CAUCHY – Jurnal Matematika Murni dan Aplikasi
Volume 11 (2) (2026), Pages 2116–2137
p-ISSN: 2086-0382; e-ISSN: 2477-3344

Randomness Test of LightMAC-Based Pseudorandom Number Generator (PRNG) with NIST SP 800-22 Revision 1A Test

Bety Hayat Susanti[1*], Wahyu Achmad Fadiel[2], Yeni Farida[1], and Obrina Briliyant[3]

[1]*Department of Cryptographic Engineering, Politeknik Siber dan Sandi Negara, Indonesia*
[2]*Badan Siber dan Sandi Negara, Indonesia*
[3]*School of Computer Science & Informatics, Cardiff University, United Kingdom*

Abstract

Random numbers are essential for cryptographic security, yet resource-constrained devices lack lightweight random number generators, leaving encryption, authentication, and key generation vulnerable to exploitation. This study develops and statistically evaluates a Pseudo-Random Number Generator (PRNG) based on the LightMAC construction instantiated with the PRESENT lightweight block cipher. An initial seed message x is iteratively processed through LightMAC-PRESENT in a feedback (chaining) configuration. Randomness is analyzed using the NIST SP 800-22 Revision 1A test suite across 96 configurations, combining two seed lengths ($|x| = 64, 128$ bits), two key lengths (80, 128 bits), and three counter lengths ($s = 8, 16, 32$ bits), each tested under four key-pattern combinations in two scenarios. Of the 96 configurations, 67 (69.8%) satisfied all 15 NIST tests under both the proportion and P-value uniformity criteria. A 128-bit key yielded a higher pass rate (77.1%) than an 80-bit key (62.5%), and a shorter counter ($s = 8$) yielded the highest pass rate (81.3%). The failure distribution showed no dependence on key-pattern entropy, suggesting that most failures reflect statistical variance (Type I error) rather than structural weaknesses, though this was not formally established. Most failures (70.3%) concentrated in the Non-Overlapping Template Matching test, whose 148 sub-tests are inherently prone to false positives at $\alpha = 0.01$. A construction-specific limitation is also noted: since the feedback mechanism re-processes a fixed 64-bit state after the first iteration, its safe-output cycle length is bounded by the 64-bit state space rather than the seed space. These findings offer empirical, statistics-only guidance for parameter selection in this construction, without constituting a claim of cryptographic security.

Keywords: LightMAC; Lightweight cryptography; NIST SP 800-22; PRESENT block cipher; Pseudorandom number generator.

1. Introduction

Random numbers are essential across numerous domains, including cryptography, gaming, simulation, statistical sampling, decision-making, and the arts [1]. In cryptographic applications, the primary purpose of random numbers is to strengthen security during encryption and key establishment. Consequently, many cryptographic algorithms and protocols rely on random

*Corresponding author. E-mail: bety.hayat@poltekssn.ac.id

Received: June 9, 2026 Revised: August 20, 2026 Accepted: September 14, 2026
DOI: https://doi.org/10.18860/cauchy.v11i2.43424

sequences for reciprocal authentication, session key generation, and stream cipher initialization [2].

Random sequences are produced by Random Number Generators (RNGs), with Pseudorandom Number Generators (PRNGs) being the most widely used. A PRNG is a deterministic algorithm that expands a short, uniformly random seed into a longer pseudorandom bit sequence. PRNGs have been a central research topic for decades, leading to a diverse array of designs [2]. Stallings categorizes PRNGs into two main types: purpose-built algorithms and those constructed from existing cryptographic primitives. Among the latter, hash functions—both unkeyed (Modification Detection Code—MDC) and keyed (Message Authentication Code—MAC)—are frequently employed. A prominent example is the Keyed-Hash Message Authentication Code (HMAC) [3], standardized in ISO/IEC 9797-2 and FIPS 198 [4], and widely adopted across protocols and applications [2]. Moreover, HMAC serves as the architectural foundation for MAC-based Deterministic Random Bit Generators (DRBGs) specified in NIST SP 800-90A Rev. 1 [5]. In a MAC-based DRBG, the internal state comprises a value V and a key K; the key K remains constant across iterations, while V for each successive block is derived from the output of the preceding block [2].

In 2016, Luykx *et al.* introduced LightMAC, an efficient MAC mode of operation for block ciphers [6]. A key advantage of LightMAC is that its security bounds are independent of the message length, allowing it to process larger datasets without degradation. Because its security is directly tied to the underlying block cipher, LightMAC offers a simple, cost-effective alternative to other MAC modes and demonstrates superior performance in parallel implementations. Evaluations using PRESENT and AES as base ciphers confirm that LightMAC outperforms traditional block-cipher-based MAC modes in these aspects [6].

PRESENT is a lightweight substitution-permutation network with a 64-bit block size and 80-bit or 128-bit key options, specifically designed for RFID (Radio Frequency Identification) tags, sensor networks, and similar devices [7]. It is standardized in ISO/IEC 29192-2 as a lightweight cryptographic primitive [8]. Its compact hardware footprint and well-studied security properties make it an attractive choice for building lightweight cryptographic mechanisms. However, the raw PRESENT output does not, by itself, satisfy the statistical randomness requirements, failing several NIST SP 800-22 categories [9]. This gap motivates the construction-level treatment proposed in this study.

To be considered cryptographically useful, a pseudorandom sequence must satisfy two fundamental criteria: randomness (uniformity and statistical independence) and unpredictability. A PRNG's output cannot be assumed to be random *a priori*; it must undergo rigorous statistical validation [2]. The NIST Statistical Test Suite (STS), specified in NIST SP 800-22 Revision 1A [10], is one of the most comprehensive tools for this purpose, comprising 15 statistical tests that evaluate the uniformity, scalability, and consistency of bit sequences.

This study develops an iterative PRNG that uses the LightMAC construction instantiated with the PRESENT block cipher, hereafter referred to as the LightMAC-PRESENT-based PRNG. In this design, the initial seed message x is processed by LightMAC-PRESENT, and subsequent output blocks are generated by feeding the previous block's output back into the system as the new input message. We note explicitly that this feedback (chaining) formulation supersedes an external-counter formulation explored in an earlier draft of this work, in which a fixed seed was combined with an explicit, separately incremented counter at every iteration ($M_i = encode(i) \parallel x$). The feedback formulation was adopted because it admits a substantially simpler and more unambiguous algorithmic specification (Algorithm 1 in Section 3.1, which improves reproducibility; all experimental results reported in Section 4 and Section 5 of this manuscript were generated using the feedback formulation only. The central research question addressed is: *How random does the LightMAC-PRESENT-based PRNG generate the bit sequence according to the NIST STS?* To answer this, the 15 NIST tests are applied to the generator's output.

The following design choices bound the scope of the investigation: the LightMAC internal counter length s is set to 8, 16, or 32 bits; the initial seed message x uses lengths of 64 and 128 bits; the truncated output length t is fixed at 64 bits; and the significance level for all statistical tests is $\alpha = 0.01$. The hypotheses tested are H_0 (the output sequence is random) and H_a (the output sequence is not random), and the experimental results will determine which hypothesis is accepted.

Research Motivation

Resource-constrained IoT and embedded devices depend on lightweight PRNGs for secure cryptographic operations. A weak or unpredictable RNG directly threatens the strength of the cryptographic system's encryption, authentication, and key generation, exposing critical infrastructure to adversarial exploitation (e.g., key recovery, nonce reuse, and spoofing attacks). Although standardized deterministic random bit generators (DRBGs) such as Hash-DRBG, HMAC-DRBG, and CTR-DRBG have been specified in NIST SP 800-90A Rev. 1 [5], there is limited research on using lightweight message authentication code (MAC) constructions as the core primitive for pseudorandom bit generation. Existing lightweight PRNGs have primarily focused on block-cipher-based, hash-based, or stream-cipher-based architectures [11–14].

In contrast, LightMAC has mainly been investigated as a message authentication mechanism rather than as a building block for deterministic random bit generators. LightMAC nonetheless possesses properties that are directly relevant to this gap: security bounds independent of message length and efficient implementation on constrained hardware [6], making it a compelling candidate for lightweight PRNG design. This work, therefore, investigates whether a LightMAC-based construction can produce statistically random output suitable for resource-constrained cryptographic environments.

It is important to note that statistical randomness alone does not imply cryptographic security. Passing statistical tests such as NIST SP 800-22 indicates the absence of detectable statistical bias in the tested samples. Still, it does not establish unpredictability, indistinguishability from randomness, resistance against state compromise, or other cryptographic security properties. Therefore, this work focuses exclusively on the statistical evaluation of the proposed generator and does not claim a formal cryptographic security proof.

