

THE ILLUSION OF RANDOMNESS: A VISUAL AND SCIENTIFIC ANALYSIS OF PSEUDORANDOM NUMBER GENERATORS

Luka Baklaga¹

¹Research and Development Department, Business and Technology University, Georgia

ABSTRACT: One of the most crucial aspects of cybersecurity and privacy is the concept of randomness. More specifically, in the modern cybersecurity environment, the generation of unpredictable numbers is a foundational requirement. Despite its importance, the significant difference between generators suitable for statistical modeling and those secure enough for cryptography is often misunderstood. The goal of this paper is to provide a practical demonstration of this difference and to highlight the issue of misleading randomness, which is critical for future cryptographic algorithms such as post-quantum cryptography, quantum cryptography, and others where true randomness is essential. The paper presents a practical and contemporary demonstration of a classic security principle. We conducted a simple experiment to showcase how randomness can be as illusory as its cryptographic properties. We compared three well-known generators: a classic Linear Congruential Generator (LCG), Python's standard random module (Mersenne Twister), and Python's cryptographically secure secrets module. By creating separate classes for each module to generate byte streams, visualizing these streams as bitmap images, and subjecting them to Chi-Squared analysis, we reveal a crucial insight: even when a visually patterned and predictable generator passes statistical tests for uniformity, its core security principle is compromised, rendering it dangerously insecure. The results provide a powerful, tangible demonstration of why statistical uniformity is a necessary but insufficient condition for security, and why purpose-built cryptographic modules are indispensable.

KEYWORDS: *cryptography, Pseudorandom, cryptography, CSPRNG, Entropy, Randomness, quantum security*

1. INTRODUCTION

In the current digital environment, where data has become a new form of currency and personal information is a valuable asset, maintaining and enhancing security is one of the most crucial aspects. To ensure a secure environment, cryptography is an essential tool that must be implemented accurately by everyone. The principles of privacy by design and security by design are fundamental components of cryptographic systems. Randomness, which exists physically everywhere and is governed by entropy according to the second law of thermodynamics, plays a vital role in cryptography. It introduces unpredictability, making systems resistant to brute-force attacks. However, randomness can be compromised or corrupted by predictable patterns. For example, in 2008, a seemingly minor change to the OpenSSL library in the Debian operating system caused a catastrophic security failure. This vulnerability affected the system's random number generator used by the OpenSSL package, which is included in Linux distributions such as Ubuntu and other Debian-based systems. The flaw made the randomness predictable, rendering countless SSH and SSL connections across the internet trivial to decrypt (US-CERT 2008). This vulnerability affects only a limited number of systems that use OpenSSL, but it serves as a reminder that similar vulnerabilities already exist in today's random or pseudorandom number generators - or are likely to exist. More importantly, the quality of randomness is not merely an academic concern; it is a critical pillar of digital security.

Our goal with this idea is to define randomness and its illusion through a simple experiment in which we compare three different scale pseudorandom generators. Pseudorandomness can be defined as a sequence generated deterministically (Knuth 1997); these types of algorithms mimic randomness efficiently, making them computationally friendly. From the range of pseudorandom generators, we

have chosen the following: the Linear Congruential Generator (LCG), which is one of the oldest and simplest PRNGs, defined by the following formula (Knuth 1997):

$$X_{n+1} = (a \cdot X_n + c) \bmod m, \quad n \geq 0$$

where X_n is the current pseudo-random number, X_{n+1} is the next number, a is the multiplier, c is the increment, m is the modulus, and \bmod is the modulo operator.

Python's `random.randbytes()` uses the traditional pseudorandom number generator (PRNG) algorithm called Mersenne Twister (Bhattacharjee and Das 2022). In contrast, Python's `secrets.token_bytes()` relies on an operating system–provided cryptographically secure PRNG (CSPRNG) (Schneier et al. 2010). The `secrets` module is designed to generate cryptographically strong random numbers by utilizing its own sources of entropy.

The proposed paper bridges the gap between the abstract theory and the practical reality of PRNG security. We aim to demonstrate that the weaknesses of insecure generators are not merely mathematical concepts but are demonstrable, observable, and statistically measurable phenomena. The paper's goal is to provide a concise conclusion on illusionary randomness, which will be beneficial for future implementations and new cryptographic developments.

2. OBJECTIVES

The proposed paper compares three pseudorandom number generators and based on their visual and scientific analysis, aims to demonstrate that randomness is crucial to security. It also highlights that initial impressions of randomness can be misleading when assessed with the naked eye.

