

Research Article

Design and analysis of an arduino-based low-cost true random number generator

Seyfullah Kaner^{1,*} , Ali Murat Garipcan²  and Ebubekir Erdem³ 

¹Muş Alparslan University, Department of Information Technologies, Muş, Turkey; s.kaner@alparslan.edu.tr

²Munzur University, Department of Computer Engineering, Tunceli, Turkey; alimuratgaripcan@munzur.edu.tr

³Firat University, Department of Computer Engineering, Elazığ, Turkey; aberdem@firat.edu.tr

*Corresponding Author

ARTICLE INFO

Keywords:

analog noise;
arduino;
cryptography;
randomness test;
true random number
generator (TRNG)

Dates:

Received: 16 Oct 2025

Accepted: 31 Jan 2026

ABSTRACT

This study covers the design and statistical randomness-based performance analysis of an Arduino-based True Random Number Generator (TRNG) that uses environmental analog noise as a randomness source. The environmental analog noise used as the randomness source was sampled from the Arduino's $A_0 - A_4$ pins, mixed at the bit level, and then subjected to the SHA-256 hash function in the post processing stage. The 256-bit random number blocks obtained using this method were transferred via the serial port and converted into a format suitable for NIST SP-800-22 and Dieharder tests. In terms of statistical randomness, the random numbers achieved successful results in all NIST tests and in 93.94% of the Dieharder tests. However, Shannon entropy and bias analyses showed that the generated numbers did not carry structural patterns. The study demonstrates that reliable randomness can be obtained even under limited hardware and low energy conditions, showing that the developed structure provides an accessible and reproducible platform for educational use, embedded system security, and lightweight cryptographic applications.

1. Introduction

Today, applications such as information security, cryptography, authentication, digital signature systems, and random key generation are increasing the need for high-quality random number generation. Randomness has become a fundamental requirement not only in the field of security but also in disciplines such as scientific modeling, Monte Carlo simulations, biomedical imaging, and artificial intelligence in terms of statistical accuracy and diversity. The unpredictability and high entropy of random numbers used in cryptographic applications, in particular, are critical for minimizing attack surfaces [1]–[3].

To address this need, two main approaches stand out in the literature: Pseudorandom Number Generators (PRNGs) and True Random Number Generators (TRNGs). PRNGs use mathematical algorithms to generate seemingly random sequences of numbers based on a specific seed. They are widely preferred in many software applications due to their high processing speeds and low resource consumption. However, due to their deterministic nature, if the initial seed is compromised, the generated sequences become predictable. This situation can lead to serious security vulnerabilities, especially in systems with cryptographic security requirements [4], [5].

TRNGs, on the other hand, provide true randomness by generating digital bit sequences from analog signals obtained from physical events (e.g., thermal noise, radioactive decay, photon collisions). This

approach makes physical processes, which are inherently unpredictable, usable in the context of information security [6], [7]. However, TRNG systems have remained limited in widespread use because they typically require specialized hardware, complex electronic circuits, or expensive sensors [8], [9]. Therefore, TRNG designs that can be implemented on low-cost hardware platforms are critically important for both academic research and engineering applications. Solutions developed on open-source microcontroller systems, in particular, offer advantages in terms of accessibility and scalability. In this context, the fundamental differences between the structural characteristics and security performance of PRNG and TRNG systems are summarized in Table 1.

Table 1. Comparison of the basic features of PRNG and TRNG systems

Feature	PRNG (Pseudo-Random Number Generator)	TRNG (True Random Number Generator)
Source of randomness	Mathematical / deterministic algorithms	Physical phenomena (noise, thermal fluctuations, etc.)
Predictability	Present (if the seed is known)	Absent
Output repeatability	Same seed → same output	Different output on each run
Hardware requirements	Generally none	Requires physical sensors or dedicated circuitry
Speed	High	Relatively slower
Suitability for security	Medium–Low (non-cryptographic uses)	High (especially for security applications)
Typical use cases	Simulation, gaming, testing	Cryptography, key generation, authentication
Entropy level	Limited (due to deterministic nature)	High (sourced from physical randomness)
Power consumption	Low (CPU only)	Medium–High (additional hardware or sensors required)
Cost	Very low (existing hardware usually sufficient)	Higher (may require extra circuits / ASIC / FPGA)
Vulnerability to attacks	Susceptible to software attacks (seed recovery, algorithm reverse engineering)	Vulnerable to hardware tampering or side-channel attacks

The fact that PRNGs are based on deterministic algorithms and have limitations that restrict confidentiality, such as reproducibility depending on the initial conditions, is the main reason for preferring hardware-based TRNGs. Hardware-based PRNGs use unpredictable physical processes such

as thermal noise, phase jitter, metastability, or chaotic oscillators in electronic circuits as entropy sources. This ensures that the generated bits are algorithmically unpredictable, providing the high level of security required for cryptographic applications. A typical hardware TRNG structure consists of an entropy source, a preprocessing/bit scrambling block, and post-processing stages. Thanks to this three-layer structure, the random number generation process offers a much stronger security infrastructure than software solutions, based on both physical unpredictability and statistical reliability [10]–[12].

Within the scope of this study, the Arduino Uno platform was used for physical noise-based true random number generation. Analog environmental electrical noise signals were collected via Arduino's analog pins and digitized using the ADC unit. In the post processing stage, the digitized data was processed using bit mixing techniques and the SHA-256 cryptographic hash function to produce fixed-length random bit sequences. Cryptographic hash functions such as SHA-256 play a critical role in the post processing stage due to their ability to convert low-level irregular data into homogeneous and high-entropy outputs [13].

The main advantage of the developed system is that it enables TRNG to be performed using only an Arduino board and a computer, without the need for any external circuitry. This makes the system both economical and accessible, while also offering the flexibility provided by open-source platforms. However, some studies on the randomness quality of embedded systems such as Arduino have revealed that these platforms may have limited entropy sources and may contain pattern repetitions [14], [15]. Therefore, techniques such as XOR mixing and hashing algorithms used in analog data processing have been systematically designed to improve the quality of randomness.

The generated random bit sequences have been analyzed using widely accepted statistical test suites such as NIST SP800-22 and Dieharder. These tests are used to evaluate the suitability of the bit sequence in terms of criteria such as statistical independence, distribution, and unpredictability. Since true randomness is measured not only by irregularity but also by long-term statistical consistency, the successful results obtained from these tests reinforce the validity of the system. Figure 1 shows the general flow diagram of the developed system, covering all stages from data collection via Arduino to the statistical testing process.

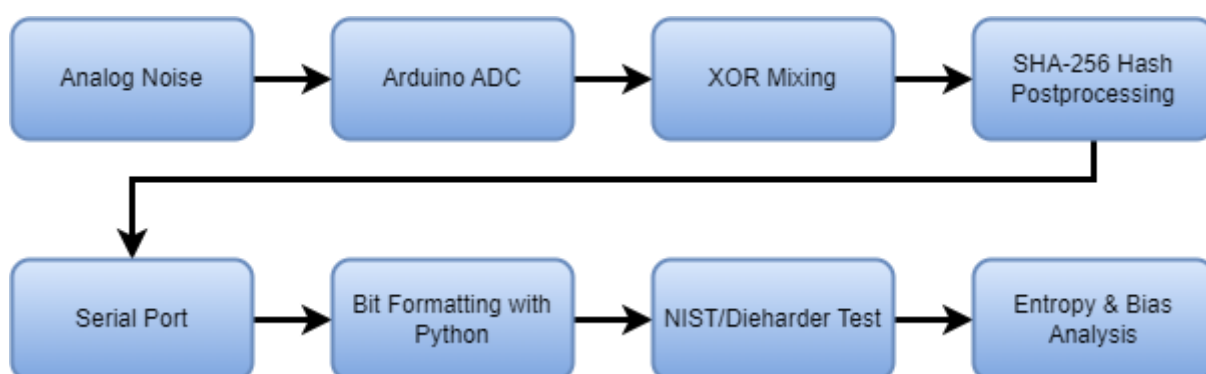


Figure 1. Data acquisition, processing, and analysis via Arduino

In general, academic studies on TRNG systems mostly address quantum-based or complex systems with specialized hardware [16], [17]. However, the system proposed in this study stands out because it

is implemented on low-cost and widely accessible hardware such as Arduino. The accessibility of the hardware increases the scalability of the system, while the software architecture, supported by summarization and mixing algorithms, ensures statistical quality. As a result, the main contribution of this study is the presentation of a low-cost and reproducible TRNG solution that holistically combines the stages of analog noise sampling, entropy enhancement, and cryptographic hashing. The results obtained prove the statistical validity of the system and provide a solid foundation for randomness-based applications that can be implemented on similar hardware.

1.1. Literature Review and Problem Definition

True random number generation has become the focus of applied research in recent years, not only at the theoretical level but also at the hardware and software levels. In the literature, PRNG systems are generally based on high-entropy physical sources and are mostly developed using specialized circuit designs or quantum-based mechanisms [7], [18]. However, the cost and hardware complexity of such systems limit their use in widespread applications. This situation has necessitated a focus on simple but effective PRNG architectures, especially in microcontroller-based systems.

In this context, low-cost microcontrollers such as Arduino have become an attractive platform for TRNG research due to their accessibility and portability advantages, despite their limited entropy sources. Kristinsson demonstrated that the noise obtained from Arduino's analog pins is insufficient for direct use; however, acceptable levels of randomness can be achieved with simple processing techniques [14]. Similarly, Kuka and colleagues developed a TRNG system operating via near-field communication, focusing on increasing the entropy density of low-quality data from analog sources [9]. Processing ADC (Analog-to-Digital Converter) outputs with hash functions is a fundamental approach to achieving statistical homogeneity at the bit level in such systems [19].

In addition to these approaches, several studies have focused on developing efficient post-processing algorithms and hybrid architectures to improve entropy quality and computational performance. Yakut et al. [20] proposed a secure and efficient hybrid random number generator based on sponge constructions, demonstrating significant improvements in throughput and statistical stability for cryptographic applications. In a subsequent study, Yakut et al. [21] presented a novel TRNG framework integrating lossy picture compression and discrete cosine transform (DCT) to enhance randomness extraction efficiency. These works highlight the importance of combining mathematical post-processing and hardware-level entropy collection to achieve high-quality randomness even in constrained environments.

FPGA-based designs have attracted significant interest due to their flexibility and reprogrammability. For example, Yüksel et al. [22] developed a high-performance TRNG named TuRaN using a voltage-dropping error-induction method in SRAM cells, reporting speeds reaching Gbps. This method has been demonstrated practically on an FPGA and successfully passed NIST tests. Similarly, Olgun et al. [23] produced a high-speed (3.44 Gbps) TRNG by simultaneously activating four rows in DDR4 DRAM chips in their work titled QUAC-TRNG, demonstrating that this method is also applicable on FPGAs. These studies demonstrate that FPGAs are not only a flexible development environment but also a suitable platform for integrating memory-based TRNGs.

