1-1-2010

Developing pseudo random number generator based on neural networks and neurofuzzy systems

Kayvan Tirdad
Ryerson University

DEVELOPING PSEUDO RANDOM NUMBER GENERATOR BASED ON NEURAL NETWORKS AND NEUROFUZZY SYSTEMS

by

Kayvan Tirdad

MEng, Amir Kabir University (Tehran polytechnic), Iran, 2005

BEng, Islamic Azad University, Iran, 2002

A thesis

Presented to Ryerson University

In partial fulfillment to the

Requirements for the degree of

Master of Science

In program of

Computer Science

Toronto, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis or dissertation.

I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research

DEVELOPING PSEUDO RANDOM NUMBER GENERATOR BASED ON NEURAL NETWORKS AND NEUROFUZZY SYSTEMS

Kayvan Tirdad

MSc, Computer Science, Ryerson University, 2010

Abstract

Pseudo random number generators (PRNGs) are one of the most important components in security and cryptography applications. We propose an application of Hopfield Neural Networks (HNN) as pseudo random number generator. This research is done based on a unique property of HNN, i.e., its unpredictable behavior under certain conditions. Also, we propose an application of Fuzzy Hopfield Neural Networks (FHNN) as pseudo random number generator. We compare the main features of ideal random number generators with our proposed PRNGs. We use a battery of statistical tests developed by National Institute of Standards and Technology (NIST) to measure the performance of proposed HNN and FHNN. We also measure the performance of other standard PRNGs and compare the results with HNN and FHNN PRNG. We have shown that our proposed HNN and FHNN have good performance comparing to other PRNGs accordingly.

Acknowledgements

I owe my utmost gratitude and especially want to thank my supervisor, Dr. Alireza Sadeghian, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. His perpetual energy and enthusiasm in research had motivated all his advisees, including me. He was always accessible and willing to help me in my research. As a result, research life became smooth and rewarding for me. Throughout my thesis-writing period, he provided encouragement, sound advice, and lots of good ideas without which I would have been totally lost.

I am heartily thankful to all my professors and teachers at Computer Science Department Ryerson University who taught and gave me immense knowledge of the subjects.

I am sincerely grateful to my exam committee panel Dr. Ali Miri, Dr. Mark Fiala and Dr. Eric Harley for their time and patience to read through my thesis and give their valuable comments.

It is a pleasure to sincerely thank the administrative staff members Mrs. Lucia Flaim and Mrs. Lori Haddad for their tolerance and extended help whenever needed.

I would like to extend my gratitude and thank Mr. William Zereneh and Mr. Ivan Rubiales for their sincere help in technical support.

I am greatly indebted and want to specially thank Dr. Hooman Tahayori and Dr. Mehrdad Tirandazian for their countless valuable suggestions and kind support during my thesis completion.

To my loving wife, for her generous support, encouragement, and tolerance without which it would have been difficult to complete my research.

To my parents, who have been a role model and have encouraged, supported and motivated me all throughout my life.

To my dear sister and brother, who support me in all aspects of life.

Table of Contents

List of Figures

List of Tables

"Chance governs all"
~ Milton

Chapter 1

Introduction

1.1 History of Random Process

Literal meaning of Random in English language is disorder, unpredictable and without any purpose. In statistics, the term randomness is used to emphasis on the well defined statistical properties, such as lack of bias or correlation. When a variable is said to be random, it means that the variable follows a given probability distribution. The term of "*arbitrary*" implies that there isn't such determinable probability distribution. In all the arguments that relates to random topics, the difference between random and arbitrary must always be considered and differentiated [1].

A sequence of numbers that contain no patterns and regularities is known as statistically random sequence. When a number is chosen arbitrarily from some specific distribution it can be called as a random number. Such numbers are almost expected to be independent with no correlations with successive numbers. Random numbers generated by computers are called pseudo random numbers [2] [3] [4] [5].

However, when randomness is used within the context of uniform distributions, then random sequences are generated in an almost predictable fashion using some mathematical formula and stochastic process, but since the distribution area is large enough and normalized, hence the prediction is almost impossible [1].

Random process is defined as one whose consequences are unknown [6]. The term "random" is usually used when the output of a process is unpredictable.

The history of random processes backs to ancient times. When, the concept of fate was the translation of chance or random event. Throwing a dice to determine fate was common among ancient peoples. Most ancient cultures used various methods of divination to attempt to circumvent randomness and fate [5].

1.2 Usages of Random Numbers

Random numbers are widely used in gaming industry. Gambling is the most prevalent act of all times which directly relates to random events. Gambling is based on random events such as throwing a dice, flipping a coin or shuffling of playing cards that can be found in all human history [7][8].

In most modern societies and cultures, legalized gambling plays an important role in economics. It presents a very important economic aspect of modern society. So where ever gambling is legalized, to ensure the unpredictability of the outcome results, plenty of stringent regulations exist. These rules cover different aspect form of physical devices, like dice, cards decks and all electronic devices used for gambling, etc. to employ the gambling facilities like Casinos [9].

Beside the gambling and gaming industry, random numbers have many other usages in today engineering problems. Random numbers are used widely in device testing and simulation like white spectrum generate by sequences of (pseudo) random numbers, simulating network traffic with certain statistical properties in order to perform an off-line test. Random numbers are used in computer simulation *(Monte-Carlo simulations)* to model all non-idealities (e.g. noise, device mismatches, and particle interactions) which exist in real systems [5].

Random numbers are present as a string of bits (sequence of bits). Depending on the type of the problem that computer tries to simulate, the size of this string could vary- even hundreds of

gigabytes [1]. Scientists try to simulate complex systems every day. Increasing the numbers of experiments and increasing the levels of complexity, demand for bigger amount of random numbers grow rapidly.

Random number generator plays a fundamental role in computer security issues. By increasing the demand for secure communication, secure storing and secure management of sensible data over public networks, specifically internet data encryption techniques have expanded rapidly. On the other hand by increasing wireless communication new high-security standard of data encryption was born [10].

The process of converting ordinary data (*plaintext*) into something unintelligible (*ciphertext*) is defined as encryption; decryption is the reverse of encryption process. A cipher is a pair of algorithms which perform this encryption and the reversing decryption. A cryptographic key is a secret parameter (it is known only by the communicants) for the cipher algorithm that controls the output of cipher algorithm. Keys are fundamental in modern cryptography as ciphers without keys, though very common in the early history of cryptography, there are bunch of chipper without key that are trivially breakable [5].

Modern cryptography algorithms have transferred all the unintelligibility from the cipher to the key [5]. During the last few years many standard cryptography algorithms were developed; and the most common ones are the DES [11] and the AES [12][1].

These algorithms are public and universally studied for many years, and very few numbers of attacks - on a specific condition - are reported about them. The key in these algorithms is

[1] DES was withdrawn after the AES was adopted in 2001 by US government.

presented as a string of bits. The key size is from hundreds to thousands of bits depending on the type of the algorithm.

There is a well known comment in security that says, *"A chain is only as strong as its weakest link"*, which means same high-security standards should apply for the generation and the distribution of the security key. If the generation or distribution of the key was not secure the security of the whole system would be under threat [5].