Contributions of This Work

This paper proposes and evaluates a novel LightMAC-based PRNG architecture instantiated with the PRESENT block cipher. The primary contributions are:

- **Empirical Statistical Evaluation of a Novel Construction:** The primary contribution of this work is a systematic, empirical statistical evaluation — under the NIST SP 800-22 test suite and 96 parameter configurations — of a deterministic PRNG built around the LightMAC mode of operation. No formal security proof or theoretical randomness bound is claimed; the construction itself is a natural application of LightMAC and PRESENT, and the contribution lies in the empirical characterization of its statistical behavior, which, to the authors' knowledge, has not previously been reported for a LightMAC-based generator.
- **Lightweight-Oriented Architecture:** By combining LightMAC with the PRESENT block cipher, the proposed PRNG is instantiated with the PRESENT lightweight block cipher, a primitive designed for hardware-constrained environments such as RFID and IoT.
- **Empirical Statistical Validation:** The proposed PRNG is rigorously evaluated using the NIST SP 800-22 Revision 1A test suite. The analysis demonstrates the impact of varying key lengths (80-bit and 128-bit) and internal counter sizes (8-bit, 16-bit, and 32-bit) on the statistical robustness of the generated sequences.

Scope and Limitations

This study evaluates the proposed LightMAC-PRESENT-based generator solely from the perspective of statistical randomness using the NIST SP 800-22 Revision 1A test suite. Consequently, the results should not be interpreted as proof of cryptographic security. In particular, the present work does not investigate:

- Next-bit unpredictability;
- Forward secrecy and backward secrecy;
- Resistance to state compromise attacks;
- Distinguishing attacks against the generator;
- Formal reductions from the security of LightMAC or PRESENT to pseudorandom generation;
- Hardware performance, energy consumption, or throughput comparisons against standardized DRBGs;

In addition, because the proposed construction feeds each 64-bit LightMAC-PRESENT output back as the input to the next iteration ($V_i = \text{LightMAC-PRESENT}^{s,t}_{K_1,K_2}(V_{i-1})$; Equation 2), the internal state processed by the generator is 64 bits wide for every iteration after the first, irrespective of the initial seed length $|x| \in \{64, 128\}$. By a standard birthday-bound heuristic for the iteration of a function over a state space of size 2^{64}, the expected cycle length before the sequence of V_i values repeats is on the order of 2^{32}. This comfortably exceeds the $\lceil 10^6/64 \rceil = 15,625$ blocks generated per 10^6-bit test sequence used in this study, and is therefore not a concern for the statistical evaluation reported here. It is, however, a materially shorter safe-output bound than would be obtained from an external-counter formulation (in which each output block is generated independently from a counter and the full, unconsumed seed), and should be taken into account before this specific construction is adopted for applications that require very long pseudorandom streams from a single seed. A formal analysis of the cycle structure of the feedback construction is left to future work. Therefore, the conclusions of this paper are restricted to statistical randomness properties rather than comprehensive cryptographic security guarantees.

2. Theoretical Background

2.1. Hash Function

A hash function is a cryptographic algorithm that transforms an input of arbitrary length into a fixed-length output, known as a hash value [2]. As described by Menezes *et al.* [15], a well-defined hash function h must possess at least two essential properties. First, it should provide compression, meaning that the function h maps any input x of arbitrary size to a fixed-length output $h(x)$ of size n. Second, it must allow ease of computation, where the process of computing $h(x)$ from a given input x and the function h itself is computationally efficient.

Hash functions can be broadly divided into two categories: unkeyed hash functions (Modification Detection Codes, MDC) and keyed hash functions (Message Authentication Codes, MAC). Unkeyed hash functions operate solely on the input message without requiring any secret key. In contrast, MACs incorporate a secret key along with the input message to generate the hash value [15]. The presence or absence of this secret key is the fundamental distinction between the two types.

2.2. Pseudorandom Number Generator (PRNG)

A PRNG is a critical component in cryptographic applications, generating random number sequences used for key distribution, mutual authentication schemes, session key generation, public-key encryption algorithms, and bit streams in stream ciphers [2].

According to Stallings [2], there are three cryptographic approaches to constructing a PRNG:

1. PRNGs based on block cipher algorithms.
2. PRNGs based on stream cipher algorithms.
3. PRNGs based on hash functions, which include unkeyed hash functions (MDC) and keyed hash functions (MAC).

For MAC-based PRNGs, two input parameters are required: a key K and a seed V. The combination of K and V forms the entire seed material for the PRNG, as described in NIST SP 800-90A Rev. 1 [5]. The MAC algorithm historically used in constructing PRNGs is HMAC. According to NIST SP 800-90A, MAC-based PRNGs offer greater security assurance compared to MDC-based PRNGs, albeit at approximately twice the computational cost. The basic structure of a MAC-based PRNG, which serves as the foundational structure for the PRNG in this study, is shown in Fig. 1.

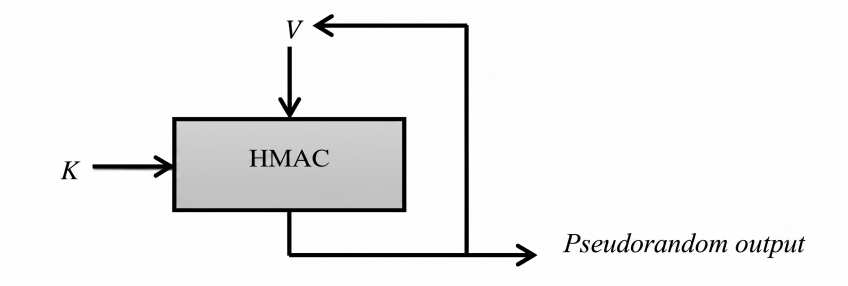

Fig. 1: Basic structure of a MAC-based pseudorandom number generator (PRNG). Figure redrawn by the author based on the conceptual description provided in [2].

2.3. PRESENT Algorithm

The PRESENT algorithm is a block cipher based on a Substitution-Permutation Network (SPN) structure and consists of 31 rounds. It has a block size of 64 bits and offers two key length variations: 80-bit and 128-bit. For practical applications, it is recommended to use the 80-bit key length, as it provides sufficient security for RFID-based applications. All discussions regarding the PRESENT algorithm are based on Bogdanov *et al.* [7].

2.4. LightMAC

All discussions about LightMAC refer to the work by Luykx *et al.* [6]. LightMAC is a mode of operation for Message Authentication Code (MAC) based on block cipher algorithms. Suppose E is a block cipher function, defined as $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$, where k is the key length and n is the block length. The LightMAC mode uses two independent keys, K_1 and K_2, derived from $\{0,1\}^k$, with the key length determined by the block cipher algorithm in use. The maximum length of the message M used as input in LightMAC is $2^s(n-s)$ bits, where s is an integer representing the bit length of the internal block counter. The output is denoted by T, with a length of t-bits, where $t \leq n$.

Specifically, the message M undergoes a padding procedure where a single '1' bit followed by the minimum number of '0' bits (10^d) is appended such that the total padded length becomes a multiple of $(n-s)$. The padded message is then divided into ℓ blocks, $M_1, M_2, \ldots, M_\ell$, each of length $(n-s)$ bits. During the multi-block processing phase, each message block M_i is concatenated with an s-bit block counter $\langle i \rangle_s$ to form an n-bit block $\langle i \rangle_s \parallel M_i$. This block is then encrypted using the first key K_1. The LightMAC finalization is achieved by computing the XOR sum of all these encrypted blocks and then encrypting the result with the second key K_2. The final MAC tag T is obtained by truncating the result to t bits. Fig. 2 illustrates the basic structure of the LightMAC scheme.

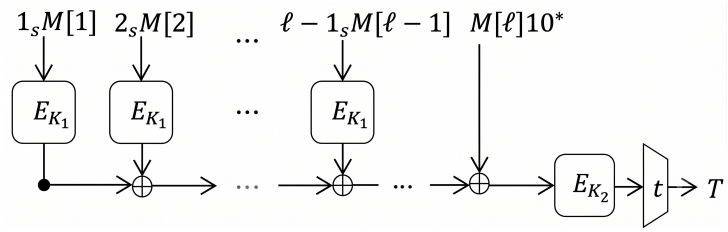

Fig. 2: LightMAC scheme. Figure redrawn by the author based on the description presented in [6].

For reproducibility, we note that the methodological description above, Algorithm 1, the accompanying source code, and the supplementary datasets (Section 3.4) are mutually consistent: the source code implements the padding rule, per-block counter concatenation, and dual-key (K_1, K_2) processing exactly as specified in this subsection, and the random and patterned input-generation routines described in Section 3.3 are implemented as a single shared module that is used, unmodified, to produce both the 1,000-sample sets analyzed in Section 4 and the corresponding datasets deposited alongside the code (Section 3.4). No divergent or ad-hoc implementation of LightMAC padding, counter handling, or key derivation was used at any stage of the experiments.

2.5. NIST Statistical Test Suite

The NIST Statistical Test Suite (STS), specified in NIST SP 800-22 Revision 1A [10], is a widely adopted methodology for evaluating the randomness of binary sequences produced by pseudorandom number generators. It comprises 15 statistical tests, each designed to detect specific deviations from randomness at a significance level α typically ranging from 0.001 to 0.1. A detailed description of each test is available in the NIST documentation [10]. For brevity, Table 1 summarizes the tests and their parameter constraints.