ASIC-based solutions, on the other hand, stand out in terms of performance and energy efficiency. Ghandali et al. [24] examined the security vulnerabilities of ring oscillator-based ASIC-TRNG architectures and demonstrated their susceptibility to hardware Trojan attacks. This finding highlights that security-focused verification, not just randomness, is critical in TRNG designs. Furthermore, Ng et al. [25] showed that magnetic tunnel junction (MTJ)-based TRNGs offer smaller area and lower power consumption in ASIC designs compared to CMOS-based methods. In another study, spintronics-based TRNG designs are emerging as a new research area. Shukla et al. [26] developed a low-power TRNG (~100 fJ/bit) using medium-barrier MTJs. Fu et al. [27] proposed a fast TRNG based on STT-MTJs and produced reliable randomness at a speed of 303 Mb/s. Such nanoelectronics-based solutions are being considered as alternatives to ASIC and FPGA-based security hardware in the future.

Memory-based TRNGs have also gained significant attention in the literature. Bostancı et al. [28] proposed the DR-STRaNGe system design, enabling DRAM-based TRNGs to provide secure entropy without affecting application performance. Hybrid solutions are also prominent in the field of hardware security. Cazzola et al. [29] presented an architecture that combines a powerful PUF (Physical Unclonable Function) with a TRNG on an FPGA, thus providing both authentication and randomness generation in a single structure. These hybrid approaches demonstrate that low-cost FPGA platforms are suitable for security-focused, multi-functional designs.

In recent years, significant work has been done on the reliability of ring oscillator-based TRNGs. Pebay-Peyroula et al. [30] shared the open-source design of ring oscillator-based TRNGs through their OpenTRNG initiative. This work contributed to standardization efforts in the academic community by offering a flexible and reusable TRNG architecture that can be hardware-verified on different FPGA and ASIC platforms. In another study, the role of metastability effects in TRNG designs on security was also investigated. Pebay-Peyroula et al. [31] examined the impact of metastability issues on entropy quality in ring oscillator jitter-based TRNGs. Tests conducted on the Xilinx Artix-7 FPGA demonstrated that metastability-induced corruption can be controlled with appropriate design measures.

Scalable TRNG architectures with low hardware footprint are also prominent in the literature. Yi et al. [32] designed a TRNG based on autonomous Boolean networks using unbalanced node oscillation loops. Experiments conducted on the Xilinx A7 FPGA demonstrated that reliable randomness can be achieved with low hardware overhead. Finally, compact solutions combining PUF and TRNG are important for IoT and embedded systems. Dang et al. [33], presented a ring generator-based combined PUF-TRNG design in their published work. This method, verified on 180 nm CMOS, provides a scalable solution for cryptographic applications with both compact hardware footprint and high randomness quality.

In summary, a review of the literature shows significant progress in terms of diversity and speed in FPGA and ASIC-based TRNG designs. While FPGA-based approaches offer flexibility and experimental prototyping advantages, ASIC-based solutions stand out in terms of energy efficiency and integrated security. Memory-based and spintronic-based TRNGs, on the other hand, open the door to next-generation hardware security solutions. In this context, the low-cost Arduino-based TRNG approach can be considered a more accessible and reproducible alternative to the high-cost FPGA/ASIC solutions in the literature.

1.2. Motivation and Contribution

TRNGs are critical in cryptography, authentication, and security protocols. However, existing solutions typically rely on high-cost FPGA/ASIC-based hardware or specialized sensors, limiting accessibility, especially in educational environments and low-resource embedded systems. Reliable random number generation on devices with limited processing power and low energy remains an open problem in the literature. Solutions that holistically address entropy enhancement and cryptographic post-processing steps on low-cost and widely accessible microcontrollers such as Arduino are quite limited. The main motivation of this work is to develop an economical and reproducible system that can provide statistically reliable randomness even under constrained hardware and low-power conditions. The main contributions of this work can be summarized as follows:

Hardware-based true random number generation has been achieved on low-cost embedded systems with limited processing power, without the need for external hardware.

The randomness performance of the proposed system has been comprehensively evaluated using NIST SP800-22 and Dieharder test suites, demonstrating its statistical reliability.

The proposed system demonstrates that reliable random number generation is possible on low-cost hardware, offering an accessible and reproducible alternative to existing expensive hardware-based solutions.

1.3 Organization of the Work

The remainder of the study is organized as follows: The second section summarizes the theoretical background related to random number generation, the concept of entropy, and the test standards used. The third section details the architecture, hardware infrastructure, and software process of the proposed Arduino-based system. The fourth section presents the performance analysis results of the obtained data, evaluating the NIST SP800-22 and Dieharder test results. The fifth and final section concludes the study by presenting the general findings and future work.

2. Theoretical Background

This section covers the theoretical foundations of hardware-based random number generation. First, the physical principles of TRNGs are explained, followed by an examination of the concept of entropy, analog noise sources, and the use of these sources in digital systems. In addition, statistical test standards commonly used to measure the reliability of TRNG outputs are introduced, and summary information on entropy calculation methods is provided.

2.1 True Random Number Generators

Random number generation is one of the fundamental components of modern digital security systems. High-entropy and unpredictable random number sequences are required, particularly in applications such as cryptographic key generation, authentication, nonce generation, digital signatures, and protocol verification [2]. The specialized design components used to generate these numbers, also known as random number generators, are primarily divided into two classes.

PRNGs are based on deterministic algorithms, and all outputs can be reproduced when the initial seed value is known. Therefore, the repetition of internal states, i.e., periodicity, is another significant limitation that restricts the use of PRNGs in sensitive cryptographic applications. TRNGs, on the other hand, use unpredictable entropy sources from physical systems to generate non-reproducible bit sequences [6], [34]. TRNGs typically rely on entropy obtained from physical processes in nature (e.g., thermal noise, phase jitter, radioactive decay). Sources such as voltage changes read from analog sensors are converted into digital bit sequences through high-frequency sampling. However, using the raw data from these sources is often insufficient; it must be converted into a stable format using post-processing algorithms [35]. TRNGs work by digitizing analog signals obtained directly from physical events. Therefore, the system consists of three main components, as shown in Figure 2:

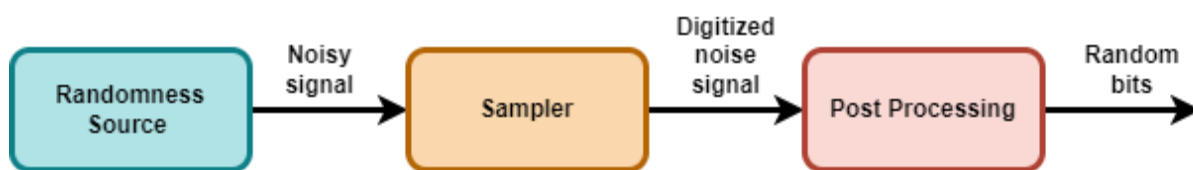


Figure 2. TRNG hierarchical design components

Physical Entropy Source (Analog Signal): One of the most common physical sources used in TRNG systems is thermal noise. This refers to random voltage fluctuations occurring on a resistor that depend on temperature. The mean square value of the signal is expressed as in equation 1:

$$V_{rms} = \sqrt{4kTR\Delta f} \quad (1)$$

where k is the Boltzmann constant, T is the temperature (Kelvin), R is the resistance (Ohm), and Δf represents the bandwidth. These voltage fluctuations are transmitted to the microcontroller via analog input pins and converted into numerical values using a 10-bit ADC in systems such as Arduino Uno. Samples obtained from the ADC are typically concentrated in the range 0 – 1023. Due to the nature of analog signals, these values are distributed in a manner similar to a Gaussian distribution. The statistical properties of the numerical data obtained from sampling can be calculated as shown in equation 2:

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i, \sigma^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2 \quad (2)$$

Here x_i , where μ is the average value measured by the ADC. The greater the variability in the thermal noise level, the larger the variance (σ^2), and the greater the potential entropy.

Analog-to-Digital Converter (ADC) Layer: The ADC module converts analog signals into digital data at a specific resolution by sampling them. In low-cost systems such as Arduino Uno, each sample is 10-bit wide. This means that an analog measurement corresponds to 1024 different digital values. However, data from the ADC may contain patterns. Therefore, it is inadvisable to use them directly in cryptographic applications.

Post-processing: Raw data obtained from the ADC undergoes a series of post-processing steps at the software layer. These steps are applied to improve the statistical structure of the data and enhance its security. First, bit-level scrambling techniques (e.g., XOR, shifting, and rotation) are used to mask patterns in the data and reduce structural dependencies. Finally, the data is converted into a fixed-length, high-entropy output using one-way cryptographic hash functions such as SHA-256. This process ensures high-quality and secure outputs, even with low-entropy inputs. Algorithms such as SHA-256 eliminate deterministic structures in the input data, increasing the statistical homogeneity of the output and making it resistant to attacks [7].

The system, whose detailed structure is shown in Figure 1, provides reliable, non-reproducible, and statistically robust randomness generation at the microcontroller level. Furthermore, the system's closed nature allows it to exhibit high resistance to external interference. However, such systems also have some disadvantages. Factors such as sensitivity to environmental factors, e.g., temperature changes and voltage fluctuations, limited entropy sources due to hardware constraints, and the direct impact of the ADC's sampling rate and resolution on randomness quality are issues that must be carefully considered in system design and implementation.

2.2 Entropy and Analog Noise-Based Approaches

The most important element forming the basis of TRNG is the physical entropy source possessed by the system. This source produces analog signals that are observable in a form transferable to the digital environment and that provide the system's unpredictability. Random physical processes such as thermal noise, zener diodes, jitter, atmospheric interference, or ADC instabilities are considered suitable basic components for entropy generation. In microcontroller-based systems such as Arduino Uno, these signals are typically sampled via analog input pins and converted to digital values using a 10-bit resolution ADC module.

The digital data obtained through the ADC are numbers that, under ideal conditions, are concentrated in the 0 – 1023 range and exhibit a Gaussian-like distribution. However, it cannot be said that these values are reliable in terms of randomness. Due to the weak variance of the analog signal, ADC resolution, and environmental effects, the generated bit sequences often contain patterns, particularly the tendency for the most significant bits to remain constant. This makes the direct use of ADC data insufficient for statistical testing. For true randomness, not only must source entropy exist, but this entropy must also be effectively extracted and processed in a digital environment without being corrupted.

