PRNG Implementation Based on Chaotic Neural Network (CNN)

Mohammed Jassim Mohammed
Department of Computer & Communication Engineering, Nawroz University, Duhok, Kurdistan Region – Iraq

ABSTRACT

In this work, a neural network with chaos activation function has been applied as a pseudo-random number generator (PRNG). Chaotic neural network (CNN) is used because of its noise like behaviour which is important for cryptanalyst to know about the hidden information as it is hard to predict. A suitable adaptive architecture was adopted to generate a binary number and the result was tested for randomness using National Institute of Standard Technology (NIST) randomness tests.

Although the applications of CNN in cryptography have less effective than traditional implementations, this is because these systems need large numbers of digital logic or even a computer system. This work will focus on applications that can use the proposed system in an efficient way that minimize the system complexity.

1. Introduction

Random number generation algorithms are very important in many practical applications of the cryptographic. Although, all of these algorithms are deterministic and produce sequences of numbers that are not statistically random, but the algorithm produces sequences pass many reasonable tests of randomness, such numbers are referred to as pseudorandom numbers [1].

One of the most important applications used the PRN is the stream cipher, in which ciphertext output is produced bit-by-bit or byte-by-byte from a stream of plaintext input, where the PRNG used instead of True Random Number Generator (TRNG) because:

- The sender need only to deliver the key (or the seed), which is typically 54 or 128 bit, to receiver in the secure fashion.
- It able to generate much faster than the true random number generator.

The necessarily compatible requirements for a sequence of random number [1] and [2].

1. Next random bit must be forward and backward unpredictable, where in both cases we cannot determine the next or previous bits from knowledge for any generated values.

   1. Random bit stream appear random even though it is deterministic and must pass the statistical tests of randomness (e.g. NIST 800-22 test suite [3])
   2. The same random bit stream must not be able to be reproduced

Chaotic systems can provide those requirements, where the main characteristics of chaotic systems are [4] and [5]:

- Dynamical systems that highly sensitive to initial condition, i.e. a small differences in initial conditions cause unpredicted output.
- Noise-like behavior, a small differences in initial conditions cause unpredicted output
- Unstable periodic orbits with long periods

Due to these features, chaotic systems are extensively incorporated into encryption systems as a random generator [2] and [6-9], or block cipher application [10].

Academic Journal of Nawroz University
(AJNU) Volume 8, No 4 (2019).
Regular research paper : Published 4 Nov 2019
Corresponding author's e-mail : mjm562000@yahoo.com

doi : 10.25007/ajnu.v8n4a459

On the other hand, Artificial Neural Network (ANN) represents highly nonlinear systems able to handle noisy data and fault tolerance and difficult decrypting by brute-force attack [11], make it more suitable choice in cryptosystem. So we can find several application of neural network in cryptosystem like:

- PRNG [12]
- Image and data encryption [13]
- Public key generation [14]
- Block cipher [15]

And many other applications can be reviewed in [6] and [17].

The goal of this work is to implement the proposed PRNG based Chaotic Neural Network using Matlab and test the performance of the proposed generator using NIST 800-22 test suite.

The rest of the paper is organized as follows: related works in section 2, basics of ANN and its learning in section 3, section 4 describes the behaviour of some chaotic equations implementation, PRNG structure using chaotic neural network given in section 5, at last the system implementation and conclusions given in sections 6 and 7 respectively.

2. Related Work

The major weakness of the most present random number generators is linearity. In other words, if we obtained portion of a random sequence, the successive numbers may be calculated using the associated linear function [17]. We can find different applications of the neural network in cryptography in [18]; this review gives some examples of highly nonlinear PRNGs and some applications of different neural networks architecture in cryptography.

Singla et al. [5] merged the features and strengths of chaos and neural network are combined to design a pseudo-random binary sequence generator. The statistical performance was examined against the NIST SP800-22 randomness tests. The results of investigations are promising and depict its relevance for cryptographic applications.

The structure of artificial neural network was used as a key as a solution of synchronization in cryptography [11]. The proposed method was employed for text, audio and image data. The results were compared with k nearest neighbor and wavelet transforms and showed that his algorithm faster than the others with 100% decryption accuracy.

Yayik and Kutlu [12], proposed a neural network-based pseudo-random numbers. The performance of this generator was tested for randomness using National Institute of Standard Technology (NIST) randomness tests. After they built two identical ANNs, one for non-linear encryption was modeled using relation building functionality. The encrypted data was decrypted with the second neural network using decision-making functionality.

A recurrent neural network was used to design a symmetric cipher able to resisting different attacks [19]. The weight distribution of the hidden layers was totally depends on the original key. The proposed system supports variable key and block length.