Table 1: NIST SP 800-22 Revision 1A test parameters

Test Name	n	m **or** M	**Sub-tests**
Frequency (Monobit)	≥ 100	–	1
Frequency within a Block	≥ 100	$[20, n/100]$	1
Runs	≥ 100	–	1
Longest Run of Ones	≥ 128	–	1
Binary Matrix Rank	≥ 38912	–	1
Discrete Fourier Transform	≥ 1000	–	1
Non-overlapping Template	$\geq 8m - 8$	$[2, 21]$	148*
Overlapping Template	$\geq 10^6$	–	1
Maurer's Universal	≥ 387840	–	1
Linear Complexity	$\geq 10^6$	$[500, 5000]$	1
Serial	–	$]2, \lfloor \log_2 n \rfloor - 2[$	2
Approximate Entropy	–	$< \lfloor \log_2 n \rfloor - 5$	1
Cumulative Sums	≥ 100	–	2
Random Excursions	$\geq 10^6$	–	8
Random Excursions Variant	$\geq 10^6$	–	18

*The Non-overlapping Template Matching test has 148 sub-tests for the default $m = 9$.

2.5.1. Interpretation of Test Results

The interpretation of test results in the NIST Statistical Test Suite (STS) employs two complementary approaches: the proportion of passing sequences and the uniformity of P-values. The

outcomes of both approaches are reported in the file `finalAnalysisReport.txt` upon completion of testing. If either or both of these approaches fail, leading to the rejection of the null hypothesis H_0 (the sequence is random), it is recommended that additional experiments be conducted using different samples from the PRNG to determine whether the observed phenomenon is a statistical anomaly or indicative of genuine non-randomness [10]. Furthermore, recent work by Luengo [16] provides a detailed coverage study of the NIST SP 800-22 test suite, emphasizing the importance of understanding the interdependence among the individual tests when interpreting combined results. Isolated single-criterion subtest failures, which are statistically expected at a rate of approximately 1% under $\alpha = 0.01$ when a large number of tests are performed simultaneously, are detailed in Table 4 and Table 8 for transparency.

Proportion of Passing Sequences. This approach examines the proportion of binary sequences that pass a given statistical test. If the proportion falls outside the acceptance region, the sequences are considered non-random. NIST SP 800-22 defines the acceptance region as

$$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}, \tag{1}$$

where $\hat{p} = 1 - \alpha$ and m is the number of binary sequences tested. We follow NIST's own notation here: the constant 3 is a fixed value specified directly in NIST SP 800-22 [10] as an approximation to a three-standard-deviation (approximately 99.7%, two-sided) normal confidence bound, and is used generically in the STS regardless of the particular α adopted for the individual tests; it is *not* the same as the significance-level-specific critical value $z_{\alpha/2}$ (for $\alpha = 0.01$, $z_{\alpha/2} = z_{0.005} \approx 2.576 \neq 3$). For $\alpha = 0.01$ and $m = 1000$, this yields the acceptance interval $[0.9805607, 0.9994392]$ as used throughout Section 4.

Uniformity of P-values. This approach evaluates the distribution of P-values to assess their uniformity. The P-values are divided into 10 sub-intervals, and the uniformity is computed using the Chi-square statistic:

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - s/10)^2}{s/10},$$

where F_i is the number of P-values in the i-th sub-interval, and s is the total number of samples. The total P-value is then calculated as:

$$P\text{-value}_T = \text{igamc}\left(\frac{9}{2}, \frac{\chi^2}{2}\right).$$

If $P\text{-value}_T \geq 0.0001$, the P-values are considered uniformly distributed, and the sequence passes the test. Conversely, if $P\text{-value}_T < 0.0001$, the P-values are not uniformly distributed, indicating non-randomness in the sequence for that test.

2.6. Research Gap

Lightweight cryptographic primitives and PRNGs have attracted significant research attention in resource-constrained IoT environments, where conventional cryptographic algorithms can impose impractical computational and memory overheads. A foundational lightweight MAC design was proposed by Luykx *et al.* [6], which addresses a critical limitation of conventional MAC modes: their security bounds degrade with both message count and length, making them impractical for lightweight block ciphers with small block sizes (32-64 bits). LightMAC resolves this by ensuring that security bounds are independent of message length through a protected counter-sum approach using two independent keys. Implemented with PRESENT and AES, LightMAC demonstrated approximately 50% faster performance than MAC with Parity for PRESENT and 20% faster for AES.

Building on LightMAC's theoretical foundations, Saldamli *et al.* [17] evaluated the practical performance of LightMAC alongside CHASKEY on an STM32F401RE microcontroller, measuring execution time, memory consumption, power consumption, and throughput across key sizes of 128 and 192 bits, and message lengths ranging from 16 to 3072 bytes. The results strongly favored LightMAC, which achieved execution times approximately 100 times faster, power consumption approximately 1000 times lower, and throughput approximately 1000 times higher than CHASKEY, establishing LightMAC as a practical and efficient authentication primitive for constrained IoT deployments.

Meanwhile, the randomness quality of PRESENT-based PRNGs has been shown to have notable limitations. Mohammad Shah and Ismail [9] conducted a NIST SP 800-22 statistical analysis of PRESENT used directly as a random number generator and found the outputs to be essentially non-random across six data categories, highlighting that raw block-cipher outputs do not inherently satisfy statistical randomness requirements. Similarly, Susanti *et al.* [14] evaluated a Davies-Meyer scheme instantiated with PRESENT-80 and PRESENT-128, generating 1,000 samples of 1,000,000 bits each across multiple input types. Of the 48 tested samples, 17 (35%) failed the NIST test suite, with 12 failures concentrated in the Non-Overlapping Template Matching test, suggesting susceptibility to repeating and non-periodic patterns under certain input configurations.

Beyond PRESENT-based constructions, several alternative lightweight PRNG architectures have demonstrated compliance with NIST SP 800-22. Kaya and Ince [18] evaluated nine chaotic maps and identified five as passing all NIST SP 800-22 tests. The tests show that the Gauss Map additionally offers quantum resistance, alongside the fastest execution among the compliant designs. Sivaraman *et al.* [19] demonstrated a Tinkerbell map-based PRNG that satisfied NIST SP 800-22 randomness requirements while consuming only 0.73 mW and less than 1% hardware utilization, underscoring the viability of chaotic map-based generators in ultra-low-power deployments. Sandakly *et al.* [20] validated XorshiftH128+ against NIST SP 800-22 alongside additional test suites, while Cabuk *et al.* [21] noted that xorshiftR+ demonstrated strong suitability for constrained environments but required further optimization to balance security guarantees, execution speed, and resource consumption.

The literature reveals that while LightMAC offers compelling efficiency advantages for IoT authentication, its direct application as a PRNG construction has not been systematically evaluated against the NIST SP 800-22 test suite. Existing PRESENT-based PRNG evaluations rely on simpler constructions such as Davies-Meyer, which exhibit significant failure rates. This work addresses the gap by proposing and rigorously evaluating a LightMAC-PRESENT-based PRNG across multiple parameter configurations, providing empirical guidance for statistically robust random bit generation in resource-constrained environments.

3. Research Methodology

This study employs an experimental method to evaluate a LightMAC-PRESENT-based Pseudo-random Number Generator (PRNG) implemented in Python. The theoretical research population encompasses 2^{64} and 2^{128} possible values for the seed message x, and 2^{80} and 2^{128} for keys K_1 and K_2. Due to computational resource constraints, a simple random sampling technique is applied to draw 1,000 samples per configuration. This sample size is consistent with common practice in the NIST SP 800-22 literature and with the guidance in [10], which recommends generating a sufficiently large number of sequences – typically on the order of $m = 1{,}000$ – so that the proportion-of-passing-sequences criterion (Equation 1) has adequate statistical resolution; NIST SP 800-22 does not itself mandate an exact minimum sample count. We adopt $m = 1{,}000$ sequences per configuration accordingly, together with a sequence length of 10^6 bits per sequence, which satisfies the minimum input-length requirement of every individual test in Table 1, including the most demanding ones (Overlapping Template Matching, Maurer's Universal, Linear Complexity, Random Excursions, and Random Excursions Variant, each requiring $n \geq 10^6$ or

$\geq$ 387,840 bits).

3.1. PRNG Construction Details

Let E_K denote the PRESENT block cipher with a 64-bit block size and a key length $k \in \{80, 128\}$ bits. The LightMAC construction utilizes two independent keys $K_1, K_2 \in \{0,1\}^k$ and an internal block counter of length s (set to 8, 16, or 32 bits). In this proposed design, the PRNG iteratively feeds the MAC output of the previous iteration back as the input message for the current iteration.