The key is secret and it is very critical that the key would not be guessed by anyone or any computer program. For this reason the key is chosen randomly, i.e. the key strings consist of random bits. If the key is not truly random, by looking at the sequence of bits composing the key, some patterns or some regularity would be found, and this helps to guess the key from part of it, then security of the system would be under threat. In summary, the performance of a good cryptography algorithm is tied up to performance of good random numbers [13] [14].

Random number generator is widely used in security and is very essential in cryptography, in a way that most cryptographic procedures require random numbers, e.g. in generating the key material. Random number generators (RNG) used in cryptographic application; typically produce a sequence of bits consisting of zeros and ones which can be combined into blocks or subsequence of random numbers [1].

1.3 Different Type of Random Number Generators (RNG)

RNGs are divided in two basic classes called deterministic and nondeterministic. The deterministic random number generators produce a sequence of bits from an initial value (seed) with a specific algorithm, and nondeterministic generators produce and generate the outputs that is unpredictable and depend on some physical source, outside of the human control [1][15].

1.3.1 Pseudo (Deterministic) Random Number Generators

A pseudo random number generator (PRNG) is a deterministic algorithm that generates a sequence of numbers. They are computed starting from an initialized vector, called *seed* [16] [17]. The seed is the only presenter of uncertainty in these pseudo random number generator algorithms. The seed is the initialization value of pseudo random number generator algorithms. Some of PRNG algorithms acquire the seed from the hardware of the system i.e. from the lower bits of the system clock. According to John von Neumann this process is not random at all [16],

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

The property of periodicity is common among all of PRNGs, but a true random sequence can never have this property. The reason for this property relies on the deterministic nature of algorithm. A periodic system is a system that repeats itself after some iterations and so is completely predictable after the first period. It is possible to build PRNG with periods so long that no computer could ever complete a single period. However, some people believe that it does not change the fact that a pseudorandom generator is not a random generator [5].

1.3.2 True (Nondeterministic) Random Number Generators

Nondeterministic random number generators are based on the direct observation of a physical process that has random-like features i.e. roll of a dice or the flip of a coin. These types of systems are called True Random Number Generators (TRNG). Most of today's TRNGs are based on the microscopic phenomenon, Such as the Brownian motion, quantum effects, Johnson thermal noise over a resistor and the Shot noise in vacuum tubes. All these phenomena placed in the basic physical processes at a very low level. Usually these phenomena referred as a "noise".

The radioactive nuclear decay of instable isotopes is a good example of a quantum noise which is widely used [1] [5].TRNG is build by embedding a random like phenomena in system and adding an observing part to the system. The system acts as an interface that converts uncertainty in observed phenomena to the random numbers [18].

1.3.3 Comparison of PRNGs to TRNGs

Depending on the application, both TRNG and PRNG have their own strengths and weaknesses. Generally speaking limitations of one is advantage of the other and vice versa. Table 1 shows the most important features that an ideal RNG should have, and shows what features PRNGs and TRNGs have. So based on the application needs, TRNG or PRNG would be used.

Table 1- comparing the features of TRNGs to PRNGs

Ideal RNGs Features	**TRNGs**	**PRNGs**
No periodicities	Yes	No
No predictability of output	Yes	No
No Dependencies Present	Yes	No
High Level of Security	High	Not as high as TRNG
Not Based on Algorithm	Yes	No
Fast and Efficient	No	Yes
Install and Run	Hard	Easy
Reproducibility Random Number Sequences	No	Yes
Cost	High	low
Need Manipulation	Yes	No

Determining the output of a well designed PRNG from prefect TRNG - just by knowing the current state of system and output of system - is an open question. Developing an efficient RNG

with all the features mentioned in Table 1, is an ultimate goal, and to this end researchers are trying to increase the performance of TRNGs and PRNGs.

In term of the usage, TRNGs are very useful in most of the security applications and in gaming industry. For simulation usages, however PRNGs are applicable. According to the amount of random numbers needed and the cost of the generation, TRNG is not suitable for simulation problems.

1.4 Importance of Pseudo Random Number Generators

One of the major problems in cryptography and other computer science concepts is developing a Pseudo Random Number Generator (PRNG) with acceptable quality in term of randomness. The acceptable qualities of PRNGs are very crucial for both cryptosystems and simulation problems. Huge demand is for developing a good quality PRNG in most of the computer science concepts such as Simulation and Cryptography, which has placed PRNG in center point of the problems, and lots of people try to solve. There are lots of algorithms suggested to work as PRNG in past years [1] [2] [15].

Linear Congruential, Quadratic Congruential 1 and 2, Micali-Schnorr, Blum Blum Shub, Modular-Exponentiation and G using SHA-1 are some examples of the PRNGS that have been developed and studied in the past [19].

1.5 Measuring the Quality of Pseudo Random Number Generators

By developing different PRNG there was a need to come up with procedures that could measure the quality of PRNGs. To this end, randomness was identified as a feature that would demonstrate the quality of PRNGs [6]. Moreover by increasing the speed of computers in recent years

anybody could use much more intense procedure to predict consequence. In particular, the increase in processing capacity of computers has led to the stronger Random Tests and newer PRNG with better quality of randomness has been devised. During the past few years both Random number tests and PRNG has progressed rapidly [2] [20] [21].

To check the quality of the random number generators and evaluate their suitability for a particular cryptographic application, a number of tests exist that enable the checking of randomness and unpredictability of the results of software or hardware based random number generators [21]. Statistical tests are widely used [19][22] and provide good measurements for randomness. For example, National Institute of Standards and Technology (NIST) provides statistical standards such as NIST Test Suite that measures the randomness degree of binary sequences. NIST Test Suite consists of a number of tests that all together search for different types of non-randomness that might exist in binary sequences [19].

1.6 Soft Computing Techniques for Developing PRNGs

Information security is one of the most important technology concerns today. Cryptosystems are mainly focused on addressing this concern. Different kinds of cryptosystems for tackling different problems and answering different needs in security field are developed. Cryptosystems provide an answer for protecting the integrity, confidentiality, and authenticity of information resources. In parallel, through availability of powerful computers and developing tools, many techniques for breaking cryptosystems are developed. By developing more powerful breaking techniques the demand for stronger cryptosystems has increased rapidly.

The other issue is related to increase the power of computation according to more laws. The increase in the power of computation has led us to more complex algorithm in our cryptosystems.

Also during the past years soft computing techniques such as neural networks, fuzzy systems, Genetic algorithm, and Rough sets have been developed rapidly. Some of these techniques are applied on development of PRNGs. These techniques showed that there is a great potential for tackling cryptosystems problems by applying soft computing methods. In the following section, usage of neural networks and Fuzzy logic in development of PRNGs are shortly explained.

1.6.1 Usage of Neural Networks in development of PRNGs

Neural network is a computational model that has been inspired by biological Neural Network. Connections of groups of artificial neurons made the structure of neural network. The structure of Neural Network changes by flow of information through the networks in learning phase. Neural Networks (NN) specifically have developed hugely in recent years. By developing new Neural Networks model and study different variation of them, different characteristics and features have been revealed. Some features of NN make them suitable for specific security problem.

