800-90A Rev. 1, NIST, June 2015. URL https://doi.org/10.6028/NIST.SP.800-90Ar1. Revision 1.

[10] Yanbao Zhang, Hsin-Pin Lo, Alan Mink, Takuya Ikuta, Toshimori Honjo, Hiroki Takesue, and William J. Munro. A simple low-latency real-time certifiable quantum random number generator. *npj Quantum Information*, 7(1):71, 2021. doi: 10.1038/s41534-021-00433-z.