Let the initial seed message x be denoted as $V_0 = x$, where the length of x is either 64 or 128 bits. For each iteration $i \geq 1$, the generator computes the i-th 64-bit output block V_i as:

$$V_i = \text{LightMAC-PRESENT}^{s,t}_{K_1,K_2}(V_{i-1}), \tag{2}$$

where $t = 64$ bits is the truncation length (the entire 64-bit LightMAC tag is retained). To generate a required sequence of length L bits (e.g., $L = 10^6$ in our experiments), $N = \lceil L/64 \rceil$ consecutive output blocks are generated and concatenated. The entire generation uses a single fixed key pair (K_1, K_2).

Note that the LightMAC internal counter i_s is used during the processing of each message V_{i-1} according to the standard LightMAC specification, which natively handles internal message block indexing and padding. For all tested values of s (8, 16, 32), the message lengths of 64 or 128 bits are well within the allowed LightMAC limit of $2^s(64 - s)$ bits, ensuring that no internal counter wrap-around occurs. The generation procedure is explicitly defined in Algorithm 1.

Algorithm 1: LightMAC-PRESENT PRNG Generation

Input: Seed $x \in \{0,1\}^{64} \cup \{0,1\}^{128}$, Keys $K_1, K_2 \in \{0,1\}^k$ where $k \in \{80, 128\}$, Requested length L

Output: Pseudorandom bit sequence S of length L

1: $V_0 \leftarrow x$
2: $S \leftarrow$ empty string
3: $N \leftarrow \lceil L/64 \rceil$
4: **for** $i = 1$ **to** N **do**
5: $\quad V_i \leftarrow \text{LightMAC-PRESENT}^{s,64}_{K_1,K_2}(V_{i-1})$
6: $\quad S \leftarrow S \parallel V_i$
7: **end for**
8: **return** First L bits of S

3.2. Experimental Design

The experimental design utilizes two distinct scenarios to assess the randomness of the generated 10^6-bit output sequences, which serve as the dependent variable. As detailed in Table 2, the scenarios manipulate the independent and control variables in different ways.

Scenario 1: Evaluates the effect of the pseudorandom seed message x on output randomness. The seed message x is the independent variable (1,000 samples), while K_1 and K_2 are control variables (fixed single values: one random and one patterned), tested across all four combinations in Table 3.

Scenario 2: Evaluates the effect of key variations on the randomness of the output. A single seed message x is fixed as the control variable, while K_1 and K_2 are the independent variables (1,000 samples each, with both random and patterned values tested across all four combinations).

The specific combinations of random and patterned values applied to K_1 and K_2 across both scenarios are outlined in Table 3. For each combination, exactly 1,000 sequences of 10^6 bits are generated to match the NIST sample size requirements.

Table 2: Size and number of samples for each experimental scenario

No.	Variable	Size (bit)	Scenario 1	Scenario 2
			Total Sample	Total Sample
1.	Seed message x	64 128	1,000	1
2.	K_1	80 128	1	1,000
3.	K_2	80 128	1	1,000
4.	Output sequence	1,000,000	1,000	1,000

Table 3: Variations of key input values K_1 and K_2

K_1	K_2
Random	Random
Random	Patterned
Patterned	Random
Patterned	Patterned

3.3. Generation of Random and Patterned Inputs

Two categories of inputs are used throughout the experiments.

1. **Random inputs**, generated using Python's cryptographically secure random source.
2. **Patterned inputs**, generated using a deterministic repeating bit pattern "10101010..." (repeated to fill the required key length of 80 or 128 bits). This pattern is intended to represent low-entropy, non-random key material for stress-testing the robustness of the proposed generator under highly structured input conditions. It is not meant to model realistic cryptographic keys, which should be drawn from high-entropy sources.

3.4. Data Processing and Analysis

Data generation and processing are executed in Python on a system equipped with an Intel Celeron 1000M 1.80 GHz processor and 1.89 GB of RAM. The generated 10^6-bit sequences are subsequently evaluated using the official NIST Statistical Test Suite software (sts-2.1.2). For long-term reproducibility, the complete Python source code for the proposed LightMAC-PRESENT PRNG, the random/patterned input-generation module described in Section 3.3, all $96 \times 1{,}000$ generated sequences, and the complete `sts-2.1.2 finalAnalysisReport.txt` output for every one of the 96 configurations (including the exact proportion of passing sequences and $P\text{-value}_T$ for all 15 tests, 188 sub-tests in total) have been deposited in the persistent, versioned Zenodo repository https://doi.org/10.5281/zenodo.22023993 [22].

The statistical analysis evaluates the results across all 15 NIST tests using both the proportion and P-value uniformity approaches. A test sequence is considered to pass the randomness criteria if its P-value $\geq \alpha = 0.01$; otherwise, it is deemed to have failed. The study compares the outcomes of the two scenarios to determine the extent to which key configurations affect the PRNG's compliance with established randomness standards.

The overall research procedure is conducted in seven sequential stages: (1) literature review on hash functions, MAC, LightMAC, the PRESENT algorithm, and PRNGs; (2) implementation of the LightMAC-PRESENT-based PRNG in Python according to the construction defined in Eq. (2); (3) generation of random and patterned samples for x, K_1, and K_2; (4) processing

of inputs to generate 10^6-bit output sequences; (5) execution of the NIST statistical tests; (6) comparative data analysis based on the test results; and (7) formulation of conclusions regarding the PRNG's randomness performance.

4. Results and Discussion

This section presents the interpretation of the NIST Statistical Test Suite results applied to the output sequences generated by the LightMAC-PRESENT-based PRNG. The evaluation encompasses two seed message lengths ($|x| = 64$-bit and 128-bit), two key lengths (80-bit and 128-bit), and three internal counter lengths ($s = 8$, 16, and 32 bits), tested under four key-pattern combinations across two experimental scenarios. Overall, of the 96 evaluated configurations, 67 (69.8%) satisfied all 15 NIST tests under both the proportion and P-value uniformity criteria. The subsequent sections present a detailed breakdown of these results.

4.1. Interpretation of Isolated Test Failures

Note that the NIST SP 800-22 test suite includes numerous individual statistical tests and subtests. When a large number of tests are performed simultaneously using a significance level of $\alpha = 0.01$, occasional failures may occur purely due to statistical fluctuations. This consideration is particularly relevant for the Non-Overlapping Template Matching Test, which contains 148 individual template tests under the default configuration. Consequently, isolated failures in individual templates should be interpreted cautiously and not automatically regarded as evidence of fundamental weaknesses in the generator. This perspective aligns with the coverage study of the NIST test suite by Luengo [16], which highlights the statistical interdependence among the individual tests and reinforces the need for a holistic interpretation.

In accordance with the NIST SP 800-22 methodology [10], a test is considered failed if *either* the proportion of passing sequences falls outside the acceptance interval defined in Eq. (1) *or* the P-value uniformity criterion yields $P\text{-value}_T < 0.0001$. A configuration is declared to have failed a test if either criterion is not satisfied.

4.2. NIST Test Results for 64-bit Seed Message x

The bit sequences obtained from the LightMAC-PRESENT-based PRNG with a 64-bit seed message x and key variations K_1 and K_2 of 80-bit and 128-bit sizes were subjected to NIST statistical testing. Test failures were observed under the proportion approach and the P-value uniformity approach across the tested internal counter lengths. Table 4 details the failed test results.

4.2.1. Proportion of Passing Sequences

Of the 48 evaluated configurations utilizing a 64-bit seed message x, 31 (64.6%) satisfied all 15 NIST tests under both the proportion and P-value uniformity criteria. As shown in Table 5, the key length strongly influences statistical performance: configurations using a 128-bit PRESENT key achieved higher aggregate pass rates than those using 80-bit keys. A shorter internal counter ($s = 8$ bits) generally yielded the highest pass rates, consistent with the interpretation that a smaller counter leaves more message bits per block for diffusion.

4.2.2. P-value Uniformity

To illustrate the distinction between a failed and a passed test under the P-value uniformity criterion, Fig. 3 and Fig. 4 present the P-value histograms for a representative failed Non-Overlapping Template sub-test and a passed Frequency test, respectively, both drawn from the same configuration (Scenario 1, $|x| = 64$, $k = 80$, $s = 8$, random–random keys).