At this point, Shannon entropy is at the forefront of the criteria used to evaluate entropy quality. According to Shannon, the level of randomness of a bit sequence is defined based on the probability distribution of the bits in that sequence. When both bit values occur with equal probability (0.5), entropy theoretically reaches its maximum and is calculated as in equation 3:

$$H(X) = - \sum_{i=1}^n p_i \log_2 p_i \quad (3)$$

Here, p_i is the probability of the corresponding bit value (0 or 1) appearing in the sequence. As the entropy value approaches 1, the quality of randomness increases. Additionally, bias analysis, particularly in digital systems, is used to measure the imbalance in the distribution of 0 and 1 bits in the generated sequence. The closer the bias value is to 0, the more balanced the sequence is considered, and thus the higher its entropy. Bias is calculated as in equation 4:

$$\text{Bias} = |P(1) - 0,5| \quad (4)$$

On the other hand, minimum entropy (min-entropy) is also accepted as an evaluation criterion, complementing entropy measurement. Minimum entropy is defined based on the probability of repetition of the most frequently occurring bit value in the sequence and represents the level of randomness in the worst-case scenario for the system. Standards such as NIST SP 800-90B particularly emphasize this metric [19].

The process of converting an analog signal into a random bit sequence often encounters the following problems: signal centering, ADC sampling producing repeated values due to low variance, and signal characteristics changing over time depending on environmental conditions. To overcome these challenges and convert the entropy in the signal into secure digital sequences, software processes called post-processing are applied. Bit-level mixing, such as XOR operations and cyclic shifts, followed by cryptographic hashing algorithms like SHA-256, achieves strong randomness by diluting entropy and breaking systematic patterns.

The successful implementation of these steps depends not only on the statistical analysis of analog signals but also on the system's ADC architecture, sampling frequency, susceptibility to noise, and software processing steps. Therefore, the concept of entropy is an interdisciplinary engineering problem that must be addressed at both the hardware and software levels.

2.3 Randomness Test Suites and Standards

Basic statistical measures such as entropy or bias alone are not sufficient to evaluate the accuracy and reliability of PRNGs. The statistical structure and unpredictability of randomness over long sequences can only be evaluated with multidimensional and comprehensive test suites. For this purpose, two fundamental test standards that are widely accepted in the field of cryptography and information security and frequently used in the literature stand out: NIST SP 800-22 and Dieharder test suites.

NIST SP 800-22, published by the National Institute of Standards and Technology (NIST), was developed to evaluate random bit sequences used in cryptographic applications. This test suite includes 15 different tests. These tests examine the statistical properties of the output to evaluate its predictability or whether it contains patterns [36]. The parameter sets for the test criteria and the metric values for sequence lengths are provided in Table 2.

Table 2. Probability distributions and input sizes of NIST 800-22 test criteria

Test No	Test Name	Reference Distribution	Size
1	Frequency test	<i>Half normal</i>	$n \geq 100$
2	Frequency test within a block	χ^2 (<i>chi – square</i>)	$n \geq 100$
3	Run test	χ^2	$n \geq 100$
4	Test for the longest run of ones in a block	χ^2	$n \geq 128 - 750,000$
5	Binary matrix rank	χ^2	$n \geq 38,912$
6	Discrete Fourier transform	<i>Normal</i>	$n \geq 100$
7	Non-overlapping template matching	χ^2	$n \geq 10^6$
8	The overlapping template matching test	χ^2	$n \geq 10^6$
9	Maurer's universal statistical test	<i>Half normal</i>	$n \geq 387,340 - 109$
10	Linear complexity test	χ^2	$n \geq 10^6$
11	Serial test	χ^2	$m \leq \log_2(n) - 2$
12	Approximate entropy test	χ^2	$m \leq \log_2(n) - 2$
13	Cumulative sums	<i>Normal</i>	$n \geq 100$
14	Random Excursions	χ^2	$n \geq 10^6$
15	Random Excursions Variant	<i>Half normal</i>	$n \geq 10^6$

The result obtained for each test is expressed as a p-value. This value is used to determine whether the sequence is random in terms of its relevant statistical property. An acceptable p-value should generally fall within the range of 0.01 to 0.99 and is expressed as in equation 5:

$$0.01 \leq p \leq 0.99 \quad (5)$$

The Dieharder Test Suite is an expanded and modernized version of the classic Diehard tests developed by George Marsaglia [37]. It contains more tests than NIST and is generally considered more rigorous. Some tests are based on Monte Carlo Pi calculations, direct number frequency analysis, or matrix rank. Dieharder is a powerful analysis tool, particularly prone to failure with low-entropy data sources [38].

Table 3. Classification of Statistical Randomness Tests in the Dieharder Suite

Test Group	Sample Tests	Description
Distribution Tests	Birthdays, Rank, Bitstream, OPSO, OQSO	Tests the distribution properties of random numbers in various dimensions.
Classic Probability Tests	Squeeze, Parking Lot, Runs, Craps	Performs probability-based systematic analyses.
High Complexity Tests	RGB Lagged Sum, Minimum Distance	Tests the stability of the data structure.

This open-source test suite, widely used on Linux-based systems, offers more test types and more flexible configurations. The Dieharder test is particularly suitable for evaluating random number generators with high sample volumes and covers the specific analyses summarized in Table 3.

As in NIST tests, the p-value parameter is used to evaluate the test statistic in the Dieharder test. However, the nature of some tests may cause them to be reported as weak or failed, which may be due to the high sensitivity of the test. For this reason, Dieharder is generally considered a challenging testing process in the literature. To obtain reliable results, Dieharder tests require very large data sets (often at the gigabit level). Using an insufficient data set may increase the failure rate. Such failures may not always be due to low entropy but may sometimes be due to the test's excessive sensitivity.

Since both test suites evaluate different aspects, it is not sufficient for a system to pass only NIST or only Dieharder tests. TRNG systems must pass such tests successfully, both individually and collectively, to prove their reliability. In this study, the outputs of the developed system were subjected to both NIST SP800-22 and Dieharder tests; analysis was performed on files converted to match the test formats. This allowed for a multifaceted evaluation of the system's statistical randomness quality.

2.4 Post Processing Stage with SHA-256

The direct use of data obtained from physical entropy sources in true random number generation often fails to provide an adequate level of security due to the presence of statistical patterns and bias effects. Therefore, processing analog signals to convert them into bit sequences with higher entropy and unpredictability is a critical step. This stage, referred to in the literature as the post processing stage, encompasses techniques such as hashing, bit-level mixing, and whitening. In this study, the SHA-256 algorithm was chosen for the post processing step. Standardized by NIST and widely used in cryptographic applications, this algorithm was selected for its reliability and ease of integration [19], [39].

SHA-256 is a one-way function that converts any length of input into a fixed 256-bit output sequence. Thanks to this feature, it can produce more homogeneous and unpredictable bit sequences from physical data that may contain low entropy and systematic patterns. The fundamental function of the algorithm is to ensure that even the smallest change in the input causes significant differences in the output, thanks to a feature known in the literature as the avalanche effect. This allows micro-level changes observed in analog signals to directly contribute to randomness generation.

The basic processing steps of the SHA-256 algorithm are presented visually in Figure 3. The algorithm begins by dividing a 512-bit input block into sixteen 32-bit words (W_0 – W_{15}). These initial words are expanded via the Message Expander (ME) module, resulting in a total of 64 words (W_0 – W_{63}). This expanded word sequence is then passed to the Message Compressor (MC) module. The MC process operates in a loop consisting of rounds, updating the intermediate digest values in each round. It works by blending the compression functions with four Boolean functions (Ch , Maj , Σ_0 , Σ_1) and specific constants (round constants). This structure effectively eliminates patterns in the input space. In each iteration, the compression process is performed using values from the previous round and the expanded message words. At the end of this process, the algorithm produces a fixed-length 256-bit digest value [40].

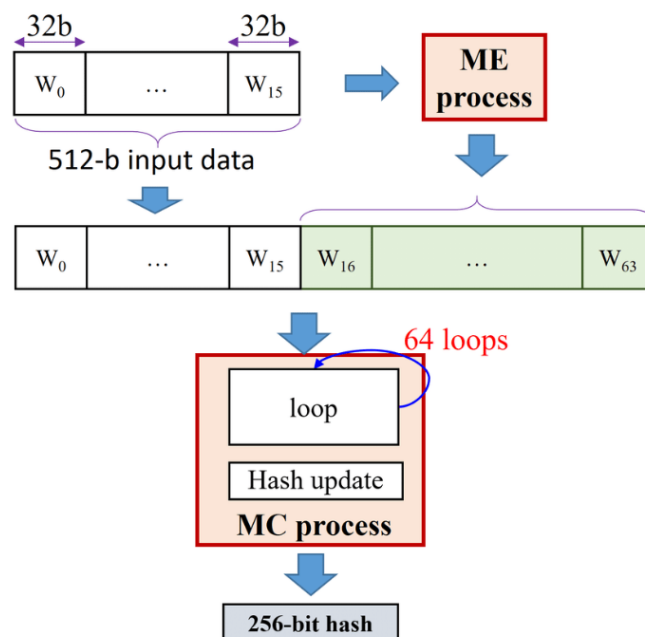


Figure 3. General operation of the SHA-256 algorithm

The SHA-256 function is structured as shown in equation 6:

$$SHA - 256(x) \rightarrow h \in \{0,1\}^{256} \quad (6)$$

Here, x represents the input data, and h represents the 256-bit-long summary output. The SHA-256 function has the ability to balance bit sequences with low variance and bias. This feature provides a significant advantage, especially in ADC-based systems where entropy is limited. The application results in a noticeable increase in the minimum entropy value. Minimum entropy ($H_{\min}$) is calculated based on the probability of the most frequently occurring value in a sequence, as shown in equation 7:

$$H_{\min} = -\log_2 \left(\max_{b \in \{0,1\}} \Pr[b] \right) \quad (7)$$

With the implementation of SHA-256, the minimum entropy value increases and the success rate in randomness tests rises. The reason for choosing this algorithm is not only its widespread use and security. It is also because it can be easily integrated into hardware-based systems at a high speed. Although alternative algorithms, such as SHA-3/Keccak, BLAKE2, or SipHash, offer more modern designs, the applicability of SHA-256 is more practical in microcontrollers with limited resources, such as Arduino [41]. Furthermore, SHA-256's compliance with the FIPS 180-4 standard provides an additional advantage in terms of reliability for public security and military applications.

