Comparative Analysis Between Counter Mode Deterministic Random Bit Generators and Chaos-Based Pseudo-Random Number Generators

Raluca Ionela Caran
Faculty of Information Systems and Cyber Security
Military Technical Academy "Ferdinand I"
Bucharest, Romania
raluca.caran@mta.ro

Abstract— **Pseudo-random number generators (PRNGs) are essential components in cryptographic applications, providing the basis for generating keys, creating digital signatures, and ensuring secure communications. This research explores two methodologies for pseudo-random number generation: the implementation of the Counter mode deterministic random bit generator (CTR_DRBG) according to the National Institute of Standards and Technology (NIST) specifications, and a chaos-based pseudo-random number generator. The CTR_DRBG implementation utilizes a 256-bit seed and follows strict NIST guidelines, ensuring resistance against brute force and cryptanalytic attacks. In contrast, the chaos-based approach harnesses chaotic dynamics to generate high-quality random values efficiently based on a 256-bit key. By optimizing parameters and introducing a threshold for random bit generation, we prove that the chaos-based generator achieves superior randomness and statistical properties.**

Keywords: pseudo-random number generators, CTR_DRBG, chaos-based generator, security, chaotic map

I. Introduction

A wide range of domain applications are based on sequences of random numbers, for example, simulations (in fields like physics [1], [2] or engineering [3]), statistics [4]-[6], machine learning [7], gaming [8], and especially cryptography [9]. Random numbers can be achieved in two ways [6]: (1) using a true random number generator (TRNG) or (2) using a pseudorandom number generator (PRNG). PRNGs play an important role in various cryptographic applications, providing a fundamental building block for generating keys, creating digital signatures, and ensuring secure communications [10]. The reliability and unpredictability of these generators are essential for maintaining the security of sensitive data and systems in modern digital environments, even more so in today's technological context, where large volumes of data are generated from different sources. According to [11] there are several types of PRNGs: linear congruential generators (LCG), linear feedback shift registers (LFSR), and PRNGs that exploit the intricate chaotic behavior in dynamic systems that includes principles from chaos-based cryptography. Chaos-based cryptography is a branch of cryptography that includes the principles of chaotic systems to secure communications [12]. Chaotic systems are highly sensitive to initial conditions, proving deterministic yet unpredictable behavior, and are non-linear dynamical systems. These characteristics make chaos a good foundation for developing cryptographic algorithms. Chaos-based cryptography has applications in different fields [12]: secure communications, pseudorandom number generation, image and video encryption, hash functions, secure key exchange, etc.

In this paper we implemented the CTR_DRBG following NIST's guidelines [13] and a PRNG based on the logistic map, one of the most used technique in chaos-based cryptography. The second approach is inspired from study [14] that involves an enhanced logistic map. Compared to [14], we refined parameters and changed the way in which the digits are generated. We prove that both implementations pass the NIST test suite for random and pseudorandom number generators [15], but the chaos-based PRNG demonstrates superior performance in terms of execution time. The rest of the paper is organized as follows: Section II presents the methods and tools used in this study, Section III presents the details of implementation for the two approaches, Section IV discuss the results, and Section V presents the conclusions of the study and further research.

II. Methods and Tools

A. *Counter mode Deterministic Random Bit Generator*

The first tool implemented in this research is based on the CTR_DRBG, which is a specific type of Cryptographically Secure PRNG (CSPRNG) defined in NIST SP 800-90A, Rev. 1 [13]. A CTR_DRBG needs an entropy input, has an internal state that stores the parameters, variables and additional values, and consists of the following five functions: [13]

- *Instantiate Function*: Initializes the CTR_DRBG with an entropy-based seed, possibly enriched with a nonce and a personalization string, to establish its initial internal state.
- *Generate Function*: Produces pseudorandom bits using the current state, then updates the state in preparation for future requests.
- *Reseed Function*: Refreshes the generator by integrating new entropy with the existing state and any additional input, resulting in a rejuvenated internal state.
- *Uninstantiate Function*: Securely deletes the internal state to prevent any subsequent recovery or misuse of the state.
- *Health Test Function*: Verifies the operational integrity of the DRBG to ensure it continues to function as expected.