Table 4: Failed NIST test samples for 64-bit seed message x

Key Size	Counter (s)	Scenario / Sample (K_1–K_2)	NIST Test	Remark
80-bit	8-bit	Scen. 1: random–random	Non-Overlapping Template (pattern 111010010)	Failed P-value uniformity
		Scen. 1: random–random	Random Excursions	Failed proportion
		Scen. 1: random–patterned	Non-Overlapping Template (pattern 111111010)	Failed proportion
		Scen. 2: patterned–random	Random Excursions Variant	Failed proportion
80-bit	16-bit	Scen. 1: patterned–random	Non-Overlapping Template (pattern 000000001)	Failed proportion
		Scen. 1: patterned–random	Non-Overlapping Template (pattern 000010111)	Failed proportion
		Scen. 1: patterned–random	Non-Overlapping Template (pattern 100000000)	Failed proportion
		Scen. 2: random–random	Random Excursions	Failed proportion
80-bit	32-bit	Scen. 1: random–patterned	Discrete Fourier Transform	Failed proportion
		Scen. 1: patterned–random	Non-Overlapping Template (pattern 010110111)	Failed P-value uniformity
		Scen. 1: patterned–random	Non-Overlapping Template (pattern 101001000)	Failed P-value uniformity
		Scen. 1: patterned–patterned	Non-Overlapping Template (pattern 110100010)	Failed proportion
		Scen. 2: random–patterned	Non-Overlapping Template (pattern 000110101)	Failed proportion
		Scen. 2: patterned–patterned	Non-Overlapping Template (pattern 000001101)	Failed proportion
		Scen. 2: patterned–patterned	Non-Overlapping Template (pattern 000100101)	Failed proportion
128-bit	16-bit	Scen. 1: random–random	Discrete Fourier Transform	Failed proportion
		Scen. 1: patterned–patterned	Non-Overlapping Template (pattern 111011010)	Failed proportion
		Scen. 2: random–random	Non-Overlapping Template (pattern 000010001)	Failed proportion
128-bit	32-bit	Scen. 1: random–random	Non-Overlapping Template (pattern 111001100)	Failed proportion
		Scen. 1: patterned–random	Maurer's Universal	Failed proportion
		Scen. 1: patterned–patterned	Non-Overlapping Template (pattern 110111000)	Failed P-value uniformity
		Scen. 2: patterned–random	Non-Overlapping Template (pattern 000011011)	Failed proportion
		Scen. 2: patterned–random	Non-Overlapping Template (pattern 000101101)	Failed proportion

Note: A single configuration may fail multiple tests simultaneously.

Table 5: Aggregate NIST SP 800-22 pass rates by individual parameter values (64-bit seed message x configurations only)

Parameter	Value	Passed	Total	Pass Rate (%)
Key k	80-bit	14	24	58.3
	128-bit	17	24	70.8
Counter s	8-bit	13	16	81.2
	16-bit	11	16	68.8
	32-bit	7	16	43.8
Key pattern	Random–Random (a)	7	12	58.3
	Random–Patterned (b)	9	12	75.0
	Patterned–Random (c)	7	12	58.3
	Patterned–Patterned (d)	8	12	66.7

4.2.3. Test Analysis for 64-bit Input

As shown in Table 6, the observed failures are heavily concentrated in a small subset of the 15 NIST tests. The analysis of the 64-bit input results, based on the purpose of each statistical test, is as follows:

1. **Frequency (Monobit) Test:** All samples passed, indicating a balanced ratio of 0s and 1s.
2. **Block Frequency Test:** All samples passed, confirming that the number of 1s in M-bit blocks approximates $M/2$.
3. **Runs Test:** All samples passed, indicating that the oscillation between 0s and 1s is neither too fast nor too slow.
4. **Longest Run of Ones:** All samples passed, showing consistency with expected random behavior.
5. **Binary Matrix Rank Test:** All samples passed, suggesting no detectable linear dependencies among fixed-length substrings.
6. **Discrete Fourier Transform (Spectral) Test:** Most samples passed. Two samples

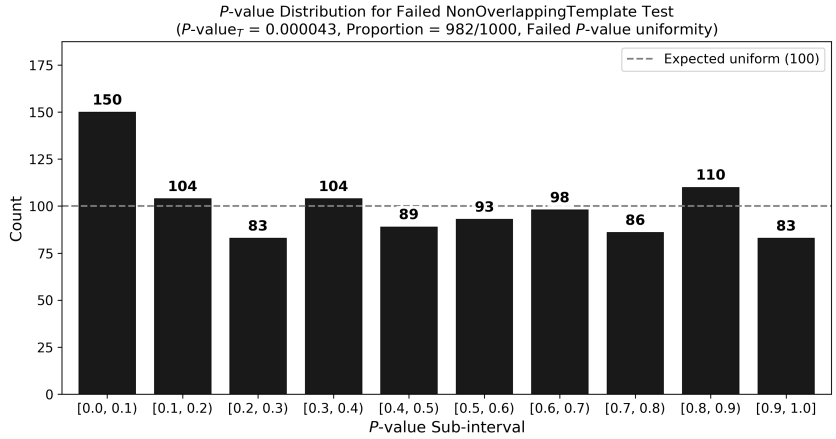

Fig. 3: P-value distribution for a failed Non-Overlapping Template sub-test, showing non-uniform concentration that yields $P\text{-value}_T < 0.0001$

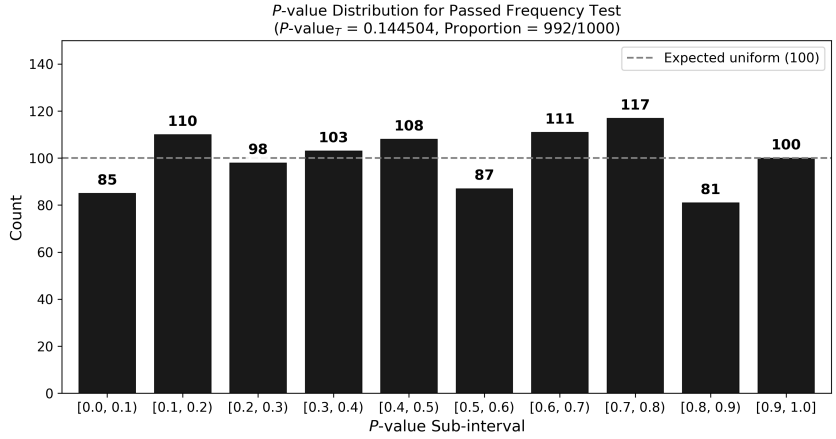

Fig. 4: P-value distribution for the passed Frequency test from the same configuration, showing the approximately uniform distribution expected under H_0

failed (e.g., Scenario 1, 128-bit key, $s = 16$ bits, random–random), suggesting the presence of closely repeating periodic patterns in those specific sequences.

7. **Non-Overlapping Template Matching Test:** Several samples failed (e.g., Scenario 1, 80-bit key, $s = 8$ bits, random–random), indicating an excessive occurrence of specific aperiodic patterns. This test accounts for the vast majority of all failure instances.
8. **Overlapping Template Matching Test:** All samples passed.
9. **Maurer's Universal Statistical Test:** Most samples passed. One sample failed (Scenario 1, 128-bit key, $s = 32$ bits, patterned–random), indicating that the output sequence exhibits detectable compressibility under this specific test and sample; this is a sample-level statistical outcome, not a general claim about the generator's compressibility.
10. **Linear Complexity Test:** All samples passed the Linear Complexity Test at $\alpha = 0.01$, i.e., no statistically significant deviation from the expected linear-complexity profile was detected in the tested samples. This indicates the absence of a detectable short linear-feedback-shift-register-equivalent structure in those samples; it is a sample-level statistical finding and not a general guarantee of algorithmic complexity or security.
11. **Serial Test:** All samples passed, showing overlapping m-bit patterns that match random expectations.
12. **Approximate Entropy Test:** All samples passed the Approximate Entropy Test at $\alpha = 0.01$, indicating no statistically detectable regularity in overlapping m-bit patterns for

the tested samples. As with all NIST SP 800-22 outcomes reported in this study, this is a statistical, sample-level finding rather than a general irregularity or security property of the generator.

13. **Cumulative Sums (Cusums) Test:** All samples passed.
14. **Random Excursions Test:** Most samples passed. Two samples failed (e.g., Scenario 2, 80-bit key, $s = 16$ bits, random–random), indicating unbalanced cycles at specific states.
15. **Random Excursions Variant Test:** Most samples passed. One sample failed (Scenario 2, 80-bit key, $s = 8$ bits, patterned–random), indicating unbalanced cycles at specific states.

Notably, 9 of the 15 NIST tests produced *zero* failures across all 64-bit configurations, indicating that the PRNG exhibits strong global statistical properties.

Table 6: Distribution of individual NIST SP 800-22 test failures across evaluated 64-bit seed message x configurations

NIST Test	Failure Instances
Non-Overlapping Template	17
Random Excursions	2
Random Excursions Variant	1
Discrete Fourier Transform	2
Cumulative Sums	0
Maurer's Universal	1
Remaining 9 tests	0

Note: A single configuration may fail multiple tests simultaneously.

The high failure rate in the Non-Overlapping Template Matching test must be analyzed in context. This test comprises 148 independent sub-tests under the default $m = 9$ configuration. In multiple hypothesis testing, the Family-Wise Error Rate (FWER) measures the probability of making at least one Type I error (false positive) among all tests [23]. Assuming independent tests at a significance level of $\alpha = 0.01$, the FWER across 148 sub-tests is:

$$\text{FWER} = 1 - (1 - \alpha)^{148} = 1 - 0.99^{148} \approx 0.773. \tag{3}$$

Thus, even for a perfectly random source, there is approximately a 77.3% probability of at least one Non-Overlapping Template sub-test failure per configuration (Equation 3).