3. RESEARCH METHODOLOGY

3.1 EXPERIMENTAL METHODOLOGY

The experiment aims to compare the byte-stream outputs of three distinct Python-based generators to distinguish randomness from repetitive patterns, which can be easily compromised. For simulation purposes, we used the Python programming language due to its simplicity and broad variety of accessible libraries. In our code configuration, for each generator, we produced 262,144 bytes (512 x 512).

The three distinct Python-based generators and their specifications are as follows:

1. Weak Generator (LCG) is an implementation of a linear congruential generator (LCG) using the parameters from `glibc's rand()` function.
2. Statistically Good Generator (`random`) refers to Python's `random.randbytes()` function, which utilizes the Mersenne Twister algorithm - a general-purpose pseudorandom number generator.
3. Secure Generator (`secrets`) refers to Python's `secrets.token_bytes()`, which uses the operating system's built-in cryptographically secure pseudorandom number generator (CSPRNG) to produce unpredictable random bytes.

For each proposed byte stream, we conducted three analyses:

We conducted a visual analysis by mapping the byte stream to a 512 x 512 grayscale image. Additionally, we performed a quantitative statistical analysis using Pearson's Chi-Squared (χ^2) goodness-of-fit test. a significance level (alpha) of 0.05, a p-value below 0.05 indicates that output distribution is statistically different from uniform. a uniform distribution. To elaborate, for uniform distribution, the expected frequency is the same for all outcomes. In code, the degree of freedom is calculated as $k - 1$, where k is the number of categories (256).; Finally, we performed a frequency analysis where a frequency count of each byte value (0-255) was plotted.

4. EXPERIMENTAL RESULTS AND ANALYSIS

The goal of the experiment was to measure and clearly define randomness, as not every secure system is entirely random; therefore, randomness can be an illusion. The idea is that if a module or function is statistically sound, it may also be cryptographically secure. From the code, we generated static images to visually represent the randomness based on patterns, as well as statistical charts to identify biases (consistent clusters of bars above or below average) for all three functions.

Visual and Frequency Results:

We have visually represented our results to demonstrate a clear and definite hierarchy, as shown in Figure 1,2,3. The first proposed function, idealized as having the lowest randomness, exhibits prominent, repeating diagonal patterns. These patterns are directly related to its core visual artifact, which arises from its simple, linear algorithm. Clear repeating diagonal patterns are evident throughout the image. As previously noted, the statistical chart shows clear biases. In contrast, the second and third modules (random and secrets) produce images that, upon analysis, appear mostly patternless and resemble uniform static.

"random" module showed clear improvement over the LCG, as there are no obvious large-scale patterns, making it appear random to the naked eye. In the "Secrets" module, we observe completely patternless static. Like the random module, the chart for Secrets is perfectly flat, indicating a uniform statistical distribution, as expected from a high-quality CSPRNG.