Some features of Neural Networks like complexity, parallel ability, nonlinearity, unpredictability make them suitable to work as a random number generator. Different types of neural networks and also combination of them have been used as random number generator component in cryptosystem. Hopfield is a type of neural networks which is used as pseudo-random number generator [22] [23].

1.6.1.1 Usage of Hopfield Neural Networks in Development of PRNGs

Hopfield Neural Networks (HNN) are recurrent neural networks that are developed by John Hopfield. This type of neural network is widely used as content-addressable memory. The basic form of these networks uses linear activation function. The basic form guarantees convergence to

a local minimum. To store patterns in HNN the network should be set in a way that desired patterns are placed in local minimums. Notice that in HNN convergence to one of the stored patterns is not guaranteed [24].

There are some special features that make Hopfield Neural Networks (HNN) different from other types of neural networks. First, function approximation capabilities, which make HNNs a powerful tool in many scientific disciplines. Second, their generalization capability, which is a kind of nonlinear operation, that makes HNNs suitable for generating random number.

Hopfield Neural Network approach is based on decreasing energy by finite cycle. By any changes in the input of network, the output is calculated with, not previously encountered inputs, hence it tries to decrease energy, and after some cycles the network converges. Before the network convergence, the output is unpredictable, so if the network does not converge in any state, the output remains unpredictable. This feature is one of the mainly used one, in the random number generators. The main idea is finding some ways to make network not to converge [23].

1.6.2 Fuzzy Systems and Random Number Generator

Lotfi Zadeh in 1965 formalized fuzzy set theory [25] and in 1973 he applied fuzzy set theory to control systems [26]. He found that multi valued logic system could be much more useful than traditional binary logic system for solving some problems. Fuzzy logic added *"higher machine intelligence quotient"* to control systems [27].

Main advantage of Fuzzy logic system is providing a platform to acquire human knowledge and feeding it to the system. There are a huge number of applications of fuzzy systems, such as classification, handwriting recognition, voice recognition, image stabilization and data mining. Also there are some applications of fuzzy logic in developing a PRNG [28].

1.7 Motivation

Different types of Neural Networks are applied to different problems in cryptosystems. There is a huge interest to apply Neural Network in different aspect of cryptosystems. Before applying Neural Networks to any aspect of cryptosystem such as hash function, visual cryptography, steganography, steganalysis and PRNGs, we should examine features of problem and compare them with features of specific Neural Network that want to be applied. In recent years different kind of neural networks applied to different aspect of cryptosystems such as Encryption schemes, Secret Key protocol, random number generator, Prime factorization, Hash table, Digital Water Marking, Steganalysis and etc.

Concept like usage of NN for digital water marking, using different NN for Staganalysis or public/private key exchange protocol by NN and usage of NN in PRNGs was always interesting [4].

The other motivation for us is the ability to make a NN in chip format. So if we develop PRNG based on NN with good performance, it would be easily convertible to hardware format and it would have big usage in simulation projects.

There is a need to check previous PRNGs that we have with new test suit that exists and we have to be aware of their reliability with today computational power. By recent advancements that has happened in PRNGs tests and also in computation power it is much easier to exploit any non-randomness in PRNGs than couple of years ago. Hence another motivation for us is to answer this need and check the performance of old PRNGs with new available PRNGs test.

1.8 Contributions of the Thesis

The following concepts have been investigated in this thesis:

- In this thesis we look at the definition and usage of different random number generators. It makes our goal clearer and helps us acquire better understanding of the problem.
- Test suite for randomness is the other side of PRNGs problems. So in this thesis we investigate different type of random number test suite, and try to mention the history of these test suites.
- In this thesis by looking in depth to each test of NIST PRNG test suite we acquire more detail about the result of each PRNG and find their weakness.
- We look at different PRNGs that developed in previous years; we mention the history of PRNGs and study the algorithm of each PRNG to produce random numbers.
- The performance of previous PRNGs have been studied in this thesis. We compare the performance of each PRNG with each other.
- We Implement a Hopfield Neural Network PRNG and evaluate its performance in different conditions.
- We try to make some improvement in our Hopfield Neural Network PRNG by embedding Fuzzy system in each neuron and reducing the number of neurons. Also, we evaluate the performance of final Fuzzy Hopfield Neural Network PRNG.

1.9 Structure of the Thesis

In this thesis, in chapter one, we have discussed a brief introduction on random number and random number generator. We have looked at the history of random number and usage of it. We have looked at the problem on how to measure the randomness and different random test suite as

well. We examined different type of RNGs and compared them to lightening problem. Also we have prepared a brief introduction in neural networks and fuzzy system in chapter one.

In chapter two we looked at the algorithm and usage of some previous PRNGs. Also we investigated the different techniques for measuring the randomness. Specifically we overviewed the NIST PRNGs tests suite and checked all of its tests with more details. Further, different approaches which have been used to develop PRNGs by using different type of neural networks and also fuzzy logic systems are reviewed.

In chapter three some of previous different PRNGs, have been presented. Also we have talked about how we have implemented our Hopfield Neural Network PRNG (HNN PRNG). Next PRNG system presented in this chapter is based on Neuro-Fuzzy application. We have tried by combining fuzzy logic and Hopfield neural network, and developed a FHNN PRNG that does not have the weakness of our HNN PRNG. In chapter four we have showed the result of our HNN PRNG in different conditions. We have discussed the strength and weakness points of our HNN PRNG and we tried to conquer those weakness points. After that we check the result of our FHNN PRNG. Also, we have compared the result of HNN PRNG and FHNN PRNG with other PRNGs.

Finally, in chapter five we have our conclusion of this thesis. We review the results and conclude the final points of our systems. Also we look at the possible future works in this field.

Chapter 2

Background of PRNGs

It may be taken for granted that any attempt at defining disorder in a formal way will lead to a contradiction. This does not mean that the notion of disorder is contradictory. It is so, however, as soon as I try to formalize it.
~ Hans Freudenthal

2.1 Some well known PRNGs

PRNGs, with their long history in Computer Science, always have been controversial. In particular, the deterministic RNGs (PRNGs) are always shown to suffer from artifact patterns in their output. These artifact patterns made them venerable to statistical tests. Most of the developed PRNGs in the past years, do not pass the recent statistical tests. Some of their main problems would be categorized as:

I. Very short period for some specific seed (weak seed).

II. Not presenting uniform distribution when generating huge amount of data.

III. Vulnerable correlation between successive and future outputs, that makes the output guessable.

Classical example of such PRNG is RANDU. RANDU is a PRNG that was used during 1960's in the mainframe machines. RNADU suffered from serious flaws (mainly the output was not random), but unfortunately they were not recognized for decade. Figure 1 shows that the output of RANDU is always on one of the 11 plains. Donald Knuth about RANDU has quoted *"...its very name RANDU is enough to bring dismay into the eyes and stomachs of many computer scientists!"* [17].