In this work a PRNG using the CNN, as described in [5], was implemented using Matlab, at same time several programs was built to test the generator performance based on the NIST SP800-22 [3].

3. PRNG Based on Chaotic Neural Network

In this section the proposed PRNG architecture will discuss. Figure 8 shows the general structure of the proposed system. The network consists of 4 layers: input layer, the first hidden layer, the second hidden layer and the output layer. The function of each layer (or so called forward computation) given by:

Input Layer:

The input for this network is 64 bits represent the seed

doi : 10.25007/ajnu.v8n4a459

(P = 64 bit) of the PRNG, and the output of this layer given by:

$net_0 = W_0 P + B_0$ … (1)

$Y_0(0) = f(net_0, Q_0)$ … (2)

$Y_0(k+1) = F(Y_0(k),\ Q_0) \quad k = 1\colon n_0$ … (3)

Hidden Layer 1:

$net_1 = W_1 Y_0 + B_1$ … (4)

$Y_1(0) = f(net_1, Q_1)$ … (5)

$Y_1(k+1) = F(Y_1(k),\ Q_1) \quad k = 1\colon n_1$ … (6)

Hidden Layer 2:

$net_2 = W_2 Y_1 + B_2$ … (7)

$Y_2(0) = f(net_2, Q_2)$ … (17)

$Y_2(k+1) = F(Y_2(k),\ Q_2) \quad k = 1\colon n_2$ … (8)

Output Layer:

$net_3 = W_3 Y_2 + B_3$ … (9)

$Op(0) = f(net_3, Q_3)$ … (10)

$Op(k+1) = F(Op(k),\ Q_3) \quad k = 1\colon n_3$ … (11)

Normalize the output

$$Op = (Op \times 10^{10}) mod(256) = \begin{cases} 0 & if \quad < 127 \\ 1 & if \quad \geq 127 \end{cases}$$

… (12)

Where:

W₀(8×8), W₁(4×8), W₂(2×4), W₃(1×2) :Weight matrices< 1

B₀(8×1), B₁(4×1), B₂(2×1), B₃(1×1): Bias vectors < 1

Q₀(8×1), Q₁(4×1), Q₂(2×1), Q₃(1×1): Control parameters 0.4 < q < 0.6

0 < n₀, n₁, n₂, n₃ <=10: number of iteration 1 <= n <= 10

All these values are initialized using 64 bit key as describe in the next section.

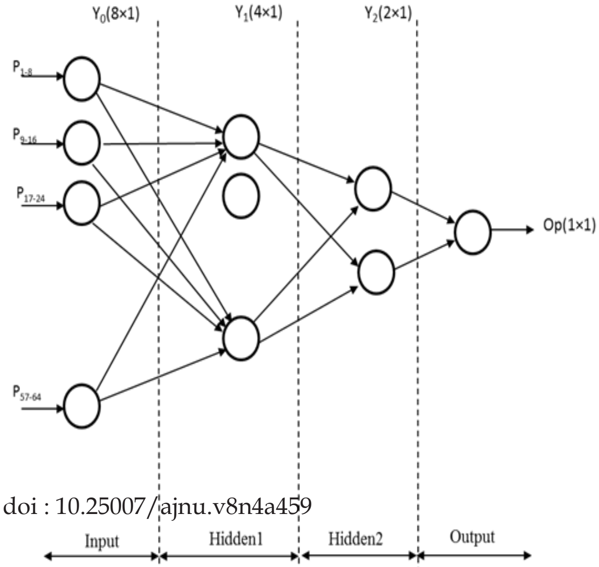

Figure 1. Neural Network Architecture

3.1 Key Generator and Initial Values

A 64 bit key with a 1-D chaotic cubic map used to generate the initial values of the CNN. As described in the following algorithm [5]:

i. *K = K₁ K₂ K₃ K₄*

Where *Ki* is a 16- bit component of the key *K (64 bit)*

ii. *Calculate the initial condition:*

$$x(1) = \sum \frac{K_i}{2^{16}} mod(1) \qquad \text{… (13)}$$

iii. *Calculate the state of the cubic map:*

$$x(k+1) = \lambda . x(k) . (1 - x(k)^2) \qquad \text{… (14)}$$

Where

λ : control parameter ($\lambda = 2.59$)

x(k) : the state (*0 <= x(k) <=1*)

3.2 Backward Adaptation

The only adapted values are the control parameter matrices $\boldsymbol{Q_i}$ by [5]:

$$Q_0 = 0.2 \times Y_1 + 0.4$$
$$Q_1 = 0.2 \times Y_2 + 0.4$$
$$Q_2 = 0.2 \times Y_3 + 0.4$$
$$Q_3 = 0.2 \times Op + 0.4$$