B. *Chaos Pseudo-Random Bit Generator (Chaos PRBG)*

The second tool implemented in this research explores chaos-based pseudo-random number generation techniques. In chaos-based cryptography, one of the most important map is the logistic map, which has the following definition [16]:

$$F_L(x_{n+1}, r) = F_L(r \cdot x_n \cdot (1 - x_n)), \qquad (1)$$

where the parameter $r \in (0,4)$, the initial value $x_0 \in (0,1)$, $x_n \in (0,1)$, and $n \in \{1,2,\ldots,N\}$, N being the number of iterations. In our implementation, we used an enhanced logistic map proposed in [14]:

$$x_{n+1} = \frac{2^{10}}{2^{F_L(x_n, r)}} \tag{2}$$

where F_L is the logistic map defined in Eq. (1). The author introduces an enhanced version of the logistic map in equation (2) to achieve increased chaotic complexity and a broader chaotic range. This enhancement involves incorporating the chaotic sequences generated by the logistic map into a generic equation with heightened sensitivity, facilitated by the utilization of a multiplicative inverse function. By leveraging this approach, the proposed chaotification model exhibits generality and maintains a range within $x \in [0,1]$. Notably, the design choice to employ a fractional function yields improved statistical properties, enhancing the overall efficacy of the method. We chose 2^{10} because the system requires a reasonable computation time, unlike higher powers, which would necessitate more time.

III. The Proposed Method

A. Counter Mode Deterministic Random Bit Generator Configuration

For the CTR_DRBG implementation, Advanced Encryption Standard (AES)-256 [17] was employed with the following parameters for instantiation, reseeding, and generation functions:

- Highest Supported Security Strength: 256 bits, ensuring a minimum entropy level [19]
- Block Length of AES: 128 bits
- Key Length: 256 bits (Increases the complexity of the algorithm. Currently, there is no known efficient method to break AES-256 through brute force or other cryptographic means)
- Entropy Input Length: Ranging from 256 to 1000 bits [13]
- Maximum additional input length: 800 bits [13]
- Seed Length: 384 bits (seed length=output length + key lenght)
- Maximum number of bits per request: 4000 bits [13]
- Reseed Interval**:** 100000 requests [13]

The seed material for the CTR_DRBG was obtained using an entropy pool provided by the Windows operating system. An Application Programming Interface (API) **Error! Reference source not found.** facilitated the collection of entropy from diverse system data sources, including mouse or keyboard timing input, various system data and user data, such as the process ID, thread ID, system clock, system time, system counter, memory status, free disk clusters, and hashed user environment block.

Seed Derivation and Generation Function. The initial step involved instantiating the CTR_DRBG to obtain the seed material, which was further processed through a derivation function to derive the seed. Subsequently, a secure generation function was developed to produce pseudo-random bits based on the derived seed.

Periodic Reseeding for Enhanced Security. Periodic reseeding was implemented to address potential threats to the security of the DRBG seed, entropy input, or working state over time. By periodically refreshing the seed material, the security risks were mitigated, reducing the likelihood of compromising the data protected by cryptographic mechanisms employing the DRBG.

Prediction and Backtracking Resistance. The CTR_DRBG implementation prioritized prediction resistance and backtracking resistance, ensuring robust protection against potential attacks attempting to predict future random outputs or backtrack previously generated random values.

1) Randomness Testing Applied on CTR_DRBG

To assess the statistical randomness of the generated output, we utilized the NIST SP 800–22 test suite [15]. Each sub-test within the NIST suite was applied to evaluate the randomness properties. The significance level was set to $\alpha = 0.01$, and a test sample was considered to have passed each sub-test if the P-value exceeded α. A total of 20 test samples, each with a length of 100000 bits, were used for testing. The results are summarized in Table I. The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 18 for a sample size = 20 binary sequences. The minimum pass rate for the random excursion (variant) test is approximately = 2 for a sample size = 3 binary sequences.