3. System Architecture and Implementation Methodology

This section details the hardware infrastructure, software architecture, data processing workflow, and preparation of outputs for testing of the proposed random number generation system. The system

was implemented on a low-cost hardware platform based on Arduino Uno; data collection, processing, and preparation for statistical testing were facilitated through a software infrastructure supported by the Python programming language. The performance of the system varies depending on the amount of data used, the sampling frequency, and the structure of the processed data; therefore, these parameters must be carefully defined. The setup configuration based on these parameters is shown in Figure 4.

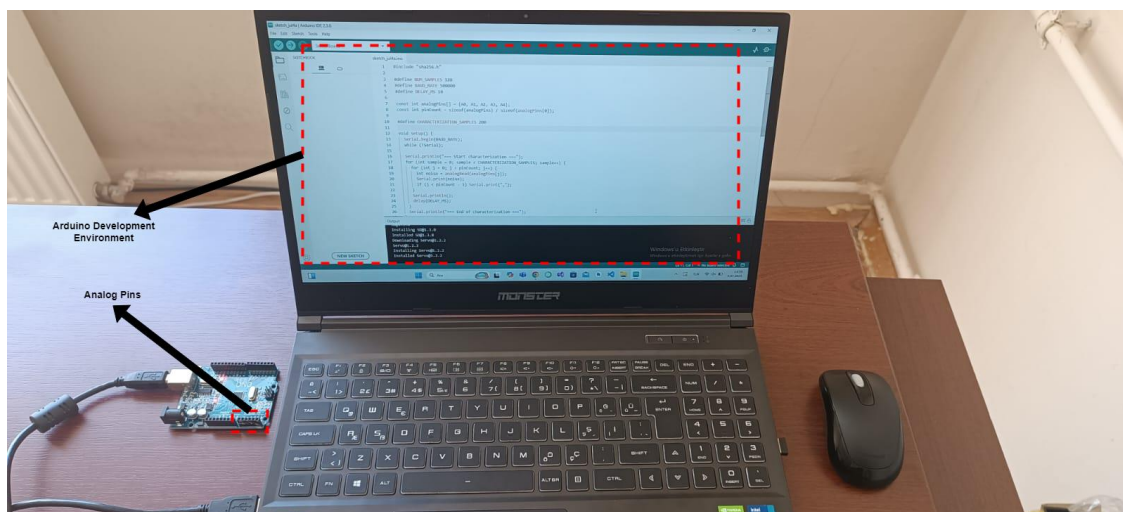


Figure 4. Setup configuration

3.1. Hardware Infrastructure and Entropy Source Design

For the purpose of generating true random numbers, the Arduino Uno (ATmega328P), an open-source and economical microcontroller platform, was chosen. In this study, the hardware components of the system were limited as follows:

- Arduino Uno (ATmega328P processor)
- Power supply and data transfer via USB connection
- Analog input pins ($A_0 - A_4$): sources where environmental electromagnetic noise is passively collected

No external sensor, oscillator, or module was used; the hardware consists solely of the elements listed above. The Arduino's built-in 10-bit ADC module ($A_0 - A_4$) was configured to detect environmental noise through the pins. The setup requires no soldering, resistors, or filters; this simple design ensures the system is an accessible application example for low-resource devices and educational environments. Since the USB connection handles both power and data transfer functions, no additional power circuits or communication modules are required. The non-deterministic (stochastic) nature of the signals read from the analog inputs forms the system's fundamental entropy source. Reliably digitizing these physical signals, determining the sampling interval, and applying the appropriate pre-processing technique are the first and critical steps in the random number generation process.

The experiments were conducted in a controlled laboratory environment at room temperature (approximately 23°C), using a standard USB-powered Arduino Uno connected to a desktop computer. The analog noise sources were naturally occurring environmental electromagnetic interferences

collected via the analog input pins ($A_0 - A_4$) without any external sensors or amplifiers. The laboratory environment included common electronic equipment (monitors, computers, and power supplies), which provided a realistic level of background noise. This setting was intentionally selected to demonstrate that the proposed TRNG design can operate reliably under typical conditions without requiring a shielded or noise-isolated setup, thereby ensuring reproducibility for other researchers using similar low-cost hardware.

3.2. Analog Data Transmission and Conversion Processes

The embedded software running on Arduino cyclically collects analog samples from the pins ($A_0 - A_4$). The data obtained in each sampling cycle is processed at the bit level and passed through specific mixing and preprocessing steps to increase entropy density before being applied to the SHA-256 cryptographic hash function. One SHA-256 digest is generated from every 128 analog sampling groups, yielding outputs 256 bits (32 bytes) in length. The process of digitizing analog signals from the Arduino, increasing entropy, digesting them with the SHA-256 algorithm, and transferring them to the computer is explained step by step. Figure 5 shows this software process, from the Arduino to the data formatting on the Python side.

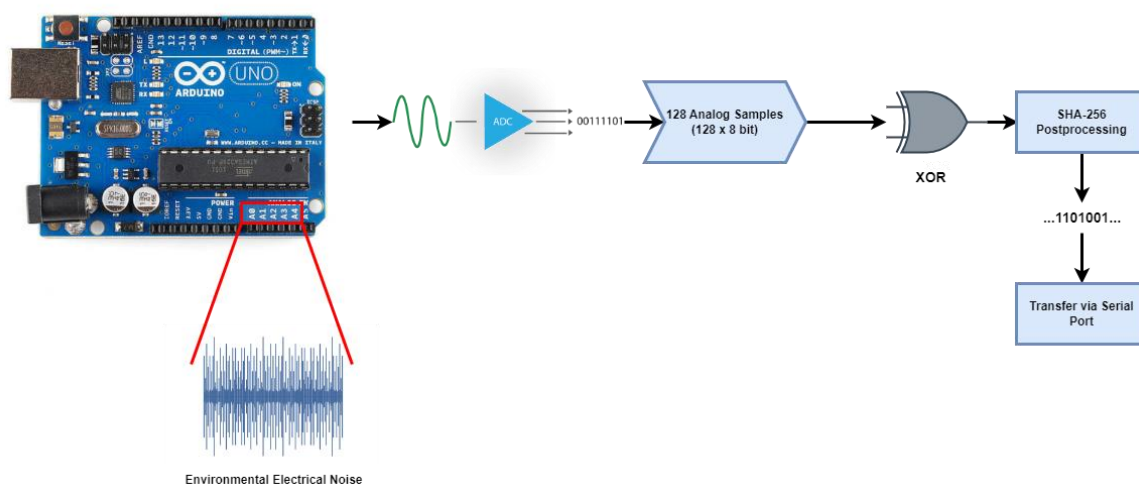


Figure 5. Bit sequence generation process from analog pins

In the data processing stage of the system, the numerical data obtained from the Arduino's analog input pins was collected within a specific sampling cycle and subjected to various bit-level mixing operations. These operations include XOR-based mixing and bitwise rotate techniques. The purpose of the mixing process is to weaken the patterns of the raw data obtained hardware-wise and increase the entropy level. After each analog sampling (128) is completed, the resulting temporary bit sequence is fed into the SHA-256 algorithm, yielding fixed-length (256-bit) summary data. SHA-256 was used as the post processing step in this system and plays an important role in increasing the randomness quality of the data obtained from the physical entropy source. The working principle of SHA-256 and its effects on randomness are presented in detail in Section 2.4 within the theoretical background.

The embedded software running on Arduino packages these 256-bit summary outputs in hexadecimal format and transfers them to the computer via the serial port. Each transferred data block is marked with specific characters (#...\$) at the beginning and end, and is safely parsed by the data acquisition code on the Python side. The obtained data was then processed to be compliant with NIST SP800-22 and Dieharder test formats.

The outputs obtained from the SHA-256 algorithm were transferred to the computer at high speed (500.000 baud) via the serial port. The data was sent in character string format, marked with the characters # at the beginning and \$ at the end. These markers enabled the Python code to safely parse each data packet and ensure data integrity. Data acquisition was performed using Python-developed code, the incoming data was cleaned in hexadecimal format, and converted into file formats suitable for two different test platforms. The random data outputs used in the study were prepared in accordance with the formats required by the relevant test types. For the NIST test suite, the data file was created by writing each bit to a separate line, resulting in the random_nist.txt file. For the Dieharder test suite, a file named random_dieharder.bin containing random data in raw binary format was created. This ensured that each test type could be fed according to its specific data input requirements.

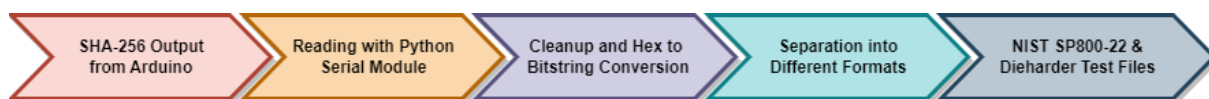


Figure 6. Data collection and formatting process using Python

A total of 1.204.224 bits of data were generated for each test file, and for the Dieharder tests, these data were combined and adjusted to the appropriate size. The Python code was also structured to perform basic statistical pre-analysis, such as bias analysis and Shannon entropy estimation, on the bit sequence. The data processing process performed on the Python side is shown in Figure 6. The summarized data from Arduino is read and parsed using Python and converted into two different file formats suitable for randomness tests.

3.3. Entropy Extraction and Cryptographic Post-Processing

In the Arduino-based hardware random number generator, the multiple noise data obtained from the analog pins is configured to read five different analog pins ($A_0 - A_4$) in each sampling cycle. The voltage-time graphs formed on each pin are presented in Figure 7. The values coming from these pins are digitized at each sampling moment and combined with bit-shifting and XOR operations to form a single value. This process increases the entropy value by reducing the correlation between pins and ensures that a homogeneous data sequence is obtained. These processed values are constrained to the range 0 – 255 in byte (8-bit) format. This constraint is preferred for both memory optimization and more consistent execution of statistical analyses.