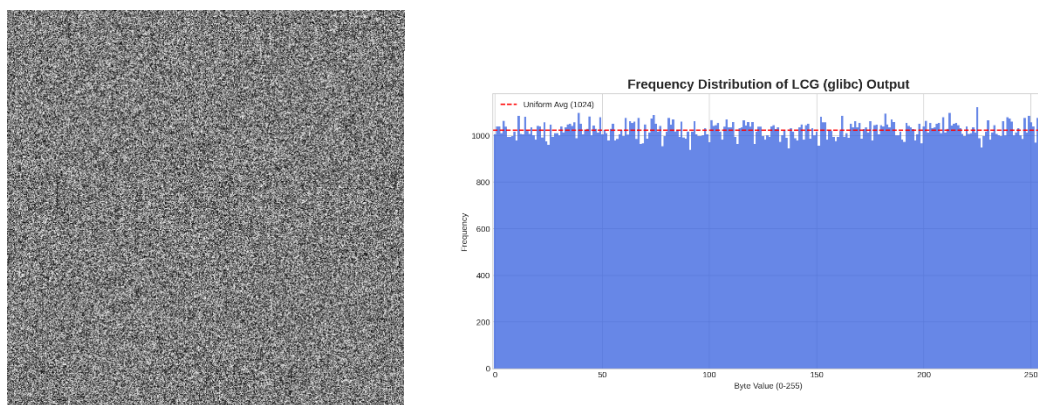


Figure 1. Visual and Frequency Distribution result of LCG module

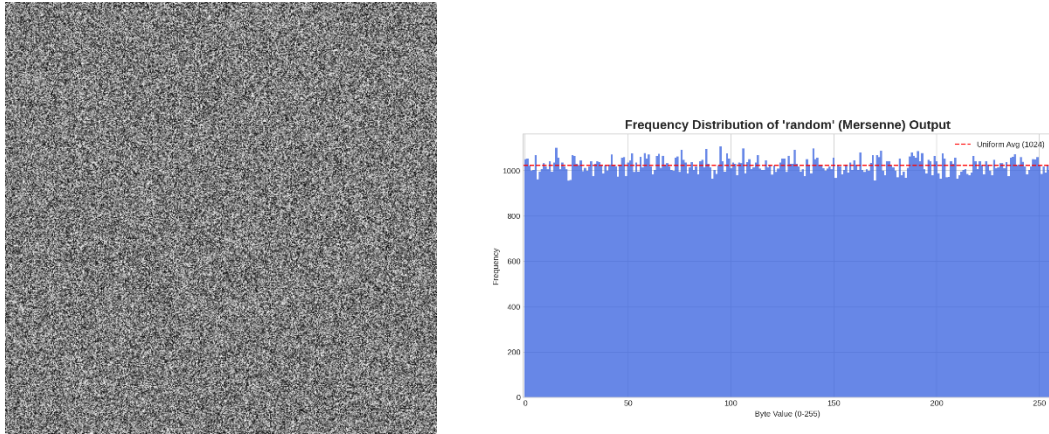


Figure 2. Visual and Frequency Distribution result of random module

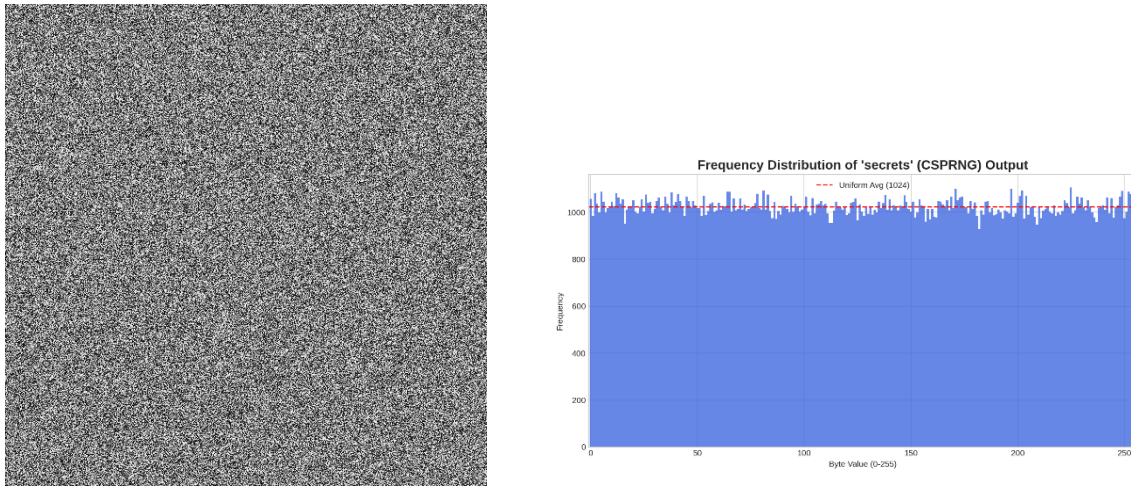


Figure 3. Visual and Frequency Distribution result of secrets module

Quantitative Statistical Results:

The Chi-Squared test offers a precise, scientific measure of the statistical quality observed in the charts. The results from our experimental run are summarized in Table 1.

Table 1. Chi-Squared Goodness-of-Fit Test Results (Degrees of Freedom = 255, $\alpha = 0.05$)

Generator	Chi-Squared Statistic	P-value	Uniformity Test
LCG (glibc)	275.90	0.1760	Passes
'random' (Mersenne)	253.63	0.5125	Passes
'secrets' (CSPRNG)	254.85	0.4909	Passes

From the chart, we can see that LCG's output produced a p-value of 0.1760, which is well above the 0.05 threshold. This indicates that the test did not find a statistically significant deviation from a uniform distribution. However, when compared to the visual evidence in the image, this leads to an interesting conclusion: statistical uniformity is not the same as cryptographic security. The other two modules (random and secrets), as expected, passed with high p-values, confirming their excellent statistical properties.

But also, as we see, LCG's p-value of 0.1760 is sufficient to pass the test; however, it is substantially lower than the near-ideal p-values of the "random" and "secrets" modules. This suggests that the Chi-Squared test detected some deviation from uniformity, even though it was not statistically significant at the $\alpha = 0.05$ level.