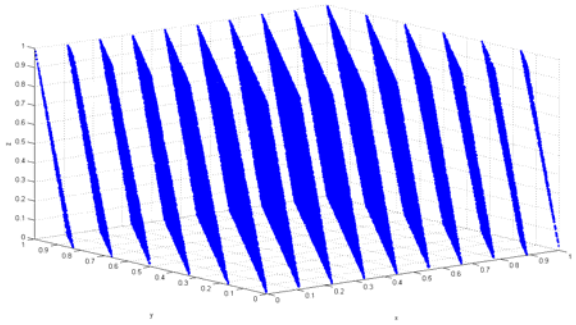

Figure 1- RNADU Outpots [29]

2.1.1 Linear Congruential

Linear Congruential is one of the oldest Pseudo Random number generators that is defined as [30] :

$X_{n+1} = (aX_n + c) \bmod m$ where X_n is the sequence of pseudorandom values,

m : the "modulus" $0 < m$

a : the "multiplier" $0 < a < m$

c : the "increment" $0 \leq c < m$

X_0: the "seed" or "start value" $0 \leq X_0 < m$

All of these parameter are integer constants that specify the generator.

Form a given "seed" X_0 (2-1)

$$X_{i+1} = (aX_i + c) \bmod M$$

Example $X_{i+1} = 7\, X_i \bmod 11 \longrightarrow 1, 7, 5, 2, 3, 10, 4, 6, 9, 8, (1)$

$X_{i+1} = 6\, X_i \bmod 11 \longrightarrow 1, 6, 3, 7, 9, 10, 5, 8, 4, 2, (1)$

The Linear Congruential PRNG will have a full period for all seed values if and only if c and m

are relatively prime and $a - 1$ is divisible by all prime factors of m. Linear Congruential PRNG maximum period length, could be m.

Linear Congruential based PRNGs are extremely sensitive to the choice of the coefficients *c*, *m*, and *a*.The main advantage of Linear Congruential PRNG is speed. Linear Congruential based RNGs require a little amount of memory and they are incredibility fast. One of its disadvantages is its weakness in randomness and security, so in the application that high level of randomness or security is needed, this type of PRNG is not suitable. Some researchers reported successful crack attempt on Linear Congruential based RNGs [31] [32].

2.1.2 Quadratic Congruential 1, 2

Both Quadratic Congruential 1 and Quadratic Congruential 2 are based on Linear Congruential but they are nonlinear [33].

Quadratic Congruential 1 is defined as:

$$X_{n+1} = (aX_n^2) mod\ m \tag{2-2}$$

Quadratic Congruential 2 is defined as:

$$X_{n+1} = (aX_n^2 + bX_n + c) mod\ m \tag{2-3}$$

Like Linear Congruential PRNG, maximum period length of Quadratic Congruential 1, 2 are m. these types of PRNGs present weaker linear properties [17].

2.1.3 Cubic Congruential Generator II

Cubic Congruential II PRNG is slightly different from the Quadratic Congruential PRNG. Cubic Congruential PRNG uses cubic of input for calculating the result to gain better distribution in all dimensions. Cubic Congruential II PRNG is defined as [19]:

$$X_{n+1} = (aX_n^3) mod\ m \tag{2-4}$$

2.1.4 Modular Exponentiation

Modular Exponentiation is widely used as random number generator [34].it applies an exponentiation on a modulus. In fact, "modular exponentiation" means calculating the remainder c where there is a positive integer b we called it base raised to the e^{th} power - e is referred to as an exponent - divided by a positive integer m, and m is called the modulus,

$$c \equiv b^e (mod\ m) \tag{2-5}$$

The fact that remains that, by using Modular Exponentiation we could build one way function, and there are lots of PRNGs, which are based on such functions [34].

2.1.5 G using SHA-1

G function is a one way function that has been specified in Appendix 3 of FIPS 186 [35] and is based on the Secure Hash Algorithm (SHA-1). FIPS 186 [35] explains a digital signature standard algorithm, this algorithm needs secure random number to use as a private key, so by developing a one way function they generate desired private key. For developing that one way function which we call G function, two methods are presented [35] the first one is based on SHA1

algorithm and the other one is based on DES algorithm. Nowadays performance of G function based on SHA1 used as comparison for other PRNGs [36].

2.1.6 Micali-Schnorr

Micali-Schnorr pseudorandom bit generator has been considered as an existing secure PRNGs. Micali-Schnorr PRNG is an efficient version of RSA pseudorandom bit generator. Micali-Schnorr PRNGs are relatively slow in comparison with the other PRNGs; the reason relies on huge use of modular multiplications that have a slow procedure. Moreover, by implementing this algorithm on a hardware that has circuit for modular multiplications the speed problem would be solved [36].

2.1.7 Blum Blum Shub

Blum Blum Shub (BBS) PRNG is proposed in 1986, and it is based on Prime factorization [37].

$$X_{n+1} = X_n^2 \ mod \ M \tag{2-6}$$

Where $M=pq$ is the product of two large primes p and q. At each step of the algorithm, some output is derived from x_n; the output is either the bit parity of x_{n+1} or one or more of the least significant bits of x_{n+1} . In BBS, x_i would be calculate directly from:

$$X_i = \left(X_0^{2^i \ mod \ (p-1)(q-1)}\right) mod \ M \tag{2-7}$$

BBS are extremely slow and hence, it is not suitable for simulation, but it has been proved to have good security features.

2.1.8 Other random numbers

There are other types of PRNGs such as Inversive Congruential Generator, Logged Fibonacci Generator, Linear Shift Register, Mersenne Twister and Xorshift etc. Since these PRNGs are not used for performance comparison, therefore we do not discuss them here.

Cipher algorithms and cryptographic Hash functions were often used as PRNGs. Block cipher in counter mode, cryptographic hash functions in counter mode and stream cipher techniques are some example of this type of PRNGs.

Other type of PRNGs is based on External entropy. This type of PRNGs is widely used in different operating systems. CryptGenRandom, Yarrow, LavaRnd and Fortuna belong to this type of PRNGs.

2.2 Randomness Tests

Random numbers are expected to be independent with no correlation between successive numbers. The term “random” is usually used for the output of a process which is unpredictable, but without qualification, “random” is being used as random with some uniform distributions. It means that they are generated in an almost predictable fashion using some mathematical formula and stochastic process, but if the distribution area is big enough and most normalized, the prediction would be almost impossible [38].

It is not sufficient to judge the randomness of a bit sequence only by its appearance. In other word human is not good enough to detect or even generate random bit sequence [17]. Knuth shows that if you ask a random person to write down a random sequence of bits with the length of 50, there is a big chance that he fails. Most people try to avoid repeating previous bits when they want to generate random bit sequences. On the other hand, if you show people a random sequence most

of them detect it as non-random sequence. Human nature is designed to look for patterns and similarities, but not for randomness [17].

We need a test to determine the level of randomness, and based on the level of randomness we may decide to consider a sequence as random or not. Gaining higher level of randomness does not mean getting a flawless RNG, it just means that we have acquired higher level of confidence for our PRNG [39]. “Deciding if the result of a RNG is random or not” is meaningful with respect to the level of acceptance. It means that a RNG that is considered as random number generator could sometimes generate nonrandom sequence, and a RNG that is considered as nonrandom number generator could produce random bit sequence occasionally, either software or hardware based.

To check the results of random number generators and determining whether or not they are suitable for a particular cryptographic application, some tests are developed that check the randomness and unpredictability of the results [21]. There are different measurements of randomness that are based on complexity, transforms, and statistical tests or their combination. Statistical tests have shown good measurement for random test and nowadays most researcher use this kind of package for randomness test [19][30].