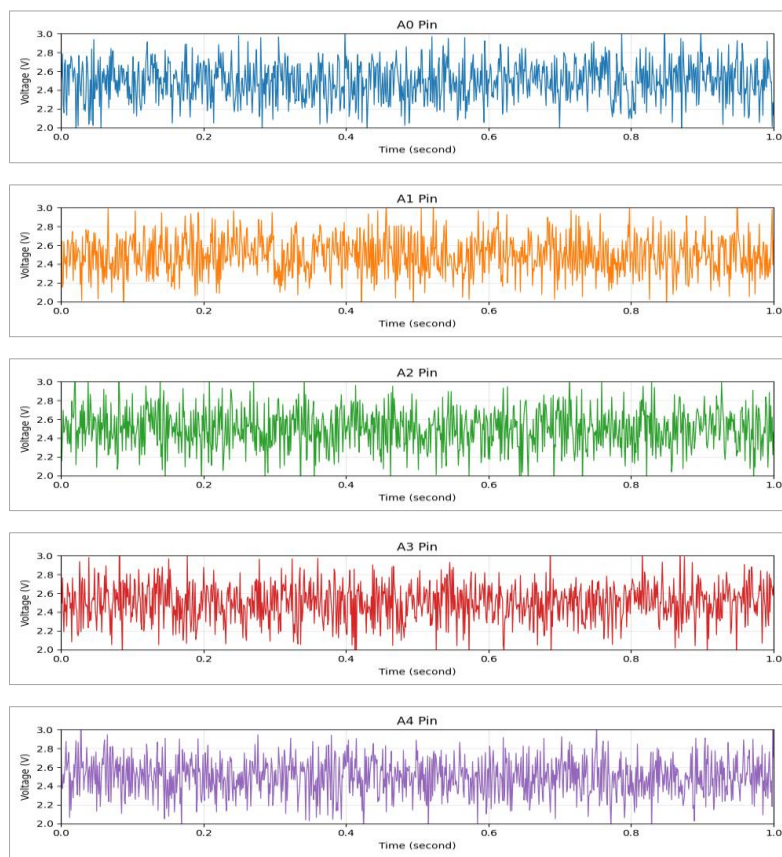


Figure 7. Voltage-time graph for $A_0 - A_4$ pins

Arduino's 10-bit resolution ADC produces numerical values in the range of 0 – 1023 when converting analog signals to digital format. However, these raw data are converted to a lower resolution but more compact representation during the system's random number generation and processing stages. The fundamental reason for this is that the values obtained as a result of the bit-shift and XOR operations applied during the random number generation process represent only a 8-bit (1 byte) value each time. Therefore, the numbers obtained at the end of the process are naturally limited to the range 0 – 255. There are several important reasons for this limitation:

- Memory and bandwidth optimization: 8-bit values consume fewer resources both for data transmission (via UART) and for processing on the computer side.
- Compatibility with hash operations: Cryptographic functions such as SHA-256 operate on 8-bit data blocks, so providing data in the appropriate format directly simplifies the processing.
- Statistical consistency: Entropy, frequency analysis, and most other tests are more commonly and consistently performed on 8-bit data.

Values taken from five different pins in each sampling cycle from Arduino were combined using bit rotation and XOR-based mixing operations. As shown in equation 8, each analog reading was processed with specific shift amounts (left/right rotation), and XOR operations were used to eliminate both inter-pin and temporal dependencies. Thus, the data derived from different components of environmental noise is made more irregular and unpredictable.

$$b_i = \left(\bigoplus_{j=0}^n (a_j \ll r_j \mid a_j \gg (8 - r_j)) \right) \ll s_i \oplus b_{i-1} \quad (8)$$

Here, ' a_j ': the digital value of the analog pin at the sampling instant, ' $r_j = j \bmod 8$ ': the shift amount specific to each pin, ' $s_i = i \bmod 8$ ': the final scrambling amount depending on the sampling index, ' $\ll$ ' and ' $\gg$ ': left and right bit shift operations, respectively, ' $\oplus$ ': the XOR operator. This structure XORs the analog values from different pins in each sampling cycle after shifting them by specific amounts; then, the resulting intermediate outputs are reshuffled with the previous sample's output (b_{i-1}). This reduces both inter-pin correlation (source dependency) and temporal dependency. As a result, equation 8 defines the system as a multi-layer mixing function designed to produce higher entropy and unpredictable bits by mixing analog noise sources both across different pins and temporally.

The direct use of these analog values carries the risk of systematic biases and low entropy. Therefore:

- Each analog reading is subjected to XOR operations with values taken from different pins.
- The results are mixed using cyclic bit shifts.
- The resulting values are written to a buffer and fed into a SHA-256 function in 128 sampling to concentrate entropy.

In this context, the raw output of the ADC in the 0–1023 range is not used directly. Multiple samples taken from the analog pins via $A_0 - A_4$ were combined in the software using bit shifting to obtain a single 8-bit random number sequence. The time variation of the random number sequence formed after the shifting and post processing steps and its distribution in the frequency spectrum are shown in Figure 8.

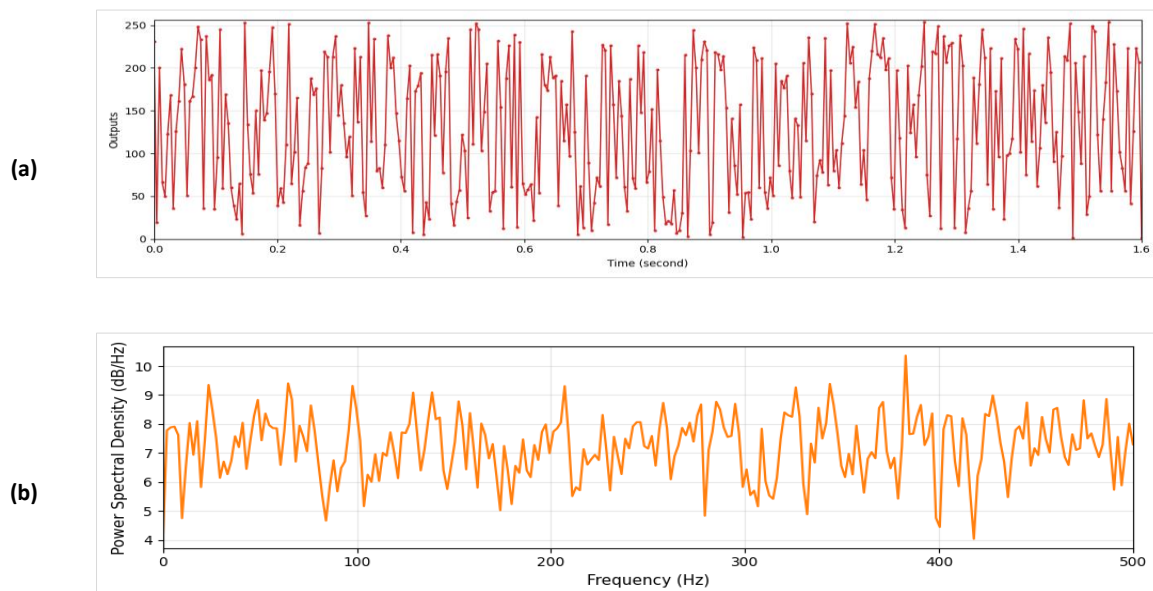


Figure 8. (a) Distribution and (b) power spectral density of 8-bit random outputs after shifting and post processing

The graphs presented in Figure 8 reveal the behavior of the generated random number sequence. The 8-bit values seen on the time axis are irregularly distributed in the 0–255 range and have no discernible repetition pattern. In the power spectral density graph, the energy is spread across a wide frequency band, and no narrow-band peaks are observed. The weak DC component at low frequencies is a natural effect. These findings reveal that the random number sequence obtained after the shift and post processing steps exhibits characteristics close to white noise and contains no periodicity.

As a result of the aggregation process, each data file was 1.204.224 bits (approximately 147 KB) in length. To achieve the data size required for the Dieharder tests, 129 of these files were combined. Throughout the process, the data sampling interval was set to 1 ms, and 128 analog samples were taken for each SHA-256 hash value. Thus, each generated block provided an output of 32 bytes (256 bits) in length. The data obtained with Python code was first converted from HEX format to binary format, each character was processed as 4 bits, and then grouped into 8-bit chunks and separated into bytes to be made suitable for Dieharder tests. Since the system has a data transmission capacity of 500.000 baud rate, the required data set size could be collected in a relatively short time of approximately 5 – 6 minutes. This transmission speed provides sufficient sample size for both test packages to be reliably applied.

4. Performance Analysis

In this section, the statistical accuracy and system performance of the Arduino Uno-based TRNG were comprehensively evaluated. First, the quality of the generated bit sequences was examined using basic statistical measures such as Shannon entropy, bias, and min-entropy. Then, compliance with international standards was tested by applying the NIST SP 800-22 and Dieharder test packages, which are widely used in the literature. The results obtained reveal not only the randomness quality of the generated bit sequences but also the hardware efficiency of the system. In this context, both statistical analyses and time-based performance measurements are presented; data collection speed, processing time, and output production capacity are examined in detail.

A review of the current literature on hardware random number generators reveals significant differences in parameters such as production speed, entropy quality, and cost. Although FPGA- and ASIC-based TRNG solutions offer high performance, they can be limiting in terms of cost and accessibility. Table 4 presents the production capacities and estimated costs of different TRNG systems according to their hardware infrastructure, highlighting the advantages of the Arduino-based structure proposed in this study.

While the proposed Arduino-based TRNG operates at a lower bit generation rate compared to FPGA or ASIC-based systems, it offers a unique balance between cost, accessibility, and reliability. The total hardware cost remains below 10 USD, and the system consumes less than 50 mA during operation. Despite these modest specifications, the design achieves statistically verified randomness, making it suitable for educational purposes, lightweight cryptographic applications, and prototyping environments where affordability and simplicity are prioritized over extreme performance. This cost–security trade-off demonstrates that reliable randomness can be achieved even on minimal hardware platforms.

Table 4. Comparison of hardware and bit output speeds with other studies

Article & Year	Hardware Type	Production Speed (bit/s)	Approximate Hardware Cost (USD)
Cao et al. (2021)[42]	ASIC (2-phase Ring Oscillator)	~3 Gbps	\$50 – 100 +
Liao et al. (2022)[43]	FPGA (Chaotic Jitter)	~1.4 Gbps	\$60 – 100
Alçın et al. (2019)[44]	FPGA (ANN-Based)	~640 Mbps	\$80 – 150
Gu et al. (2018)[45]	FPGA	~256 Mbps	\$50 – 100
Dang et al. (2024)[46]	FPGA (Jitter + Ring Oscillator)	~360 Mbps	\$60 – 120
Pazos et al. (2023)[47]	Memristor + Microcontroller	~360 Mbps	\$30 – 50
Hajduk (2024)[10]	FPGA (Capacitive Oscillator TRNG)	~800 Mbps	\$70 – 120
Sánchez-Solano et al. (2024)[11]	ASIC/FPGA (Ring Oscillator PUF+TRNG)	~500 Mbps	\$80 – 150
Peetermans & Verbaauwhede (2024)[48]	ASIC (Phase Noise Analysis-Based TRNG)	~1 – 2 Gbps	\$100 – 200
Ahmed, Kim & Buchanan (2024)[49]	FPGA (Race Hazard + Jitter TRNG)	~500 Mbps	\$70 – 130
This study (2025)	Arduino + SHA-256	~4 kbps	< \$10

Random data obtained from the Arduino Uno-based TRNG system was examined both statistically and in terms of timing. This analysis was performed at two levels to evaluate the production quality and real-time performance of the system together. Basic statistical measurements were performed on the collected data. The mean, variance, and Shannon entropy were calculated separately for each analog pin. The data is shown in Table 5:

Table 5. Statistical measurements on analog pins

Analog Pin	Mean	Variance	Shannon Entropy (bits)
A_0	116.14	81.46	2.63
A_1	114.72	79.88	2.69
A_2	114.34	78.64	2.88
A_3	115.80	78.65	3.25
A_4	108.23	136.05	3.52

The Shannon entropy value, calculated over a maximum of 8 bits, does not indicate high quality for the direct analog data. However, it has been observed that this entropy is significantly increased at the bit level after the post processing with SHA-256.