6. DISCUSSION

The paper presents one of the most critical concepts in practical cryptography: statistical randomness does not equate to cryptographic security. From the experiments, we observe that the visually and structurally flawed Linear Congruential Generator (LCG) passed a standard test for randomness, which is the central finding of this study. A cybersecurity analyst relying solely on this quantitative metric might incorrectly conclude that the LCG is "good enough" for our application. However, the visual evidence in Figure 1 clearly demonstrates that the output is structured and predictable. Statistically, the LCG can deceive us by producing the correct frequency of each byte value to pass the frequency test, yet the output sequence remains highly predictable. In security, predictable sequences or patterns can be catastrophic for a company's protection, as even less predictable patterns can now be easily detected and decrypted by modern computers. The purpose of this experiment was to demonstrate that a single statistical test is insufficient to evaluate a generator's security, especially in the context of the forthcoming post-quantum era.

For the other two modules, we can note that the random module also passed; however, it is known to have a predictable internal state, a flaw not detected by this test. The secrets module is the only one of the three designed to be unpredictable, effectively resisting adversarial analysis.

7. CONCLUSION AND FUTURE WORK

Our experiment and its results demonstrated a powerful and useful illustration of the properties of pseudorandom number generators (PRNGs). The idea that a generator can be statistically perfect yet dangerously insecure serves as a forewarning for the future of cryptography and security. We can draw a metaphorical comparison to security: just as humans are often the most vulnerable element in security systems and cannot be fully relied upon, statistical results alone cannot guarantee security. We cannot conclusively determine that a model is secure solely because it has passed statistical tests. The visual artifacts of the linear congruential generator (LCG), despite its passing the Chi-Squared test, provide a compelling argument against using simple generators or relying on incomplete metrics for validation. In his famous book "The Art of Computer Programming", Knuth introduced one of the most important tests - the spectral test - as a method to evaluate the quality of LCG – PRNGs (Knuth 1997). The core idea of this test is that it bridges both empirical and theoretical evaluations. Consequently, this test reveals that LCGs fall into predictable patterns of hyperplanes, which become apparent in graphical applications.

The best way to verify the security of cryptographic modules or algorithms is to rely on an already tested suite designed for this purpose, such as NIST Special Publication 800-22 (Rukhin et al. 2010). This publication presents, validates, and concludes various random number and pseudorandom number tests, ranging from frequency tests to Maurer's "Universal Statistical" test and entropy tests. It also introduces its own testing strategy and guidelines, allowing anyone to validate and test their modules before implementing them in production. Although our experiment does not follow this publication step by step, it aligns with our main idea: randomness can be as illusory as security features. Therefore, in

our experiment, we validate the foundational security principle of using dedicated cryptographic tools like Python's secrets module - not because they are statistically "more random," but because they are designed to be unpredictable.

The main insight from this simple demonstration is that the concept of a statistical test can be illusory and does not provide a definitive answer regarding the security of a module. For future security developers, researchers, educators, and decision-makers, it is essential to enhance, improve, and experiment with the concept of randomness in cryptographic modules using various mathematical approaches. Novel methods already exist, such as employing Hamiltonian conservative chaotic systems for pseudorandom number generation (Patidar and Singh 2025) and quantum/post-quantum pseudorandom generators. These advancements offer researchers opportunities to build upon previous work and create a more secure future environment.

ETHICAL STATEMENT

This study does not contain any studies with human or animal subjects performed by any of the authors.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest to this work.

REFERENCES

1. Knuth, D. E. 1997. "Random Numbers." In *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Third Edition. Reading, MA, USA: Addison-Wesley.
2. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., and Vo, S. 2010. "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications." NIST Special Publication 800-22 Revision 1a. Gaithersburg, MD, USA: National Institute of Standards and Technology.
3. Schneier, B., Ferguson, N., and Kohno, T. 2010. "Randomness." In *Cryptography Engineering: Design Principles and Practical Applications*. Hoboken, NJ, USA: Wiley.
4. US-CERT. 2008. "Debian OpenSSL predictable random number generator." Vulnerability Note VU#970105.
5. Patidar, V., and Singh, T. 2025. "A novel approach to pseudorandom number generation using Hamiltonian conservative chaotic systems." *Front. Phys.* 13: 1553389. doi: 10.3389/fphy.2025.1553389.
6. Bhattacharjee, K., and Das, S. 2022. "A search for good pseudo-random number generators: Survey and empirical studies." *Computer Science Review* 45: 100471.