With the advent of more powerful computational devices in recent years, weaknesses in generated random sequences would be more easily revealed. Therefore, there is an increasing need for both stronger PRNGs and more accurate randomness tests [20][37].

According to various type of non randomness that may exist in Random bit sequences it is not practical to find non randomness patterns by just using one test. Most of the statistical tests are collection of tests. This collection is generally known as ‘suite’ or ‘battery’ of statistical tests [40].

While it is impossible to mathematically prove that a RNG is the best random number generator, using RNG test suites help determining the level of randomness of RNG with some level of confidence. There is a chance that RNG would not acquire good result in the entire tests of RNG test suite. By increasing the number of tests that RNG pass with good result the level of randomness of RNG will increase.

For testing RNG, RNG test suite feeds with sequences or subsequences of RNG output. By increasing the length of the sequences the chance of finding any nonrandom pattern increases and RNG could pass with higher level of confidence. Moreover by increasing the number of sequences the chance of finding any none uniformly behavior of RNG will increase [19].

During the past years, different RNG test suites are developed. The most famous one is the test suite that was present by Donald Knuth, known as Random Number Grue. This test is outdated now and compared to other existing tests its results are not satisfactory by today's standards [17].

Diehard was another RNG test suite that was developed by Marsaglia in 1995.It was developed to Address the problems Knuth RNG test had. Unfortunately after he retired nobody else followed his work, and his test is not maintained anymore [41].

ENT is another RNG test suite that was developed by john Walker in 1998 and its latest update was in 2008. This statistical test is suitable for cryptographic usage [42].

The most famous and popular RNG test is NIST RNG test suite. NIST RNG test suite is released in 2001, and since then it was updated every year [19][2].

---

[2] NIST (National Institute of Standards and Technologoy is a non-regulatory federal agency within the U.S. Commerce Department's Technology Administration. NIST's mission is to develop and promote measurement, standards, and technology to enhance productivity, facilitate trade, and improve the quality of life. http://www.nist.gov/

The binary sequences of the output result can be checked by some statistical packages such as NIST Test Suite. NIST Test Suite consists of fifteen tests, and is focused on different types of non-randomness which might exist in the binary sequences, resulting from such random number generators [19].

2.2.1 Some RNG testing issues

There are two questions about RNG test suite. First, is it true to use the same RNG tests for both PRNG and TRNG? Second, should RNG test suites be application dependent or not? [43]

By increasing the processing power and performance of RNG test suites the answer of these questions do not have significant influences in our result. Also for using which RNG test suite, here we suggest NIST. The reason is NIST RNG tests suite is standard in Random numbers community nowadays. NIST RNG tests suite is very popular among most of the users and this helps users to compare their results with each other for different RNGs.

We know that RNG tests use bit sequences of output of RNG to measure the randomness of RNGs. But the question is how many bit sequences with how much length is enough? There is no definite answer for this question however by increasing the power of computer, the number of bit sequences and their length may increase to gain better level of confidence about randomness of RNG.

As mentioned, most of the statistical RNG test suites such as NIST RNG test suite is combination of different tests. A RNG could gain different score in different tests. By testing a RNG with a test suite we reach a set of numbers that each number is associated with one specific test. For comparing two RNG with each other we have to compare a specific test result for both. It is common that one RNG has better result in one test and another RNG has better result in another

test. There are no priorities among tests in RNG test suites. There are few researchers that have tried to come up with priorities but generally speaking most of the people consider all the results with the same priorities [40].

Some of the tests in RNG test suite have some dependencies to each other, so it is possible that any nonrandom pattern in a bit sequence, affects the result of more than one test. Each RNG test suite deals with this type of dependencies differently. Also there are some tests in RNG test suite, that doesn't have any dependencies to each other [19].

Most of the RNG test suite, work on binary sequences, so for testing a RNG we should set it in a way that it produces a sequence of 0's and 1 's. Also adjusting a parameter of RNG test Suite is a job that needs some expertise. By missing the adjustments of the parameters of RNG test suites there is a great chance that some test could not be driven or the test generate a none valid result. By non valid we mean it shows a none random procedure as a random one.

2.2.2 NIST RNG Tests Suite in a Glance

NIST tests suite consist of fifteen different tests. Each test looks for different none randomness in bit sequences. Table 2 demonstrates an overview of NIST RNG test suite.

Table 2- NIST Statistical Test Suite in a Glance [40]

NIST Statistical Test Suite		
Test	Defect Detected	Property
Frequency (monobit)	Too many zeroes or ones	Equally likely (global)
Frequency (block)	Too many zeroes or ones	Equally likely (local)
Runs Test	Oscillation of zeroes and ones too fast or too slow	Sequential dependence (locally)
Longest Run of Ones in a Block	Oscillation of zeroes and ones too fast or too slow	Sequential dependence (globally)
Binary Matrix Rank	Deviation from expected rank Distribution	Linear dependence
Discrete Fourier Transform (spectral)	Repetitive patterns	Periodic dependence
Non-overlapping template matching	Irregular occurrences of a prespecified template	Periodic dependence and equally likely
Overlapping Template Matching	Irregular occurrences of a prespecified template	Periodic dependence and equally likely
Maurer's universal statistical	Sequence is compressible	Dependence and equally likely
Linear Complexity	Linear feedback shift register (LFSR) too short	Dependence
Serial	Non-uniformity in the joint distribution for m-length sequences	Equally likely
Approximate Entropy	Non-uniformity in the joint distribution for m-length sequences	Equally likely
Cumulative Sums (cusum)	Too many zeroes or ones at either an early or late stage in the sequence	Sequential dependence
Random Excursions	Deviation from the distribution of the number of visits of a random walk to a certain state	Sequential dependence
Random Excursions Variants	Deviation from the distribution of the number of visits (across many random walks) to a certain state	Sequential dependence

2.2.3 Deeper Look at NIST PRNG Test Suite

Statistical RNG test looks at Randomness as a probabilistic property and describes it in terms of probability. NIST RNG test suite aim is testing two different hypothesis; $H_0$and H_1.

- H_0 : *Null hypothesis* which assumes the sequence being tested is random.
- H_1: *Alternative hypothesis* which assumes the sequence being tested is not random.

By accepting or rejecting the null hypothesis in each test for a bit sequence a decision will be made. By applying mathematical method a critical value is determined. Based on that critical value the acceptance or rejection of H_0 is decided.

Statistical values for a bit sequence in each test are calculated and by comparing statistic values to critical value decision about acceptance or rejection of H_0 will be made. Hence, the statistical hypothesis testing procedure has two possible results, either accept H_0 (the data is random) or accept H_1 (the data is non-random). Table 3 shows four possible situations of statistical hypothesis testing procedure [19].

Table 3- Possible situations for statistical hypothesis testing procedure [19]

True Situation	Conclusion	
	Accept H_0	Accept H_1 (reject H_0)
Data is random (H_0 is true)	No error	Type I error
Data is not random (H_1 is true)	Type II error	No error

Type I error is a conclusion that happens when Data is random (H_0 is true) but the test accepts H_1 (reject H_0). On the other hand, Type II error is conclusion that happens when Data is not random (H_1 is true) but the test accepts H_0. Two other conclusions are true conclusions that would happen by statistical hypothesis testing procedure.