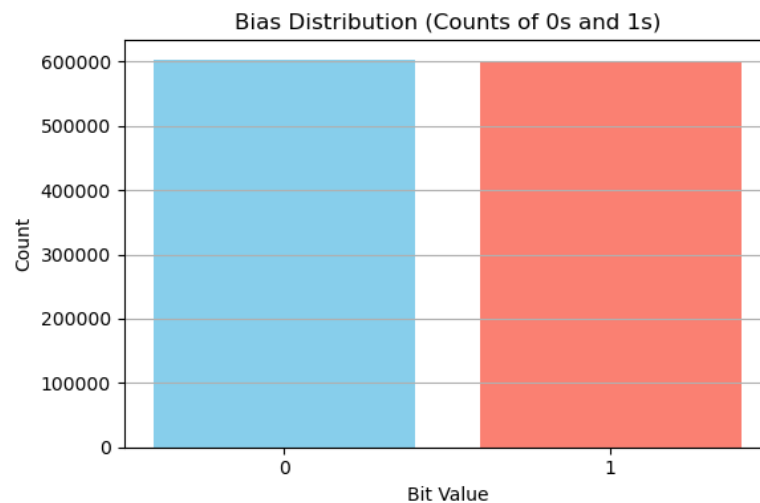


Figure 9. Bias distribution histogram

Figure 9 shows that both bit values (0 and 1) are generated approximately equally by the system, which is a positive indicator in terms of statistical randomness. Additionally, in the bias analysis (0 and 1 ratio), a symmetric distribution in the range of 49.87 – 50.13% was obtained across the entire dataset. This ratio is very close to the expected values in terms of randomness.

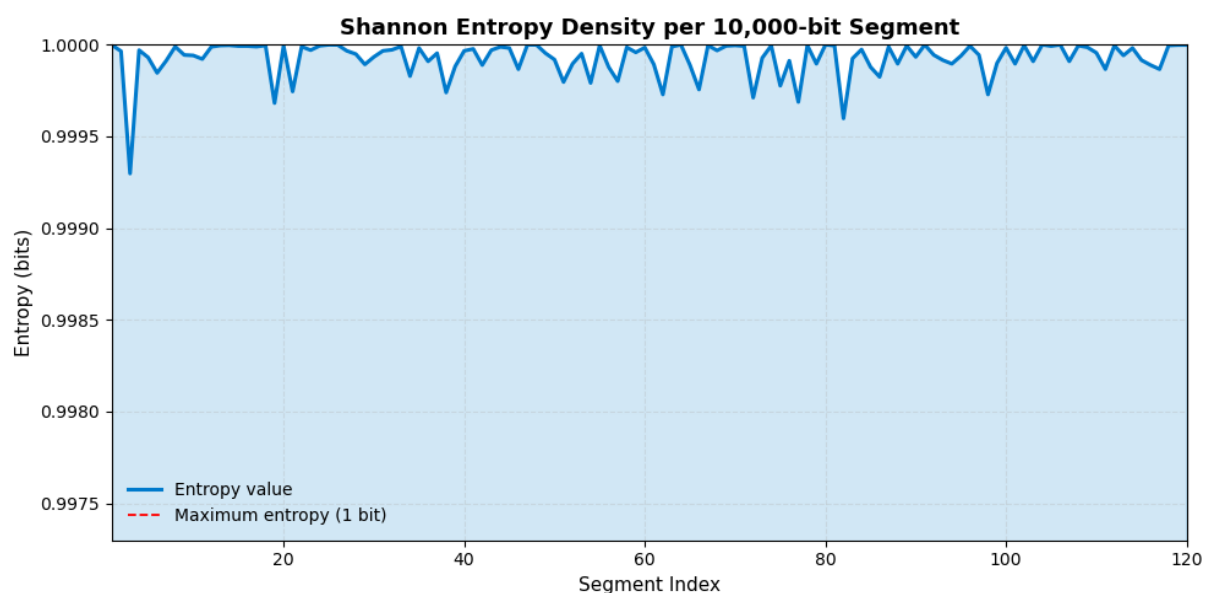


Figure 10. Entropy density distribution graph

Figure 10 shows the Shannon entropy values calculated for each segment of the 1.204.224-bit data set obtained from the TRNG system, divided into 10.000-bit segments. This graph aims to analyze how consistently the system behaves in generating randomness over time.

The entropy values in the graph largely fluctuate between 0.99 and 1.00, indicating that the distribution of 0 and 1 in each segment is almost equal and that the system produces a high degree of randomness. These values, which fluctuate close to 1.00, indicate ideal randomness in terms of

information theory. The small variations observed between segments are consistent with natural noise variations caused by hardware and are considered within the expected system behavior.

System performance was evaluated based on the time required to generate a random block from each analog sample of 128. The subcomponent times for each production cycle, as measured by microtime measurements, are as follows in Table 6:

Table 6. Time-based performance analysis

Process Step	Average Duration (μ s)
ADC + Mixing	73.728
SHA-256 Hashing	3.883
Serial Transmission (500k baud)	2.715
Total Duration	80.342

Based on these data, the system's sampling rate and output production rate have been calculated:
ADC sampling frequency:

$$f = \frac{128}{73.728} \approx 1.735 \text{ kHz} \quad (9)$$

Raw bit production rate:

$$\frac{1024}{0.073728 \text{ s}} \approx 13.889 \text{ kbps} \quad (10)$$

SHA-256 post-processing output rate:

$$\frac{256 \text{ bit}}{0.080342 \text{ s}} \approx 3.186 \text{ kbps} \quad (11)$$

These results show that the developed TRNG system enables high-frequency data generation after ADC sampling; secure randomness output, summarized using the SHA-256 algorithm, is obtained at a speed of approximately 3.2 kbps. Furthermore, the analog noise collection, bit mixing, and SHA-256 post processing steps implemented in the TRNG system demonstrate effectiveness in increasing and preserving entropy. Additionally, the fact that the entire data set produces high entropy not only in the overall average but also in each local segment is a positive indicator of the system's reliability.

4.1 NIST SP 800-22 Test Results

The NIST SP 800-22 test suite is one of the most widely used standards for evaluating cryptographic random number generators. All tests in this suite (15) were successfully passed. The test results are shown in Figure 11:

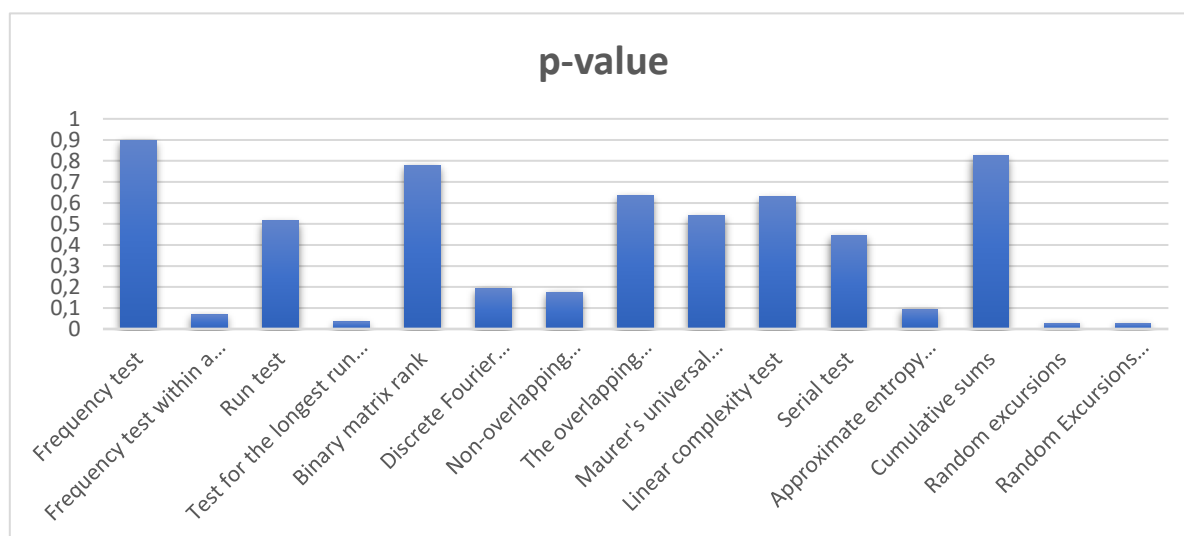


Figure 11. NIST SP800-22 test results graph

The overall success rate is 100%. The p-values are distributed in the range [0.01, 0.99], and no weak results were observed in any test.

4.2 Dieharder Test Results

Dieharder is an extended version of the classic Diehard tests and offers a rich set of tests to evaluate statistical randomness quality. This test suite consists of various tests designed to detect both low-level and complex statistical patterns. Data obtained from our TRNG system was subjected to this test suite, and the p-value and success status were recorded for each test.

Table 7. Statistical Evaluation of the Proposed TRNG Using the Dieharder Test Suite

Test Name	p-value	Assessment
diehard_birthdays	0.17885317	PASSED
diehard_operm5	0.29185347	PASSED
diehard_rank_32x32	0.07231591	PASSED
diehard_rank_6x8	0.26459426	PASSED
diehard_bitstream	0.56321169	PASSED
diehard_opso	0.05987832	PASSED
diehard_oqso	0.00064218	WEAK
diehard_dna	0.35609105	PASSED
diehard_count_1s_str	0.27574439	PASSED
diehard_count_1s_bytes	0.83490945	PASSED

Test Name	p-value	Assessment
diehard_parking_lot	0.38297064	PASSED
diehard_2dsphere	0.95127384	PASSED
diehard_3dsphere	0.92300409	PASSED
diehard_squeeze	0.04851172	PASSED
diehard_sums	0.04221761	PASSED
diehard_runs	0.66028357	PASSED
diehard_craps	0.05319428	PASSED
marsaglia_tsang_gcd	0.07139217	PASSED
sts_monobit	0.17677682	PASSED
sts_runs	0.63841504	PASSED
sts_serial	0.01903009	PASSED
rgb_bitdist	0.95072304	PASSED
rgb_minimum_distance	0.23971804	PASSED
rgb_permutations	0.22404567	PASSED
rgb_lagged_sum	0.03120759	PASSED
rgb_kstest_test	0.11039114	PASSED
dab_byte_distribution	0.04612732	PASSED
dab_dct	0.28436269	PASSED
dab_filltree	0.04863722	PASSED
dab_filltree2	0.00051792	WEAK
dab_monobit2	0.97732488	PASSED

When examining the dieharder test results provided in Table 7, a total of 33 different tests were applied, considering tests with the same name only once. Of these tests, 31 were successful and 2 were weak. Tests classified as "weak" are generally those with borderline p-values that may yield different results when repeated. Therefore, they should be treated as outputs requiring careful evaluation rather than outright failures.