B. Chaos PRBG Configuration

For the implementation of this algorithm, we started with an enhanced logistic map from Eq. (2), utilizing Lyapunov Exponent and Fuzzy Entropy as metrics for assessing sensitivity and complexity, respectively. The Lyapunov Exponent provides insights into the system's sensitivity to initial conditions, a fundamental aspect of chaotic behavior. By quantifying how nearby trajectories diverge over time, it helps gauge the unpredictability and chaotic nature of the system. In chaotic systems, even tiny differences in initial conditions can lead to vastly different outcomes over time. The exponent is positive so it indicates chaotic behavior, meaning the system is highly sensitive to initial conditions and unpredictable in the long term. On the other hand, Fuzzy Entropy offers a measure of complexity, capturing the irregularity and intricate patterns within the chaotic sequences.

1) Chaos-based PRBG Pseudo-Code:

For the Chaos-based PRBG we started with the algorithm proposed in [14] for which we adjusted the parameters to optimize the performance of the chaotic variable processing and changed the way in which the digits are obtained.

Input: secret key *key*, temporar key length *temp_key_len*, parameter *r*, initial value of x *x(0)*, threshold value *p*

Output: pseudo-random bits *result*

```
1.  temp_key_len=52;
2.  key_len=256;
3.  counter= key_len -temp_key_len+1;
4.  r=3.99999
5.  x(0)=0.66;
6.  p=5;
7.  for i=0 to counter do:
8.      key1=key.Substring(i, len_temp_key)
9.      for j=0 to len_temp_key do:
10.         if key1(j)=='1' then
11.             temp_key(i)=temp_key(i)+1/2^j;
12. for i=1 to counter do:
13.     x(i)=(x(i)+temp_key(i-1)) mod 0.99999;
```

```
x(i)=$2^{10}/2^{r*x(i)*(1-x(i))}$ mod 1;
digits=x(i).digits;
for each digit in digits do
    if digit < p then
        result=result+'0';
    else
        result=result+'1';
```

The algorithm outlines the procedural steps involved in generating pseudo-random bits, crucial for various applications such as cryptography and simulation. It relies on the integration of a secret key to perturb the chaotic variables, enhancing randomness and security. The chaotic variables produced from the secret key exhibit a high level of sensitivity to even minor alterations in the key, which can be attributed to the chaotic perturbation operations applied during the process. The entropy source of this algorithm is represented by this key, considered as seed material, which must be kept secret. The key generation process incorporates an Analog Ambient Light Sensor V2.1 strategically positioned in densely populated areas to capture dynamic environmental changes. The sensor data undergoes transformation via an Analog Digital Converter, yielding a two-byte digital representation. It has been observed that the last byte of the resulting value oscillate frequently, so the binary form of this byte is extracted for later use in the key bit component. Subsequently, the key undergoes segmentation into 52-bit blocks to maximize bit generation potential. This choice aligns with the representation of the fractional part of a double, which consists of 52 bits. The initial value of the parameter *r* is selected to leverage the high positive value of the Lyapunov exponent, ensuring precision in chaotic behavior. As per previous studies [14], the enhanced logistic map demonstrates chaotic behavior within the range of r values spanning 3.67 to 4. Each digit of the decimal part of each value of the obtained chaotic variable is compared with a threshold value, resulting in a bit. Using a 256-bit key, a variable number of pseudo-random bits can be generated, ranging between 3075 bits and 3485 bits, intensifying security due to its length. Evaluation of the execution times of both algorithms for pseudo-random number generation proves the superiority of the second algorithm.

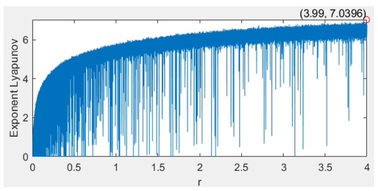

Figure 1 Lyapunov Exponent

To investigate the behavior of the Lyapunov exponent as a function of the system's control parameter, *r*, we employed a numerical method to generate the plot. First, we defined a set of values for *r* between 0 and 4, generating 4 million points to cover this interval and set initial value x=0.66. This fine resolution of *r* values allows us to observe the detailed behavior of the Lyapunov exponent and to identify the point at which it reaches its maximum. Fig. 1 demonstrates that for r=3.99 the Lyapunov Exponent has a value of approximately 7.03. We chose r=3.99999 for improved precision and in accordance with [14]. We utilized the modulus 0.99999 in the calculation of x(i) to ensure that remains in the appropriate phase space , as well as to align with the approach outlined in [14]. The initial value of x=0.66 was chosen randomly.