The probability of Type I error and Type II error are very low but not zero. *"Level of significance"* is the probability of a Type I error and is denoted by α. That would be set prior to running the test. Common values of α in cryptography are about 0.01. Calculation of Type II error denoted by β, is not an easy task [19][44]. Values of α, β and n - the size of bit sequence - are tied together. So although calculating of β is hard task but by selecting appropriate values for α and n, we could guarantee that β has a small value [19].

The result of each test is a *P-value* which shows, how random the sequence is. *P-value* ranges between 0 and 1, being equal to 1 means the sequence is totally random for that test, but if *P-value* is 0 means the sequence is not random at all. If *P-value* $\geq \alpha$ then *Null hypothesis* (H_0) is accepted and bit sequence is random, but if *P-value* $\leq \alpha$ then *Alternative hypothesis* (H_1) is accepted and the sequence is not random.

Commonly, $\alpha \in [0.001, 0.01]$. For example setting $\alpha = 0.01$, means from 100 sequences if one or zero of them is rejected the result is considered as random [1]. The other parameter in NIST PRNG test is the length of the each sequence that is denoted by n, and NIST has suggested, that it should be at least 10^6 .

For applying the test, NIST suggest applying the test to "x" number of sequences. "x" is at least inverse of "Level of significant" α. for example if α is equal to 0.01 NIST suggests to apply the test to at least 100 sequences. It is very critical to prepare sufficient number of sequences for testing. If the number of sequences is not large enough then the testing module reports a good result that indicate a good random number generator, but in fact according to the lack of feeding data the report is not valid, and there is a great chance that we face a nonrandom bit generator [1][19] [40]. In the following we explain each test with more details.

2.2.3.1 Frequency Test

The purpose of this test is to focus on the proportion of zeroes and ones in the whole sequence, In random sequence the number of zeros and ones are expected to be equal. The outcome of the test is a parameter called *P-value* that represents the ratio of the abstract of the summation of every binary numbers of the sequence in a way that the zeros are converted to value -1 to the square root of the number of the whole sequence. Thus, *P-value* indicates how random the sequence is [1][19].

2.2.3.2 Frequency Test within a block

This test relates to the proportion of ones in *M*-bit blocks, where *M* denotes the length of each block. This test determines whether the observed proportion of ones, within *M*-bit block, meets the expected proportion and the frequency of ones which would be approximately *M*/2 [1][19].

2.2.3.3 Runs Test

The test calculates the total number of the identical bits bounded with a bit of the opposite value before and after them, in an uninterrupted sequence. This series of identical bits are called Runs, and the purpose of this test is to determine whether this number of Runs with a various length are as expected for a random sequence, or to check how fast is the oscillation between the ones and zeros [1][19].

2.2.3.4 Test for the Longest Run of Ones in a Block

The test determines whether the length of the longest run of ones is as expected in a random sequence within *M*-bit blocks, after dividing the whole binary sequence into some parts with the length of *M*. As a matter of fact, after dividing the sequence into *M*-bit blocks, the frequencies of

the longest runs of ones in each block should be computed. In this test, only the runs of ones is necessary to be checked, because the irregularity in the expected length of runs of ones implies that, there would be an irregularity in the length of runs of zeros [1][19].

2.2.3.5 Binary Matrix Rank Test

By focusing on the rank of disjoint sub-matrices of the whole sequence, we check for the linear dependencies between fixed length substrings of the sequence. In this regard, the sequence is divided into R x C –bit disjoint blocks (R denotes the number of rows, and C indicates the number of columns in each matrix), in a way that each row is filled with C-bit blocks of the sequence. The number of blocks would be the result of the division of the length of the sequence, and the multiplication of R and C [1][19]. This test is also mentioned in Diehard battery test [41].

2.2.3.6 Discrete Fourier Transform (Spectral) Test

The purpose of the test is to find the periodic features and detecting the patterns which are repeated near each other. By focusing on the peak heights in the discrete Fourier transform of the sequence, this test defines the deviation of randomness by computing the number of peaks, to determine whether it exceeds a threshold. In February 2009, NIST announced that they have found a problem in Discrete Fourier Transform Test and they advised to disregard the result of this test.

2.2.3.7 Non-overlapping Template Matching Test

This is to define and detect generators which produce too many occurrences of a given non-periodic pattern, or pre-specified target string. An m-bit window is used to detect a specific m-bit pattern, after partitioning the sequence into M blocks. If the pattern is found, the window will be

reset to the bit after that pattern, and the search will be resumed, otherwise the window slides over by one bit only [1] [19].

2.2.3.8 Overlapping Template Matching Test

This is a variation of the non-overlapping template matching test, and the purpose of the test is searching for a specific *m*-bit pattern. Unlike the overlapping template matching test, in this test if there is no match and the pattern is not found, the window slides by one bit [1][19].

2.2.3.9 Maurer's "Universal Statistical" Test

This test computes the numbers of bits between matching patterns to indicate if the sequence is compressible without any loss of information - the compressible sequences are considered as non-random [1][19].

2.2.3.10 Linear Complexity Test

The test's focus is on the length of a Linear Feedback Shift Register (LFSR), and its purpose is to determine if the sequence is complex enough to be a random sequence. Random sequences are often characterized by the length of a Linear Feedback Shift Register (LFSR), if LFSR is too short, the sequence will be non-random. Statistically, the test measures how well the observed number of occurrences of LFSRs matches the expected number for each, under an assumption of randomness [1][19].

2.2.3.11 Serial Test

This test checks the frequency of all possible overlapping *m*-bit patterns, and to determine if every *m*-bit pattern has the same chance to appear as every other *m*-bit patterns in the entire

sequence. A random sequence has such a uniformity that, if the number of occurrences of the 2^m *m*-bit patterns across the whole sequence are required to be the same as expected from random sequence [1][19].

2.2.3.12 Approximate Entropy Test

Calculating the frequency of all possible *m*-bit patterns, this test determines if two physically adjacent overlapping block with the lengths of *m*, and then *m*+1, are the same as expected for a random sequence [1][19].

2.2.3.13 Cumulative Sums Test

Performing of cumulative sums test on a random sequence of zeros and ones should get close to zero. This is done by converting (0, 1) to (-1, 1), and then calculating the cumulative sums of the sequence. A large result shows a non-random sequence [1][19].

2.2.3.14 Random Excursion Test

Similar to the Cumulative Sums test, this test determines the number of cycles that have exactly *k* visits in a cumulative sum random walk. The cycle includes a sequence of steps of unit length taken periodically at random which begins and ends at the origin. Thus, the main purpose of the test is to define whether the number of visits to a particular state in a cycle is same as expected from a random sequence [1][19].

2.2.3.15 Random Excursion Variant Test

The test measures the number of times that a particular state is visited in a random walk, and defines the deviations from the same in a random sequence, and detects these deviations from expected number of visits to various states in a random walk [1][19].

2.3 Modern PRNGs

In recent years by increasing the computation power some researchers have tried to develop a PRNG based on new findings in other fields such as Evolutionary Computing, Chaos Theory, Fuzzy Logic, and etc.