A central question in this evaluation is whether the use of patterned (low-entropy) keys systematically degrades the PRNG's statistical quality. As reported in Table 5, the random–random key pattern (a) exhibited a pass rate of only 58.3%, while the patterned–patterned variant (d) achieved a 66.7% pass rate. This counterintuitive result is consistent with the interpretation that the observed failures are not primarily due to low key entropy, but rather reflect the inherent statistical variance of hypothesis testing at $\alpha = 0.01$ (cf. Equation 3); we present this as a plausible statistical explanation rather than a demonstrated causal finding, since disentangling Type I error from any residual, smaller entropy-related effect would require a dedicated, higher-power experiment (e.g., substantially more than 12 samples per key-pattern cell) that is beyond the scope of the present study.

Table 7 summarizes, for every 64-bit-seed configuration, whether all 15 NIST tests were jointly satisfied ("Pass") or at least one test failed ("Fail"), broken down by key length, counter length, scenario, and key pattern; this table is referred back to in the comparison discussion of Section 4.4.1.

4.3. NIST Test Results for 128-bit Seed Message x

Testing the PRNG with a 128-bit seed message x (with 80-bit and 128-bit keys, and $s = 8$, 16, and 32 bits) also yielded several failures under both the proportion and P-value uniformity

Table 7: Summary of NIST SP 800-22 test results for 64-bit seed message x configurations. "Pass" indicates all 15 tests satisfied the criteria; "Fail" indicates at least one test failed.

Key	Counter (s)	Scen.	Result by Key Pattern (K_1–K_2)			
			a	b	c	d
80-bit	8-bit	1	Fail	Fail	Pass	Pass
		2	Pass	Pass	Fail	Pass
	16-bit	1	Pass	Pass	Fail	Pass
		2	Fail	Pass	Pass	Pass
	32-bit	1	Pass	Fail	Fail	Fail
		2	Pass	Fail	Pass	Fail
128-bit	8-bit	1	Pass	Pass	Pass	Pass
		2	Pass	Pass	Pass	Pass
	16-bit	1	Fail	Pass	Pass	Fail
		2	Fail	Pass	Pass	Pass
	32-bit	1	Fail	Pass	Fail	Fail
		2	Pass	Pass	Fail	Pass

approaches. Table 8 details the failed test results.

Table 8: Failed NIST test samples for 128-bit seed message x

Key Size	Counter (s)	Scenario / Sample (K_1–K_2)	NIST Test	Remark
80-bit	8-bit	Scen. 1: random–random	Non-Overlapping Template (pattern 000111011)	Failed proportion
		Scen. 1: random–random	Random Excursions Variant	Failed proportion
		Scen. 2: random–patterned	Non-Overlapping Template (pattern 001100111)	Failed proportion
80-bit	16-bit	Scen. 1: random–patterned	Random Excursions Variant	Failed P-value uniformity
		Scen. 1: patterned–random	Non-Overlapping Template (pattern 100010000)	Failed proportion
		Scen. 1: patterned–patterned	Non-Overlapping Template (pattern 001000011)	Failed proportion
		Scen. 2: random–random	Non-Overlapping Template (pattern 110010000)	Failed proportion
80-bit	32-bit	Scen. 2: random–random	Random Excursions	Failed proportion
		Scen. 2: random–patterned	Discrete Fourier Transform	Failed proportion
128-bit	8-bit	Scen. 1: random–random	Non-Overlapping Template (pattern 000011101)	Failed proportion
		Scen. 1: random–random	Non-Overlapping Template (pattern 000101101)	Failed proportion
128-bit	16-bit	Scen. 1: random–random	Cumulative Sums	Failed proportion
		Scen. 1: patterned–patterned	Non-Overlapping Template (pattern 001001101)	Failed proportion
128-bit	32-bit	Scen. 1: random–random	Non-Overlapping Template (pattern 001101111)	Failed proportion

Note: A single configuration may fail multiple tests simultaneously.

4.3.1. Test Analysis for 128-bit Input

Of the 48 evaluated configurations utilizing a 128-bit seed message x, 36 (75.0%) satisfied all 15 NIST tests. As shown in Table 9, the 128-bit key configurations achieved a higher pass rate (83.3%) compared to the 80-bit key (66.7%), consistent with the trend observed for the 64-bit seed message.

Similar to the 64-bit input, the Non-Overlapping Template Matching Test was the most frequent point of failure, predominantly under the proportion approach (Table 10). Focusing on the failed tests for the 128-bit input, the analysis reveals the following:

1. **Discrete Fourier Transform (Spectral) Test:** One sample failed (Scenario 2, 80-bit key, $s = 32$ bits, random–patterned), indicating closely repeating periodic patterns.
2. **Non-Overlapping Template Matching Test:** Multiple samples failed, indicating excessive occurrences of specific aperiodic patterns.
3. **Cumulative Sums (Cusums) Test:** One sample failed (Scenario 1, 128-bit key, $s = 16$ bits, random–random), indicating an excessive accumulation of 0s or 1s.

Table 9: Aggregate NIST SP 800-22 pass rates by individual parameter values (128-bit seed message x configurations only)

Parameter	Value	Passed	Total	Pass Rate (%)
Key k	80-bit	16	24	66.7
	128-bit	20	24	83.3
Counter s	8-bit	13	16	81.2
	16-bit	10	16	62.5
	32-bit	13	16	81.2
Key pattern	Random–Random (a)	6	12	50.0
	Random–Patterned (b)	9	12	75.0
	Patterned–Random (c)	11	12	91.7
	Patterned–Patterned (d)	10	12	83.3

4. **Random Excursions Test:** One sample failed (Scenario 2, 80-bit key, $s = 32$ bits, random–random), indicating unbalanced cycles at specific states.
5. **Random Excursions Variant Test:** Two samples failed (e.g., Scenario 1, 80-bit key, $s = 8$ bits, random–random), indicating unbalanced cycles.

The random–random key pattern (a) exhibited a lower pass rate (50.0%) compared to the patterned–random variant (c), which achieved a near-perfect 91.7% pass rate (Table 9). As with the 64-bit results above, we regard this as consistent with, though not proof of, the interpretation that configuration-level failures are dominated by statistical (Type I) variance rather than by genuine structural deficiencies triggered by low-entropy keys; a dedicated, adequately powered follow-up experiment would be needed to confirm this explanation rather than an alternative, smaller entropy-related effect.

Table 10: Distribution of individual NIST SP 800-22 test failures across evaluated 128-bit seed message x configurations

NIST Test	Failure Instances
Non-Overlapping Template	9
Random Excursions	1
Random Excursions Variant	2
Discrete Fourier Transform	1
Cumulative Sums	1
Maurer's Universal	0
Remaining 9 tests	0

Note: A single configuration may fail multiple tests simultaneously.

4.4. Comparison of PRNG Implementations

4.4.1. Comparison for 64-bit Seed Message x

Analyzing the 64-bit seed message x configurations (Table 7), the most robust implementation is the 128-bit key with $s = 8$ bits, which passed all tests regardless of the key pattern in both experimental scenarios. However, the overall pass rate of the 64-bit seed message x remains lower than that of the 128-bit seed, highlighting a comparative disadvantage in statistical robustness for the shorter seed length in this dataset.

4.4.2. Comparison for 128-bit Seed Message x

For the 128-bit seed message x (Table 11), several implementations stand out. The configurations utilizing a 128-bit key with either $s = 8$ bits or $s = 32$ bits demonstrated excellent resilience,

Table 11: Summary of NIST SP 800-22 test results for 128-bit seed message x configurations. "Pass" indicates all 15 tests satisfied the criteria; "Fail" indicates at least one test failed.

Key	Counter (s)	Scen.	Result by Key Pattern (K_1–K_2)			
			a	b	c	d
80-bit	8-bit	1	Fail	Pass	Pass	Pass
		2	Pass	Fail	Pass	Pass
	16-bit	1	Pass	Fail	Fail	Fail
		2	Fail	Pass	Pass	Pass
	32-bit	1	Pass	Pass	Pass	Pass
		2	Fail	Fail	Pass	Pass
128-bit	8-bit	1	Fail	Pass	Pass	Pass
		2	Pass	Pass	Pass	Pass
	16-bit	1	Fail	Pass	Pass	Fail
		2	Pass	Pass	Pass	Pass
	32-bit	1	Fail	Pass	Pass	Pass
		2	Pass	Pass	Pass	Pass

both passing tests across almost all variables and key patterns. These results indicate that the longer 128-bit seed length enhances the statistical stability of the generated sequences, allowing multiple counter lengths to achieve near-optimal uniformity.

4.4.3. Analysis of Observed Failures and Parameter Sensitivity

Fig. 5 visualizes the pass rate for each of the 12 parameter configurations across both seed lengths. The failure patterns indicate that the PRNG's statistical behavior is sensitive to parameter choices.