4. PRNG based on CNN Implementation

The proposed generator was implemented and evaluated using Matlab programming, where the general steps given by:

i. Input *K* and calculate x form (Eqs 13 and 14):

$$x(k+1) = 2.59 . x(k) . (1 - x(k)^2)$$

Where:

$$x(1) = \sum \frac{K_i}{2^{16}} mod(1)$$

- Initialize matrices based on value of x

  Weight matrices: W0(8*8), W1(4*8), W2(2*4), W3(1*2)

  Bias vectors: B0(8*1), B1(4*1), B2(2*1), B3(1*1)

  Control parameters: Q0(8*1), Q1(4*1), Q2(2*1), Q3(1*1)

  Layer iteration: n0, n1, n2, n3

Where:

$0 < Wi, Bi, Qi < 1$ and $1 <= ni <= 10$

- Input Seed (P = 64 bit)
- Operate the neural network to calculate the Op (Forward Computation)
- Update the values of (Q0, Q1, Q3, and Q4)
- Repeat steps 4, 5 and 6 to obtain the PRN sequence of desired length.

5. Performance Evaluation

In this section, the performance of the system was measured which include: the 0/1 balance test and the NIST Randomness tests.

5.1 0/1 Balance Test

The function named (BalanceTest.m) based on Matlab used to count number of ones and compute the average as shown in table 1. The equality distribution measures are found close to 50% shown in that the proposed generator satisfy the equality distribution property.

Table 1: Equality distribution of the PRNG

Sequence Length	Count of 1s	%age
1000	510	51
10000	5175	51.75
20000	10226	51.13
50000	25420	50.84
100000	50525	50.52
200000	101153	50.58
500000	252176	50.44

5.2 NIST Randomness Test

Many of the statistical test suite proposed by NIST [19] implemented using Matlab programming language (NISTtest.m). The randomness results of the proposed generator for first 1000 and 10000 bits are listed in table 2. According to Singla et. al. [5], this generator passes all the NIST tests for 100,000 samples. But for my simulation results the sequence of generated bits didn't pass all these tests for 1000 and even 10000 bits, it failed in at least one p-value. But most of my tests output the p-values obtained were greater than 0.01, which ensures the high randomness of the generated sequence.

Table 2: Some of NIST randomness tests

Randomness Test	p-values(1000)	p-values(10000)
Frequency Test	0.5271	0.2327
Block Frequency Test	0.3857	0.6882
Run Test	0.4786	0.2135
Longest Run of Ones in a Block	0.7532	0.0210
Discrete Fourier Transform	0.5617	0.1989

6. Conclusions

After implementation of the PRNG base on CNN using Matlab and perform several statistical tests on the generated binary sequence, I can summarize the main pros of this algorithm into:

- This generator uses the high sensitivity and randomness property of chaotic functions (Piece-wise linear chaotic map).
- The four-layer Neural Network increase the nonlinear complexity of the generator.
- The key space proposed in this simulator is (128 bit) where:
  - The 64 bit **Key** used to initialize the network components.
  - The 64 bit **Seed** used as an input to the network.
- According to Singla et. al. [19] and my implementation of some of the NIST randomness tests, this generator passes most of the NIST tests.
- The generated sequence pass equality distribution (equal numbers of 0's and 1's) i.e. Uniform distributed.

doi : 10.25007/ajnu.v8n4a459

- It satisfy the two necessary compatible requirements for a sequence of random number (Randomness and unpredictability)

While the main cons of the proposed generator in my point of view:

- This scheme is not efficient because of the relatively large number of iteration steps involved in its implementation.
- Difficult hardware implementation.
- The learning rate, which has critical effect of the neural network performance, didn't adopt in this architecture. This makes the weight adaptation relatively unstable or oscillated.
- It's difficult to estimate the period of the sequence, because the number of iterations in each layer depends on the initial conditions, which is generated by the key. In other words, the key and seed values effect on the performance of the generator.