2) Randomness testing applied on Chaos_PRBG

We conducted the suite of 15 tests from NIST [15], configuring the same parameters as those used in testing the CTR_DRBG algorithm. The results are documented in Table I.

IV. Discussions

A. Results

In this section, we compare the results that we obtained by implementing CTR_DRBG and the integration of chaos-based random number generation. The test results are captured in Table I. Both implementations pass all NIST tests, indicating that the outputs of the deterministic random bit generators demonstrate adequate randomness, uniform distribution, and complexity.

TABLE I. NIST SP 800-22 RESULTS

Statictical Test	**Proportion CTR_DRBG**	**Proportion Chaos PRBG**
Frequency	19/20	19/20
Block frequency	20/20	20/20
Cumulative Sums*	20/20	19/20
Runs	20/20	19/20
Longest run	20/20	19/20
Binary matrix rank	20/20	20/20
FFT	20/20	20/20
Non-overlapping template.*	19-20/20	19-20/20
Overlapping template.*	19/20	20/20
Universal	20/20	20/20
Approximate entropy	20/20	20/20
Random excursions.*	3/3	1/1
Random excursions variant.*	3/3	1/1
Serial*	20/20	19-20/20
Linear Complexity	18/20	20/20
Success Counts	**15/15**	**15/15**

The simplified code required for chaos-based generation facilitates ease of implementation and reduces computational overhead compared to the NIST DRBG schema. The introduction of a threshold for random bit generation ensures controlled randomness, enhancing the security and predictability of the generated bits. We tried to utilize the values of x without this threshold, but the test results were unsatisfactory. Only the Rank, Linear Complexity, and a few Non-overlapping tests passed. I employed both the complete value and the final 2 and 3 digits of the binary representation acquired from the Analog-to-Digital Converter as the 256-bit key. However, the last byte proved to be a better source of entropy. When selecting a smaller precision for *r*, such as *r*=3.99, the Cumulative Sums, Frequency and Runs tests failed.

B. *Comparison with Related Work*

Our research draws inspiration from study [14] that constructed a chaos-based block cipher. While the original focus was on image encryption, we adapt the concept of chaotic variable generation, modifying parameters and precision to achieve superior chaotic behavior in generating pseudo-random numbers. By incorporating insights from chaos theory and optimizing parameter selection, we prove the quality and unpredictability of the generated random values. Our approach yields better results compared with related work [14] in certain randomness tests from the NIST test suite, as evidenced by improved p-values shown in Table III, indicating enhanced randomness and statistical properties.

TABLE II.

Statictical Test	**p-value Chaos-based PRBG**	**p-value chaos-based block cipher**
Block frequency	0.5341	0.3321
Runs	0.6371	0.2152
Approximate entropy	0.7399	0.1782
Serial*	0.2133	0.1265

The secret keys employed in these algorithms have a length of 256 bits, which offers enhanced security compared to CTR_DRBG_128, and the chaos-based implementation [14], which utilizes 128-bit keys. A 256-bit key provides high security in cryptography making it highly difficult to break through brute force or other cryptanalysis methods in a reasonable amount of time. For instance, the total number of possible keys with 256 bits is approximately 2^{256}, which is an extremely large value and exceeds the computing capacity of current computers to try all these keys in a reasonable time.

V. Conclusions

In this research, we have explored and developed cryptographic algorithms aimed at enhancing the security and efficiency of pseudo-random number generation. By focusing on two distinct methodologies, namely the Counter Mode Deterministic Random Bit Generator and a chaos-based pseudo-random number generator, we aimed to contribute to the advancement of cryptographic techniques.

Looking towards future directions, our emphasis will be on the exploration of various parameters within the chaos-based pseudo-random number generation algorithm. This exploration aims to achieve favorable results on the suite of NIST tests for bit streams of larger lengths. Exploring performance metrics beyond just execution time, such as scalability and memory usage, could enrich future research comparisons. Additionally, we intend to implement enhancements to the current implementation, striving for improved performance and resilience. Furthermore, expanding the comparison to include more varieties of PRNGs such as Mersenne Twister, Permuted Congruential Generator or another algorithm proposed by NIST could offer a more comprehensive overview of the landscape.