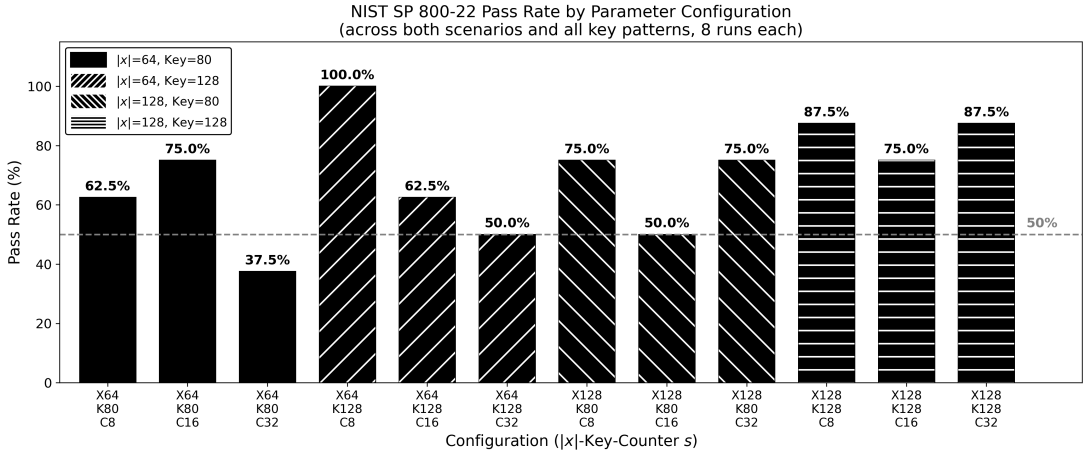

Fig. 5: NIST SP 800-22 pass rate by parameter configuration, computed across both experimental scenarios and all four key-pattern combinations (8 runs per configuration)

Configurations utilizing a 128-bit key consistently outperformed those using an 80-bit key. For instance, among 64-bit seed message x configurations, setups utilizing an 80-bit key achieved an aggregate pass rate of only 58.3%, whereas those using a 128-bit key achieved a higher pass rate of 70.8% (Table 5). This performance gap was similarly evident in the 128-bit seed configurations, where the 128-bit key achieved an 83.3% pass rate compared to 66.7% for the 80-bit key (Table 9). This observation aligns with the expectation that a larger key size provides better diffusion quality within the underlying PRESENT block cipher.

Regarding the internal counter size s, the data reveals a distinct sensitivity. Configurations

with a shorter counter ($s = 8$) provided the most robust overall statistical performance, achieving an 81.3% aggregate pass rate across all tests. Conversely, allocating a larger portion of the LightMAC block to the internal counter generally resulted in lower pass rates. For instance, increasing the counter to $s = 32$ caused the pass rate to plummet to 43.8% for the 64-bit seed configurations (Table 5). However, it recovered to 81.2% for the 128-bit seed configurations (Table 9) due to the stronger initial mixing. This suggests that reserving more message bits for state mixing per LightMAC iteration is highly beneficial for statistical uniformity, particularly when the initial seed is short.

Regarding the effect of seed length: because the feedback construction re-processes only a fixed 64-bit input from the second output block onward ($V_i = \text{LightMAC-PRESENT}(V_{i-1})$ for $i \geq 2$), the initial seed length $|x|$ can, by construction, directly influence only the very first state transition, $V_0 = x \rightarrow V_1$. We therefore interpret the observed 64-bit-vs-128-bit seed pass-rate gap (64.6% vs. 75.0% aggregate) as most plausibly reflecting a stronger avalanche/mixing effect at this first transition when x is 128 bits rather than 64 bits, with that improved initial mixing then propagating through the subsequent feedback chain. We emphasize that this is a plausible mechanistic interpretation consistent with the aggregate pass-rate data, not a result that is directly demonstrated by a dedicated diffusion or avalanche analysis; such an analysis is left to future work. An alternative, non-mechanistic explanation — that the gap partly reflects sampling variance across the 48 configurations tested per seed length — cannot be ruled out with the present data and should be kept in mind when interpreting Table 5 versus Table 9.

4.4.4. Comparison with Baseline Constructions

To situate the LightMAC-PRESENT PRNG within the existing literature, we compare its statistical performance descriptively against constructions reported in prior work. These comparisons are indirect: the cited studies differ from the present work in sample size, sequence length, seed/key generation methodology, and evaluation protocols, so the figures below should not be read as a controlled head-to-head comparison.

1. **Raw PRESENT:** Shah and Ismail [9] reported that raw PRESENT used directly as a PRNG fails NIST SP 800-22 tests across all categories, consistent with the expectation that raw block-cipher output alone is insufficient. We cite this as qualitative context from a separate study, not as a controlled comparison under our experimental conditions.
2. **Davies-Meyer PRESENT:** Susanti *et al.* [14] reported a 35% configuration-level failure rate (17 of 48 tested samples), concentrated in the Non-Overlapping Template Matching test. By comparison, the present study observed an overall failure rate of 30.2% (29 of 96 configurations). We report this as a descriptive comparison only. A controlled replication under matched conditions would be required to support claims of statistically superior performance in a formal sense.
3. **Standardized DRBGs (HMAC-DRBG/CTR-DRBG):** Standardized DRBGs are well-established and pass NIST SP 800-22 testing, but typically rely on heavier primitives (e.g., SHA-256, AES-256) [5]. The LightMAC-PRESENT construction retains the lightweight footprint of the PRESENT cipher; however, no direct empirical comparison of statistical pass rates or implementation costs against standardized DRBGs was performed in this study, and this remains an important direction for future work.

4.4.5. Best Statistical Configuration

Based on the aggregate statistical trends and theoretical cryptographic considerations, the optimal configuration for the LightMAC-PRESENT-based PRNG utilizes a **128-bit seed message x, a 128-bit PRESENT key, and an 8-bit internal counter**.

We note for completeness that a single cell in the dataset — the 64-bit seed with a 128-bit key and $s = 8$ bits — achieved a 100% pass rate across all four key-pattern combinations in both scenarios (Fig. 5), the highest of any individual cell. We do not select this cell as our

overall recommendation, because a single-cell 100% result over only 8 configuration/scenario combinations is itself consistent with sampling variance at the $\alpha = 0.01$ significance level used throughout this study (cf. Section 4.2.3), and because it does not benefit from the additional robustness margin that the 128-bit seed length provides in aggregate (Section 4.4.3). Our recommendation below is therefore based on marginal aggregate pass rates across the full set of 96 configurations rather than on any single best-performing cell, and we flag this explicitly to make the basis for the recommendation transparent.

The parameter recommendations are established as follows:

1. **Key length:** A 128-bit PRESENT key is recommended, as it consistently yields higher pass rates (77.1% vs. 62.5% overall) and provides a larger security margin against brute-force key recovery.
2. **Counter length:** An internal counter of $s = 8$ bits is preferred, as it maximizes the message-bit capacity per LightMAC block, contributing to higher aggregate pass rates.
3. **Seed length:** A 128-bit seed message x is recommended over the 64-bit seed, as it achieved a significantly higher aggregate pass rate (75.0% vs. 64.6%) across all configurations tested, demonstrating more consistent statistical uniformity.

It is emphasized that these recommendations are derived from statistical randomness evaluations only and do not constitute cryptographic security guarantees.

5. Conclusion

This study specified and empirically evaluated the statistical randomness properties of a LightMAC-PRESENT-based pseudorandom number generator using the NIST SP 800-22 Revision 1A statistical test suite, across 96 parameter configurations comprising two seed lengths ($|x| \in \{64, 128\}$ bits), two key lengths ($k \in \{80, 128\}$ bits), and three internal counter lengths ($s \in \{8, 16, 32\}$ bits), each tested under four key-pattern combinations in two experimental scenarios.

Of the 96 evaluated configurations, 67 (69.8%) satisfied all 15 NIST tests under both the proportion and P-value uniformity criteria. The key findings are:

1. **Key length is the dominant parameter:** Configurations with a 128-bit PRESENT key achieved a 77.1% aggregate pass rate, compared to 62.5% for 80-bit keys, consistent with the expectation that longer keys improve the block cipher's mixing quality.
2. **Counter length influences diffusion:** A shorter internal counter ($s = 8$ bits) yielded the highest pass rate (81.3%), which can be attributed to the larger number of message bits available per LightMAC block for diffusion.
3. **Failures are plausibly dominated by statistical noise:** 70.3% of all observed failures occurred in the Non-Overlapping Template Matching test, which comprises 148 independent sub-tests. At a significance level of $\alpha = 0.01$, the probability of at least one false positive across 148 sub-tests for a truly random source is $1-0.99^{148} \approx 77.3\%$ (Equation 3), indicating that isolated template failures are statistically expected under H_0 and, on their own, do not constitute demonstrated evidence of structural weakness – although this study does not formally rule out a smaller, superimposed structural contribution to specific failures.
4. **No systematic entropy dependence observed:** The random–random key pattern exhibited a lower pass rate (54.2%) than all patterned-key variants (75.0% each). This pattern is consistent with, but does not by itself prove, the interpretation that the observed failures reflect statistical variance rather than sensitivity to key entropy; we present it as the most parsimonious explanation of the available data rather than a demonstrated causal conclusion.