7. References

1. W. Stallings, Cryptography and Network Security: Principles and Practice, 5th Edition, Pearson Education Inc., 2011.
2. Ü. Güler and S. Ergün, "A high speed, fully digital IC random number generator," International Symposium on Circuits and Systems (ISCAS 2010), Paris, France. May 30-June 2, 2010.
3. A. Rukhin, et al. "A Statistical Test Suite for Random and Pseudo-random Number Generators for Cryptographic Applications", NIST Special Publication 800-22, 2001.
4. S. Chatzidakis, P. Forsberg, and L. H. Tsoukalas, "Chaotic neural networks for intelligent signal encryption," IEEE 5th International Conference on Information, Intelligence, Systems and Applications, IISA 2014.
5. P. Singla, P. Sachdeva, and M. Ahmad, "A chaotic neural network based Cryptographic pseudo-random sequence design," 4th International Conference on Advanced Computing & Communication Technologies, ACCT '14, 2014.
6. F. Hsiao, Y. Tsai, K. Hsieh and Z. Lin, "Fuzzy Control for Exponential H∞ Synchronization of Chaotic Cryptosystems Using an Improved Genetic Algorithm," 11th IEEE.
   International Conference on Control & Automation (ICCA), Taichung, Taiwan. June 18-20, 2014.
7. S. Behnia, A. Akhavan, A. Akhshani, and A. Samsudin ," A novel dynamic model of pseudo random number generator," Journal of Computational and Applied Mathematics 235 (2011) 3455–3463.
8. A. Akhshani, A. Akhavan, A. Mobaraki, S.-C. Lim, and Z. Hassan, "Pseudo random number generator based on quantum chaotic map," Commun Nonlinear Sci Numer Simulat 19 (2014) 101–111.
9. A. S. Mansingka, A. G. Radwan, and K. N. Salama, " Fully digital 1-D, 2-D and 3-D multiscroll chaos as hardware pseudo random number generators," 55th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS) Circuits and Systems (MWSCAS), pp 1180-1183. August 2012.
10. Shiguo Lian, "A block cipher based on chaotic neural networks," Neurocomputing 72, pp. 1296–1301, 2009.
11. Ö. F. Ertuğrul, "A Novel Approach to Synchronization Problem of Artificial Neural Network in Cryptography," American Association for Science and Technology, AASCIT Communications, Volume 1, Issue 2, pp. 27-32, July 2014.
12. A. Yay and Y. Kutlu, "Neural network based cryptography," Neural Network World 24 (2), 177-192, 2014.
13. S. D. Joshi, V. R. Udupi, and D. R. Joshi, "A novel neural network approach for digital image data encryption/decryption," IEEE International Conference on Power, Signals, Controls and Computation (EPSCICON), June 2012.
14. S. Jhajharia, S. Mishra, and S. Bali, "Public key cryptography using neural networks and genetic algorithms," IEEE 6th International Conference on Contemporary Computing (IC3), pp. 137-142. Aug. 2013.
15. P. Kotlarz and Z. Kotulski, "Neural network as a

programmable block cipher," Advances in Information Processing and Protection, pp 241-250, 2007.

16. A. A. El-Zoghabi, A. H. Yassin, and H. H. Hussien, "Survey report on cryptography based on neural network," International Journal of Emerging Technology and Advanced Engineering, vol. 3, Issue 12, Dec 2013.
17. A. G. Bafghi, R. Safabakhsh, and B. Sadeghiyan, "Finding the differential characteristics of block ciphers with neural networks," Information Sciences 178, pp. 3118–3132, 2008.
18. L. P. Yee and L. C. De Silva, "Application of multilayer perceptron networks in symmetric block ciphers," International Symposium on Neural Networks - ISNN , vol. 2, pp. 1455-1458, 2002.
19. M. Arvandi, S. Wu and A. Sadeghian, "On the use of recurrent neural networks to design symmetric ciphers," IEEE Computational Intelligence Magazine, vol. 3, no. 2, pp. 42-53, May 2008.
20. J. M. Zurada, Introduction to Artificial Neural Systems, West Publishing Company, 1992.
21. (2014, Dec 15). Chaotic System, Available: http://www.businessdictionary.com/definition/chaotic-system.html#ixzz3LujgV3Th.
22. P. Y. Kostenko, A. N. Barsukov, A. V. Antonov, and S. I. Sivachinko, "Recovery of binary message, masked with derivative of mackey–glass chaotic process," Radioelectronics and Communications Systems, vol. 52, no. 2, pp. 89–9, 2009.
23. D. Viswanath, "The fractal property of the Lorenz attractor," Physica D 190, pp.115–128, 2004.
24. E. McEvoy, "Using Matlab to integrate ordinary differential equations (ODEs)," June 17, 2009.
25. (2014, Dec 15). Lorenz system, available: http://en.wikipedia.org/wiki/Lorenz_system.
26. [Y. Li, D. Xiao, S. Deng, Q. Han, and G. Zhou, "Parallel Hash function construction based on chaotic maps with changeable parameters," Neural Computing and Applications, 20, pp.1305–1312, 2011.