Evolutionary computing is a new topic that is used to develop new PRNGs. Tuning self feedback shift register by genetic algorithm with the goal of gaining better PRNGs has been studied [45].The main problem of using evolutionary algorithm to develop PRNGs is time. Testing a bit sequence is very time consuming by itself, however using a pool of PRNGs that each should be tested separately to find which one is better makes the time complexity worst [45].

Recently some good RNGs based on chaos theory are developed. Some of these chaos based RNGs are also developed in a hardware level and are tested in real situation as well [5] [46] [47] [48] [49]. Fuzzy Logic Systems are also studied for developing RNGs [50].

2.3.1 Neural Networks for developing PRNGs

Among most of the techniques in soft computing that is applied in cryptosystem, Neural Networks have a wide variety of different usages. Different Neural Networks are applied to different problems in cryptosystems in past few years and also there is a huge interest to apply

Neural Networks in different aspect of cryptosystems. Hence, there is a high interest in applying Neural Networks on PRNGs like the other aspect of cryptosystem [4].

2.3.1.1 Neural Networks in Cryptosystems

In recent years different kinds of neural networks are applied to different aspect of cryptosystems such as Encryption schemes, Key exchange protocol, random number generator, Prime factorization, Hash table, Digital Water Marking, Steganalysis and etc.

Key exchange protocol is one of the most important parts of any crypto-systems. The basic idea is that both ends of the communication have their own secret key and both start to send some information and after transmitting some data they agree on specific secret key [51]. The benefits of using neural networks in key exchange protocols would be enumerated as, first, it is not based on the number theory. Second, there is no need to transmit N bit for secret key with the length of N bit, by using neural network both ends of communication could agree on the secret key with the length N by transmitting lesser bits than N [51]. The main idea is using two neural networks in both ends and train them to reach to the same key after some iterations [51] [52].Feed Forward neural Networks and Multi Layer Perceptron are two types of neural networks that are widely used for Key exchange protocol problem [51] [52] [53] [54].

Designing Symmetric cipher based on neural networks was studied [55] [56]also some attack method to this concept was investigated [57].

Prime factorization is another example for usage of the neural networks in cryptosystems. Lots of cryptosystems rely on the complexity of factoring a large integer number. The goal is to feed a large integer N to the system, calculating P,Q where $N= P \times Q$ and P, Q are prime numbers as

target φ(N)=(P-1)(Q-1), φ is Euler's totient function. For tackling this problem feed forward neural network (FFNN) was applied [58] [59].

Hash function is another aspect of cryptosystems, Confusion and diffusion property and also one way property of neural networks (in most of neural network especially when we have multiple inputs and one output) make them suitable for working as a hash function. Hash functions receive plain text with variable length as input and generate random text with fixed length as output. A tree layer feed forward neural network could work as a building block of hash functions [60] [61].

Visual Secret sharing was first proposed in 1994, is a picture encryption method that generates two or more pictures based on original picture which by aligning them together the original picture could appear [62]. Visual Cryptography could be applied by quantum neural networks (Q'tron).Q'tron architecture is similar to Hopfield neural networks, but in Q'tron neural network each Q'tron can be a noise-injected [63] [64] [65]. The most recent neural network which is used for visual cryptography is Pi-sigma neural networks [66].

Digital watermarking is one of the concepts that use neural networks vastly among cryptosystems. There is a main approach for tackling watermarking problems and all solutions are different variations of this method, First, splitting media to some blocks and then calculating watermark of each block [67]. Different kinds of neural networks for different purposes are applied in watermarking problem. The most used ones are back propagation neural networks (BPN). By extracting some coefficients of a block in frequency domain, feed those coefficients to BPN, next coefficients appear as output of neural network and then we change these coefficients. The point is that each time the block feed coefficients are fed to the BPN, the last coefficient must appeared as output and with this we could validate the media [68] [69]. There are some recent works on watermarking by combining more advanced neural networks and traditional water

marking methodologies. There is a successful experiment on combining spiking neural networks and wavelet transform to hide some information in pictures [70].Others, try to use Small World Cellular Neural Network (SWCNN) as an engine for watermarking, as a result the topology of that network is the secret key of images [71]. There are same experiments through applying wavelet transform on each block and also using neural networks [72].

Other technique that uses Neural Networks is lossless watermarking. Lossless watermarking refers to watermarking techniques that do not apply any change to original picture at last [73] [74].

For audio watermarking according to the fact that watermarked signal should be the same in human ears so the only possible way for watermarking is transforming signals to frequency domain then performing watermarking on signals. The usage of neural networks in signal watermarking is similar to that of image watermarking [75] [76].

Information hiding is a crypto system concept which Neural Networks are used in their implementation. Information hiding is a technique that hides message signals in host signal, such that the host signal is not distorted. Message signal and host signal could be any media such as image, audio or video [77]. There are two major classes of data hiding techniques and neural networks are used in both. Steganography refers to the techniques that are used for information hiding. In contrast Steganalysis refers to the technique that defines a signal which has hiding data or not. The goal in steganalysis is determining if a signal has hiding information or not, but not to reveal information which is hidden.

There are three different categories of information hiding which use neural networks.

First, Information hiding based on spatial domain [78][79], which is the oldest one.

Second, is Information hiding based on transform domains. In transform domain, host signal with hidden data transform to other domain and some statistical features of host signal is calculated in new domain. Discrete Fourier transform (DFT), Discrete Cosine transform (DCT) and Discrete Wavelet transform (DWT) are the most famous transforms which are applied in this field [77] [79] [80]. Some other transforms such as Haar transform has been used recently [81]. Feed forward back propagation neural networks are widely used for this category of information hiding [77] [81]. The other type of neural network that is used for this problem is Radial Base Function Neural Network (RBF) [82]. The most important feature that makes neural network suitable for this problem is the ability to classify parameters with non-linearity dependency. It is obvious that parameters extracted in new domain have nonlinearity dependency.

Third category is information hiding based on combination of different features. By using neural network that accepts this combination as an input and it tries to classify the steganalous media from non-steganalous media [80].

2.3.1.2 Neural Network as PRNG

Comparing to other aspects of cryptosystems, random numbers play a significant role in most of today's cryptosystems. Hence, random number generators are one of the most important components in today cryptosystems. A good random number generator has specific characteristics including uniform distribution, independency between the generated, cost effective and portability of random number generator, long sequence of data before repeating themselves and complexity of random number generator which makes it impossible to predict next number by looking at the previous one [83] [84].

Some kinds of Neural Networks have some characteristics that make them good candidates to be used as PRNG.

2.3.1.2.1 Multi-layer Perceptron Neural Networks (MLPNN) as PRNG

Normally we use Neural Networks as a black box that could solve specific problems. By using definite data as an input, it generates desired result as an output. In this sense, Neural Networks are predictable systems.

But if during training time of neural networks over fitting problem occurs, the Neural Network would show unpredictable output for input which is not used for training. This feature is widely used in most of the Neural Networks based random number generators [83] [85]. Multi-layer Perceptron (MLP) is one of the most famous Neural Networks that has this feature. By training MLP it learns from all the data that are presented and also it could generalize the data to what is not presented in training time. MLP fit the desired surface that represent by input data. If over fitting occurs in training time then MLP tries to fit a surface with higher degree than the desired one. So we couldn't predict the output of MLP for new input. The reason is, we don't know what surface MLP fits with is [83]. These random number generators represent powerful capability to generate real random numbers.