These findings indicate that the LightMAC-PRESENT construction produces output that generally satisfies the NIST SP 800-22 statistical criteria, with performance primarily influenced

by seed length, key length, and counter size, rather than by key-pattern entropy. Empirically, to maximize the statistical randomness of the output, deploying the generator with a 128-bit seed message x, a 128-bit key, and an 8-bit internal counter is recommended, as these parameter values consistently achieved the highest pass rates in our evaluations. We note that this feedback-based construction has a bounded safe-output length, as its 64-bit internal state after the first iteration implies an expected cycle length on the order of 2^{32} blocks (Section 1, Scope and Limitations); this is not a concern for the 10^6-bit sequences evaluated here but should be considered in any application requiring substantially longer output streams from a single seed.

The present study does not establish formal cryptographic security; the results are strictly limited to statistical randomness at the specified significance level and sample sizes tested. Passing the NIST SP 800-22 test suite does not imply unpredictability, indistinguishability from random, or resistance to state-compromise or distinguishing attacks. Future work should investigate formal security reductions, resistance to distinguishing attacks, controlled comparisons with standardized DRBGs under matched experimental conditions, and hardware performance characteristics in resource-constrained environments.

CRediT Authorship Contribution Statement

Bety Hayat Susanti: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing, and Funding acquisition. **Wahyu Achmad Fadiel**: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, and Writing – review & editing. **Yeni Farida**: Data curation, Validation, Writing – review & editing, and Administration. **Obrina Briliyant**: Writing – review & editing.

Declaration of Generative AI and AI-assisted technologies

During the preparation of this manuscript, the authors used ChatGPT version 4 and Grammarly to assist with language refinement, grammar checking, formatting adjustments, and clarity improvements. The authors reviewed and verified all content generated or suggested by these tools to ensure accuracy, validity, and alignment with the intended meaning. The final responsibility for the content of this manuscript rests entirely with the authors.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding and Acknowledgments

This research was funded by the Politeknik Siber dan Sandi Negara, Bogor, West Java, Indonesia. We extend our sincere appreciation to the reviewers for their valuable feedback and suggestions.

Ethical Considerations

This study did not involve human participants, animal subjects, or personally identifiable data. All experiments were conducted on synthetically generated bit sequences produced by the proposed generator, and no ethical approval was therefore required.

References

[1] Charmaine Kenny and Krzysztof Mosurski. "Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators". In: *Management Science and Information Systems Studies* (2005). https://www.random.org/analysis/Analysis2005.pdf.

[2] William Stallings. *Cryptography and Network Security: Principles and Practice*. 8th, Global Edition. London, United Kingdom: Pearson Education Limited, 2023. https://www.pearson.com/en-gb/subject-catalog/p/cryptography-and-network-security-principles-and-practice-global-edition/P200000007245.

[3] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. "Message Authentication using Hash Functions— The HMAC Construction". In: *RSA Laboratories' CryptoBytes* 2.1 (1996). https://cseweb.ucsd.edu/~mihir/papers/hmac-cb.pdf.

[4] Bart Preneel. "HMAC". In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. 2nd. New York, NY: Springer, 2011, pp. 554–555. https://link.springer.com/rwe/10.1007/978-1-4419-5906-5_581.

[5] Elaine Barker and John Kelsey. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. Tech. rep. NIST Special Publication 800-90A Revision 1. National Institute of Standards and Technology, 2015. DOI: 10.6028/NIST.SP.800-90Ar1.

[6] Atul Luykx, Bart Preneel, Elmar Tischhauser, and Kan Yasuda. "A MAC Mode for Lightweight Block Ciphers". In: *Fast Software Encryption*. Ed. by Thomas Peyrin. Berlin, Heidelberg: Springer, 2016, pp. 43–59. DOI: 10.1007/978-3-662-52993-5_3.

[7] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Christian Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher". In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Springer, 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31.

[8] ISO/IEC. *Information Technology – Security Techniques – Lightweight Cryptography – Part 2: Block Ciphers*. Standard ISO/IEC 29192-2:2012. Geneva, Switzerland: International Organization for Standardization, 2012. https://www.iso.org/standard/56552.html.

[9] Isma Norshahila Binti Mohammad Shah and Eddie Shahril Bin Ismail. "Randomness Analysis on Lightweight Block Cipher, PRESENT". In: *Journal of Computer Science* 16.11 (Nov. 2020), pp. 1639–1647. DOI: 10.3844/jcssp.2020.1639.1647.

[10] Andrew Rukhin et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Tech. rep. 800-22 Rev. 1a. National Institute of Standards and Technology, 2010. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf.

[11] Amalia Beatriz Orúe López, Luis Hernández Encinas, Agustín Martín Muñoz, and Fausto Montoya Vitini. "A Lightweight Pseudorandom Number Generator for Securing the Internet of Things". In: *IEEE Access* 5 (2017), pp. 27800–27806. DOI: 10.1109/ACCESS.2017.2774105.

[12] Luthfi Hakim Panuntun and Bety Hayat Susanti. "Evaluation of randomness hash-DRBG-based Ascon hash function using NIST SP800-22". In: *AIP Conference Proceedings*. Vol. 3320. 1. AIP Publishing, July 2025, p. 030003. DOI: 10.1063/5.0286255.

[13] Bety Hayat Susanti, Jimmy Jimmy, and Mareta W. Ardyani. "ENT Randomness Test on DM PRESENT-80 and DM-PRESENT-128-based Pseudorandom Number Generator". In: *4th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*. 2021, pp. 33–38. DOI: 10.1109/ISRITI54043.2021.9702862.

[14] Bety Hayat Susanti, Jimmy Jimmy, and Mareta W. Ardyani. "Evaluation with NIST Statistical Test on Pseudorandom Number Generators based on DMP-80 and DMP-128". In: *2022 5th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*. 2022, pp. 166–171. DOI: 10.1109/ISRITI56927.2022.10053041.

[15] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. Boca Raton: CRC Press, 2018. DOI: 10.1201/9780429466335.

[16] Elena Almaraz Luengo. "Coverage study of the NIST SP 800-22 randomness statistical tests suite". In: *Computational and Applied Mathematics* 45 (2026). DOI: 10.1007/s40314-026-03663-y.

[17] Gokay Saldamli, Levent Ertaul, and Asharani Shankaralingappa. "Analysis of Lightweight Message Authentication Codes for IoT Environments". In: *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*. June 2019, pp. 235–240. DOI: 10.1109/FMEC.2019.8795359.

[18] Muhammed Saadetdin Kaya and Kenan İnce. "Benchmarking Various 1D Chaotic Maps For Lightweight Pseudo-Random Number Generation". In: *2024 8th International Artificial Intelligence and Data Processing Symposium (IDAP)*. Sept. 2024, pp. 1–5. DOI: 10.1109/IDAP64064.2024.10710841.

[19] Sivaraman R, Varada Sai Madhukar, Sriranjana V, Kavyadharshni S, Naresh Kumar H, and Siva J. "Wireless Reseeding Scheme for Lightweight PRNG Architecture on FPGA -Design and Analysis". In: *2024 Control Instrumentation System Conference (CISCON)*. Aug. 2024, pp. 1–6. DOI: 10.1109/CISCON62171.2024.10696289.

[20] Serge Sandakly, Charbel Salem, Khalil Challita, Shuvalaxmi Dass, Joseph Azar, Jacques Bou Abdo, and Jacques Demerjian. "XorshiftH128+: A hybrid random number generator for lightweight IoT". In: *2023 IEEE International Conference on Artificial Intelligence, Blockchain, and Internet of Things (AIBThings)*. Sept. 2023, pp. 1–5. DOI: 10.1109/AIBThings58340.2023.10292458.

[21] UMUT ÇABUK, ÖMER AYDIN, and GÖKHAN DALKILIÇ. "A random number generator for lightweight authentication protocols: xorshiftR". In: *Turkish Journal of Electrical Engineering and Computer Sciences* 25.6 (Jan. 2017), pp. 4818–4828. DOI: 10.3906/elk-1703-361.

[22] Bety Hayat Susanti, Wahyu Achmad Fadiel, Yeni Farida, and Obrina Briliyant. *Source Code and Dataset for: Randomness Test of LightMAC-Based Pseudorandom Number Generator (PRNG) with NIST SP 800-22 Revision 1A Test*. Zenodo, Aug. 2026. DOI: 10.5281/zenodo.22023993.

[23] Erich L Lehmann and Joseph P Romano. *Testing Statistical Hypotheses*. 3rd. Springer Science & Business Media, 2006. DOI: 10.1007/0-387-27605-X.