References

[1] A. Acín and L. Masanes, ‘Certified randomness in quantum physics’, *Nature*, vol. 540, no. 7632, pp. 213–219, 2016.

[2] M. N. Bera, A. Acín, M. Kuś, M. W. Mitchell, and M. Lewenstein, ‘Randomness in quantum mechanics: philosophy, physics and technology’, *Reports on Progress in Physics*, vol. 80, no. 12, p. 124001, 2017.

[3] L. Serhii, O. Oleksandra, S. Nataliya, and Z. Andrii, ‘Modeling and signals processing using cyclic random functions’, in 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), IEEE, 2018, pp. 360–363.[Online].Available:https://ieeexplore.ieee.org/abstract/document/8526653/

[4] R. M. Bethea and R. R. Rhinehart, Applied engineering statistics. Routledge, 2019.

[5] G. Grimmett and D. Stirzaker, Probability and random processes. Oxford university press, 2020.

[6] S. M. Ross, Simulation. Academic Press, 2022.

[7] S. A. Friedler, C. Scheidegger, S. Venkatasubramanian, S. Choudhary, E. P. Hamilton, and D. Roth, ‘A comparative study of fairness-enhancing interventions in machine learning’, in Proceedings of the Conference on Fairness, Accountability, and Transparency, Atlanta GA USA: ACM, Jan. 2019, pp. 329–338. doi: 10.1145/3287560.3287589.

[8] C. Zhou, ‘Analysis of Logistic Map for Pseudorandom Number Generation in Game Development’. arXiv, Feb. 29, 2024. [Online]. Available: http://arxiv.org/abs/2403.00864

[9] O. Petura, ‘True random number generators for cryptography: Design, securing and evaluation’, PhD Thesis, Lyon, 2019. [Online]. Available: https://www.theses.fr/2019LYSES053

[10] A. Röck, Pseudorandom number generators for cryptographic applications. 2005. [Online]. Available: http://www-rocq.inria.fr/secret/Andrea.Roeck/pdfs/dipl.pdf

[11] K. Bhattacharjee and S. Das, ‘A search for good pseudo-random number generators: Survey and empirical studies’, Computer Science Review, vol. 45, p. 100471, 2022.

[12] L. Kocarev and S. Lian, Chaos-based cryptography: Theory, algorithms and applications, vol. 354. Springer Science & Business Media, 2011.

[13] E. B. Barker and J. M. Kelsey, Recommendation for random number generation using deterministic random bit generators (revised). US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory. 2007. [Online]. Available: http://www.gocs.eu/pages/fachberichte/archiv/088-Draft_SP800-90A-Rev1_May-2011.pdf

[14] M. Alawida, J. S. Teh, A. Mehmood, A. Shoufan, and W. H. Alshoura, “A chaos-based block cipher based on an enhanced logistic map and simultaneous confusion-diffusion operations,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, Part A, pp. 8136–8151, Nov. 2022, doi: 10.1016/j.jksuci.2022.07.025.

[15] A. Rukhin et al., A statistical test suite for random and pseudorandom number generators for cryptographic applications, vol. 22. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory. 2001.

[16] T. Tsuchiya and D. Yamagishi, “The Complete Bifurcation Diagram for the Logistic Map,” Zeitschrift für Naturforschung A, vol. 52, Jun. 2014, doi: 10.1515/zna-1997-6-708

[17] D. Selent, ‘Advanced encryption standard’, Rivier Academic Journal, vol. 6, no. 2, pp. 1–14, 2010.

[18] ‘CryptGenRandom function (wincrypt.h) - Win32 apps’.[Online]. Available:https://learn.microsoft.com/enus/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom

[19] E. Barker, “Recommendation for Key Management: Part 1 – General,” National Institute of Standards and Technology, NIST Special Publication (SP) 800-57 Part 1 Rev. 5, May 2020. doi: 10.6028/NIST.SP.800-57pt1r5.