The results for the Dieharder tests are as follows for the total 33 test, as shown in Figure 12:

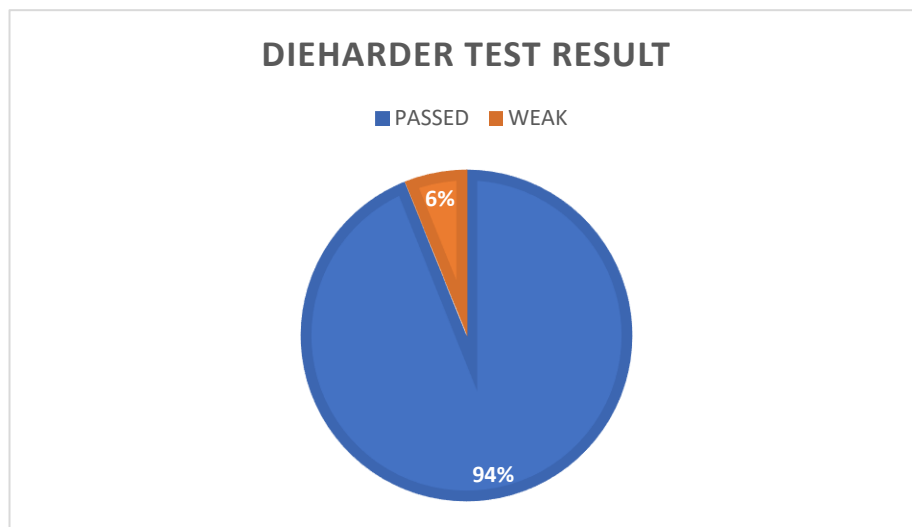


Figure 12. Dieharder test result distribution

Overall, approximately 94% of Dieharder tests were successful, while 6% yielded weak results. The tests reported as weak, particularly the oqso and dab_filltree2 tests, are highly sensitive to entropy limitations and noise imbalances in randomness sources. This can be explained by the limited ADC resolution and sensitivity to environmental noise of the Arduino Uno used. These results indicate that the system's overall randomness capacity is acceptable but contains weaknesses related to specific statistical structures. In this context, it is recommended to improve the post processing stage, use alternative mixing techniques, or support it with more sophisticated entropy sources. Considering the challenging nature of Dieharder, this level of success rate is satisfactory, especially for microcontroller-based TRNG systems.

4.3 System-Level Evaluation and Discussion

When evaluated both theoretically and experimentally, the system demonstrates strengths in various aspects. Although the signals obtained via analog pins initially have limited entropy, the randomness quality of the generated bit sequences is significantly enhanced through the XOR-based mixing operation applied to these signals and the subsequent SHA-256-based post processing. The bias analysis and Shannon entropy measurements performed reveal that the final output exhibits a statistically balanced and homogeneous distribution. Achieving a success rate of 100% in the NIST SP800-22 tests, which are a fundamental reference for randomness verification, demonstrates that the system meets the basic randomness requirements.

While general validity is maintained within the scope of Dieharder tests, "weak" results were encountered in some sub-tests. The main reason for this is thought to be the relatively limited amount of data used in the tests (approximately 155 MB, or $129 \times 1.204.224$ bits). This is because high-resolution test packages such as Dieharder generally produce more reliable and consistent results with gigabit-level data. In particular, the two subtests (oqso and dab_filltree2) yielded "weak" p-values. These results do not indicate a structural bias or deterministic pattern in the generated sequence but are attributed to the high sensitivity of these subtests to sample size and statistical variance. Both of these Dieharder tests perform multidimensional occupancy and tree-filling analyses that become more reliable

with gigabit-scale input data. Therefore, the relatively smaller data volume used in this study can lead to minor statistical deviations, which explains the observed weak outcomes. Despite this limitation, the overall 93.94% pass rate across all Dieharder tests validates the statistical soundness of the proposed TRNG architecture.

Although the proposed Arduino-based TRNG provides statistically verified randomness, its performance is naturally limited by the microcontroller's 10-bit ADC resolution and moderate sampling rate. Additionally, environmental noise variability can cause minor entropy fluctuations during data acquisition. Despite these limitations, the system's open-source, low-cost, and reproducible structure makes it suitable for educational, prototyping, and lightweight security applications.

However, the fact that all components of the system consist of low-cost and open-source hardware and software elements makes it both accessible and reproducible. With these features, the system offers sufficient quality for educational applications, academic research, and scenarios requiring low to medium security. However, for applications requiring higher security levels, it is recommended that the system be improved at the architectural level and tested with larger data sets.

5. Conclusions and Recommendations

In this study, physical random number generation was performed on the low-cost and limited-resource Arduino Uno microcontroller, and the statistical quality of the generated data was comprehensively analyzed. It has been experimentally demonstrated that data close to cryptographic-level randomness can be obtained using only analog input pins and entropy-enhancing algorithms, without the need for hardware complexity. Within the system, analog noise was evaluated as a physical entropy source due to its susceptibility to environmental conditions; these signals were sampled from multiple ADC pins, processed at the bit level, and then condensed using the SHA-256 algorithm. The approximately 1.2 million bits of data obtained were recorded in appropriate formats using Python and subjected to NIST SP800-22 and Dieharder tests. While all NIST tests were successfully passed, the Dieharder tests also yielded largely "successful" results, with only two tests receiving a "weak but acceptable" rating. These weak results, particularly observed in the oqso and dab_filltree2 tests, reflect the system's sensitivity to limited entropy sources and ADC-based noise variability. However, the overall success rate indicates that the system's ability to generate functional randomness is quite high.

The findings reveal that the proposed design possesses strong points such as low cost, accessibility, open-source software architecture, suitability for real-time operation, and compatibility with test systems for data generation. However, limitations such as the limited resolution of the ADC, sensitivity to environmental noise, and lower entropy density compared to hardware-based TRNG solutions have been observed. The poor results in Dieharder tests are considered to stem from these physical limitations of the hardware used. By addressing these limitations, it is possible that the developed system will not be limited to educational and experimental studies, but will also become usable in cryptographic applications requiring high security.

Future work may include collecting data at higher sampling frequencies with FPGA-based systems, monitoring data in real time with machine learning-based analysis methods, and making dynamic improvements. Furthermore, using Keccak (SHA-3), BLAKE2, or lightweight cryptographic algorithms

suitable for embedded systems in the post processing stage instead of SHA-256 can increase both performance and security. The integration of external high-resolution ADC modules and the use of multi-channel entropy collection structures can enrich the system's randomness capacity. In addition, future developments of the proposed TRNG may focus on optimizing the ADC sampling frequency, improving the effective resolution through external converters, and implementing real-time entropy estimation to enhance throughput without compromising randomness quality. As demonstrated by this study, it is possible to generate reliable randomness even under constrained hardware and low energy consumption conditions; therefore, the developed structure can be considered an accessible and reproducible solution in fields such as IoT, embedded system security, and lightweight cryptography.

Acknowledgment

The author would like to thank the reviewers and editorial boards of the International Journal of Pure and Applied Sciences.

Conflict of Interest

The authors declare that there is no conflict of interest in this study. The manuscript was written with the contributions of all authors.

Research and Publication Ethics Statement

The author declare that they have complied with the scientific, ethical, and citation principles of the *International Journal of Pure and Applied Sciences* throughout all stages of the study.

References

- [1] M. Stipcevic, "Quantum random number generators and their applications in cryptography," in *Advanced Photon Counting Techniques VI*, May 2012, pp. 837504-837504–15. doi: 10.1117/12.919920.
- [2] M. Herrero-Collantes and J. C. Garcia-Escartin, "Quantum random number generators," *Rev. Mod. Phys.*, vol. 89, no. 1, p. 015004, Feb. 2017, doi: 10.1103/RevModPhys.89.015004.
- [3] Y. Zhang, K. Zhong, and J. Zhang, "High Throughput and Compact FPGA TRNGs Based on Hybrid Entropy, Reinforcement Strategies, and Automated Exploration," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 45, no. 1, pp. 519–532, Jan. 2026, doi: 10.1109/TCAD.2025.3572838.
- [4] P. L'Ecuyer, *Handbook of Computational Statistics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-21551-3.
- [5] T. Stojanovski and L. Kocarev, "Chaos-based random number generators-part I: analysis [cryptography]," *IEEE Trans. Circuits Syst. I Fundam. Theory Appl.*, vol. 48, no. 3, pp. 281–288, Mar. 2001, doi: 10.1109/81.915385.
- [6] Ç. K. Koç, *Cryptographic Engineering*. Boston, MA: Springer US, 2009. doi: 10.1007/978-0-387-71817-0.
- [7] M. Stipcevic and Ç. K. Koç, *True Random Number Generators*. Cham: Springer International Publishing, 2014. doi: 10.1007/978-3-319-10683-0.

- [8] C. K. Volos and A. S. Andreatos, "Secure Text Encryption Based on Hardware Chaotic Noise Generator," *J. Appl. Math. Bioinforma.*, vol. 5, pp. 15–35, 2015.
- [9] C. S. Kuka, Y. Hu, Q. Xu, J. Chandler, and M. Alkahtani, "A Novel True Random Number Generator in Near Field Communication as Memristive Wireless Power Transmission," *J*, vol. 4, no. 4, pp. 764–783, Nov. 2021, doi: 10.3390/j4040052.
- [10] Z. Hajduk, "Field-Programmable Gate Array-Based True Random Number Generator Using Capacitive Oscillators," *Electronics*, vol. 13, no. 23, p. 4819, Dec. 2024, doi: 10.3390/electronics13234819.
- [11] S. Sánchez-Solano, L. F. Rojas-Muñoz, M. C. Martínez-Rodríguez, and P. Brox, "Hardware-Efficient Configurable Ring-Oscillator-Based Physical Unclonable Function/True Random Number Generator Module for Secure Key Management," *Sensors*, vol. 24, no. 17, p. 5674, Aug. 2024, doi: 10.3390/s24175674.
- [12] A. M. Garipcan and E. Erdem, "Kriptografide Rasgelelik Kavramı ve Gerçek Rasgele Sayı Üreteçlerinin Test Metodolojisi," *DÜMF Mühendislik Derg.*, vol. 1, pp. 61–75, Feb. 2024, doi: 10.24012/dumf.1384343.
- [13] H. G. Katzgraber, "Random Numbers in Scientific Computing: An Introduction," May 2010, Accessed: May 29, 2025. [Online]. Available: <https://arxiv.org/pdf/1005.4117>
- [14] B. Kristinsson, "The Arduino as a Hardware Random-Number Generator," pp. 1–20, Dec. 2012, [Online]. Available: <http://arxiv.org/abs/1212.3777>
- [15] N. Baker and W. Lui, "An Analysis of Entropy and Randomness as it relates to True Random Number Generation," *UMD Lab. Phys. Sci.*, 2024.
- [16] F. Frustaci, F. Spagnolo, S. Perri, and P. Corsonello, "A High-Speed FPGA-Based True Random Number Generator Using Metastability With Clock Managers," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 70, no. 2, pp. 756–760, Feb. 2023, doi: 10.1109/TCSII.2022.3211278.
- [17] R. Soorat, M. K., and A. Vudayagiri, "Hardware Random number Generator for cryptography," *Nanosyst. Physics, Chem. Math.*, pp. 600–605, Oct. 2015, doi: <https://doi.org/10.48550/arXiv.1510.01234>.
- [18] V. Fischer and F. Bernard, "True Random Number Generators in FPGAs," in *Security Trends for FPGAs*, Dordrecht: Springer Netherlands, 2011, pp. 101–135. doi: 10.1007/978-94-007-1338-3_5.
- [19] 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," Gaithersburg, MD, Jan. 2018. doi: 10.6028/NIST.SP.800-90B.
- [20] S. Yakut, T. Tuncer, and A. B. Ozer, "Secure and Efficient Hybrid Random Number Generator Based on Sponge Constructions for Cryptographic Applications," *Elektron. ir Elektrotehnika*, vol. 25, no. 4, pp. 40–46, Aug. 2019, doi: 10.5755/j01.eie.25.4.23969.
- [21] S. Yakut, "Random Number Generator Based on Discrete Cosine Transform Based Lossy Picture Compression," *NATURENGS MTU J. Eng. Nat. Sci.*, vol. 2, pp. 76–85, Nov. 2021, doi: 10.46572/naturengs.1009013.
- [22] İ. E. Yüksel *et al.*, "TuRaN: True Random Number Generation Using Supply Voltage

- Underscaling in SRAMs,” Nov. 2022, Accessed: Sep. 10, 2025. [Online]. Available: <https://arxiv.org/pdf/2211.10894>
- [23] A. Olgun *et al.*, “QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips,” *Proc. - Int. Symp. Comput. Archit.*, vol. 2021-June, pp. 944–957, May 2021, doi: 10.1109/ISCA52012.2021.00078.
- [24] S. Ghandali, D. Holcomb, and C. Paar, “Temperature-Based Hardware Trojan For Ring-Oscillator-Based TRNGs,” Sep. 2019, Accessed: Sep. 10, 2025. [Online]. Available: <https://arxiv.org/pdf/1910.00735>
- [25] H. J. Ng, S. Yang, Z. Yao, H. Yang, and C. Lim, “Provably Secure Randomness Generation from Switching Probability of Magnetic Tunnel Junctions,” *Phys. Rev. Appl.*, vol. 19, no. 3, p. 034077, Mar. 2023, doi: 10.1103/PhysRevApplied.19.034077.
- [26] A. Shukla *et al.*, “A True Random Number Generator for Probabilistic Computing using Stochastic Magnetic Actuated Random Transducer Devices,” *Proc. - Int. Symp. Qual. Electron. Des. ISQED*, vol. 2023-April, Apr. 2023, doi: 10.1109/ISQED57927.2023.10129319.
- [27] S. Fu *et al.*, “RHS-TRNG: A Resilient High-Speed True Random Number Generator Based on STT-MTJ Device,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 31, no. 10, pp. 1578–1591, Oct. 2023, doi: 10.1109/TVLSI.2023.3298327.
- [28] F. N. Bostanci *et al.*, “DR-STRaNGe: End-to-End System Design for DRAM-based True Random Number Generators,” *Proc. - Int. Symp. High-Performance Comput. Archit.*, vol. 2022-April, pp. 1141–1155, Jan. 2022, doi: 10.1109/HPCA53966.2022.00087.
- [29] R. Cazzola, C. Minwalla, C. Chan, and J. Plusquellic, “A TRNG Implemented using a Soft-Data Based Sponge Function within a Unified Strong PUF Architecture,” Jun. 2025, Accessed: Sep. 10, 2025. [Online]. Available: <https://arxiv.org/pdf/2506.17795>
- [30] F. Pebay-Peyroula, L. Benea, M. Carmona, and R. Wacquez, “OpenTRNG: an open-source initiative for ring-oscillator based TRNGs,” in *2024 IEEE International Conference on Design, Test and Technology of Integrated Systems (DTTIS)*, IEEE, Oct. 2024, pp. 1–6. doi: 10.1109/DTTIS62212.2024.10780212.
- [31] F. Pebay-Peyroula, L.-P. Benea, M. Carmona, and R. Wacquez, “On the Impact of Metastability in Jitter Based TRNG,” pp. 279–288, 2025, doi: 10.1109/host64725.2025.11050059.
- [32] M. Yi, Z. Liu, Z. Miao, Y. Lu, H. Liang, and L. Ma, “A true random number generator based on autonomous Boolean network with imbalanced node oscillation rings,” *Microelectronics J.*, vol. 160, no. November 2024, p. 106694, Jun. 2025, doi: 10.1016/j.mejo.2025.106694.
- [33] T. K. Dang, K. D. Nguyen, T. T. Hoang, and C. K. Pham, “A Unified Approach to Strong PUF and TRNG Using Ring Generator for Cryptography,” *IEEE Internet Things J.*, vol. 12, no. 17, pp. 36590–36603, 2025, doi: 10.1109/JIOT.2025.3582869.
- [34] A. M. Garipcan and E. Erdem, “Design, FPGA implementation and statistical analysis of a high-speed and low-area TRNG based on an AES s-box post-processing technique,” *ISA Trans.*, vol. 117, pp. 160–171, Nov. 2021, doi: 10.1016/j.isatra.2021.01.054.
- [35] C. Carver *et al.*, “Integrated Differential Conjugate Homodyne Detection for Quantum Random Number Generation,” Dec. 2024, [Online]. Available: <http://arxiv.org/abs/2412.02077>

- [36] L. E. Bassham *et al.*, “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” Gaithersburg, MD, 2010. doi: 10.6028/NIST.SP.800-22r1a.
- [37] R. G. Brown, “DieHarder - A Gnu Public License Random Number Tester,” *Dieharder*, 2006, [Online]. Available: <https://rurban.github.io/dieharder/manual/dieharder.pdf>
- [38] S. Li, Z. Lin, Y. Yang, and R. Ning, “A High-Performance FPGA PRNG Based on Multiple Deep-Dynamic Transformations,” *Entropy*, vol. 26, no. 8, p. 671, Aug. 2024, doi: 10.3390/e26080671.
- [39] T. Hansen and D. Eastlake, “US Secure Hash Algorithm 2 (SHA-2). RFC 6234.,” 2001. <https://datatracker.ietf.org/doc/html/rfc6234>
- [40] J. Daemen and V. Rijmen, *The Design of Rijndael*. in Information Security and Cryptography. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. doi: 10.1007/978-3-662-04722-4.
- [41] M. J. Dworkin, “SHA-3 standard :,” Gaithersburg, MD, Jul. 2015. doi: 10.6028/NIST.FIPS.202.
- [42] Y. Cao, X. Zhao, W. Zheng, Y. Zheng, and C. H. Chang, “A New Energy-Efficient and High Throughput Two-Phase Multi-Bit per Cycle Ring Oscillator-Based True Random Number Generator,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 69, no. 1, pp. 272–283, Jan. 2022, doi: 10.1109/TCSI.2021.3087512.
- [43] T. L. Liao, P. Y. Wan, and J. J. Yan, “Design and Synchronization of Chaos-Based True Random Number Generators and Its FPGA Implementation,” *IEEE Access*, vol. 10, pp. 8279–8286, 2022, doi: 10.1109/ACCESS.2022.3142536.
- [44] M. Alcin, I. Koyuncu, M. Tuna, M. Varan, and I. Pehlivan, “A novel high speed Artificial Neural Network-based chaotic True Random Number Generator on Field Programmable Gate Array,” *Int. J. Circuit Theory Appl.*, vol. 47, no. 3, pp. 365–378, 2019, doi: 10.1002/cta.2581.
- [45] F. Mei, L. Zhang, C. Gu, Y. Cao, C. Wang, and W. Liu, “A highly flexible lightweight and high speed true random number generator on FPGA,” *Proc. IEEE Comput. Soc. Annu. Symp. VLSI, ISVLSI*, vol. 2018-July, pp. 399–404, 2018, doi: 10.1109/ISVLSI.2018.00079.
- [46] T. K. Dang, T. T. Hoang, and C. K. Pham, “A True Random Number Generator on FPGA with Jitter-Sampling by Ring Generator,” *2024 Asian Hardw. Oriented Secur. Trust Symp. AsianHOST 2024*, pp. 1–6, 2024, doi: 10.1109/AsianHOST63913.2024.10838391.
- [47] S. Pazos *et al.*, “Hardware implementation of a true random number generator integrating a hexagonal boron nitride memristor with a commercial microcontroller,” *Nanoscale*, vol. 15, no. 5, pp. 2171–2180, 2023, doi: 10.1039/D2NR06222D.
- [48] A. Peetermans and I. Verbauwhede, “Characterization of Oscillator Phase Noise Arising From Multiple Sources for ASIC True Random Number Generation,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 71, no. 3, pp. 1144–1157, Mar. 2024, doi: 10.1109/TCSI.2023.3343114.
- [49] H. O. Ahmed, D. Kim, and W. J. Buchanan, “A True Random Number Generator Based on Race Hazard and Jitter of Braided and Cross-Coupled Logic Gates Using FPGA,” *IEEE Access*, vol. 12, no. December, pp. 182943–182955, 2024, doi: 10.1109/ACCESS.2024.3512419.