Some people try to combine these kinds of neural networks with other technique such as hash function to build more powerful random number generator. These kind of combinations represent better result with randomness test of random number generators [85].

2.3.1.2.2 Hopfield Neural Network (HNN) as PRNG

Hopfield neural networks are another kind of neural networks which are used as random number generator. A recurrent neural network, the Hopfield model was proposed in 1982 [86]. Learning in a Hopfield network is done by means of weight adjustment mechanism that directly relates to minimization of an energy function that decreases over time in each iteration and finally stabilizes in some point of the state space representing the problem. The basic idea of Hopfield Neural Networks (HNN) is to memorize some patterns as stable points by associating them with specific inputs to the network. That is, after some iteration the network goes to stable points in the state space that relates to the pattern that is memorized. Stability, therefore, is the most important features of Hopfield neural networks [24] [87] [88] [89]. In formula 2-8 you could find activations function that is used in basic Hopfiled Neural Network also symbols definition of 2-8 placed in Table 4. The structure of Hopfiled Neural Network presented in Figure 2 in next page.

$$X_i^{new} = \operatorname{sgn}\left(\sum_{j=1}^{n} W_{j,i} X_j - \theta_i + I_i\right) \quad (2\text{-}8)$$

Table 4- Symbols Definitions

Symbol	Definitions
N_i	Neuron number i
I_i	Input number i
X_i	Output number i
$W_{i,j}$	Weight of Connection from neuron i to j

By any change in input of Hopfield networks the network tries to decreases energy and after some cycle the network converges. Before network converges the output is unpredictable and if the network can't converges in any state the output remain unpredictable. This feature is the main focus for random number generator. The main idea is to find some way to make network not

converge in any time [22]. The ability to converge in Hopfield networks is strongly related to network architecture, network initial condition, and updating rule mode. The convergence of the network occurs when the weight matrix is symmetric. Thus, there might be some alternatives which cause the network not to converge, for e.g., (i) applying an initial asymmetric weight matrix consisting of large positive numbers in diagonal, (ii) letting two or more neurons activate simultaneously, and (iii) using large network and training it with orthogonal and uncorrelated patterns [23].

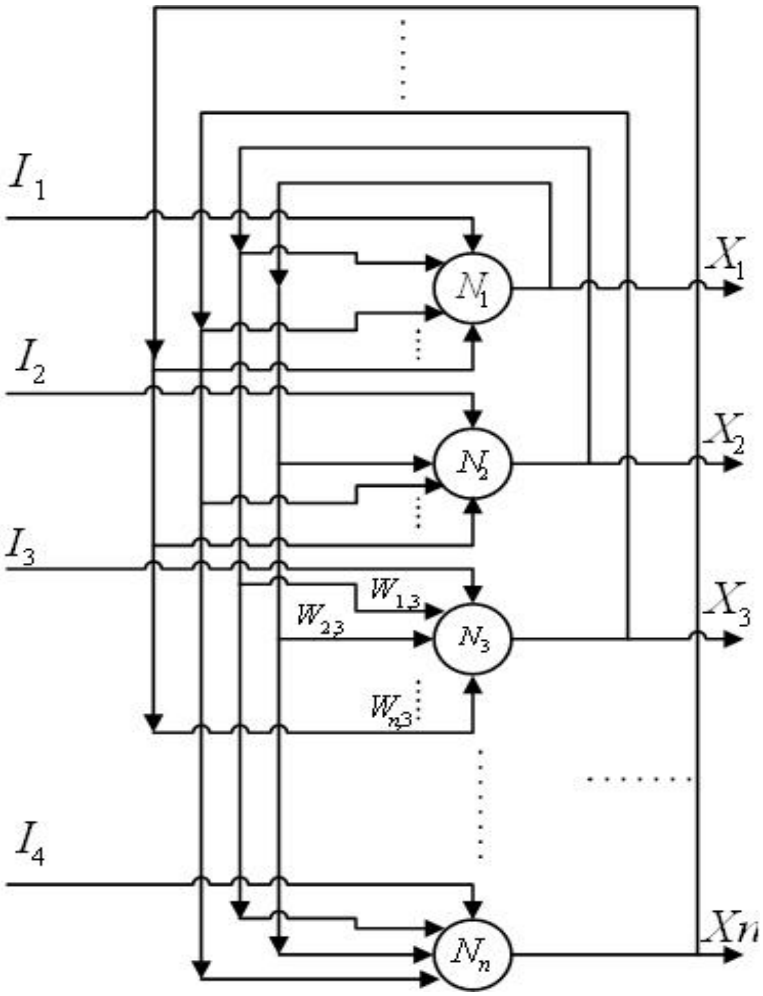

Figure 2- Hopfiled Neural Networks Structure [86]

The combination of Hopfield neural network with other techniques to produce better random number generator and also making automated Hopfield network parameter adjustment system for random number generator must be investigated in future [22].

2.3.1.2.3 Spiking Neural Network (SNN) as PRNG

Spiking neural networks are another kind of neural networks used as random number generators. Spiking neural networks are much more sophisticated neural networks that look more similar to actual neurons in cortex. Spiking Neural Networks are highly connected recurrent neural network models with information flow possible both to and from any given neuron. Brain mechanism that alters characteristics of neurons and also represents the dynamic of neural network is known as plasticity [84].

There are 3 types of plasticity which are used in spiking neural networks. Spike-timing dependent plasticity (STDP) refers to mechanism that if some neurons like $n_i$is active in discrete time interval following the activation of neuron n_j, then the connection between n_i and n_j is strengthened. STDP training makes networks to become stable rapidly. Anti-spike-timing dependent plasticity (Anti-STDP) works opposite to STDP. Anti-STDP weakens the connection between n_i and n_j. It also strengthens the connection between the neurons that do not meet the previous condition. Anti-STDP training leads networks to some kind of self disorganization. Dynamic of networks which are trained with Anti-STDP can enter chaotic regime. These networks have extremely long limit cycle. Size and average of the activity rate of these networks grow exponentially. These kinds of neural networks are well-suited for random number generator. Intrinsic plasticity (IP) works on thresholds of neurons rather than weight of connections. IP training tries to match sum of incoming weight with thresholds. In general IP training tries to

make all neurons in network to have the same average activity rate. Experiments show that by applying these 3 plasticity approaches together best random number generator system would be achieved [84].

Combination of Spiking Neural Network with different bit sampling techniques would generate even better results. Also using spiking neurons in other neural networks such as Hopfield to develop random number generator must be investigated in future [84].

So in conclusion, despite the usage of neural networks in other aspect of cryptosystems, Neural Networks has lots of features such as complexity, parallel ability, nonlinearity, unpredictability that make them suitable for random number generator. Different kind of neural networks and also different combination of neural networks with other technology have been used as random number generator component in cryptosystem.

Chapter 3

Nothing is random, only uncertain.
~ Gail Gasram

(Fuzzy) Hopfield Neural Networks as PRNG

3.1 An Introduction to Neural Networks

Neural networks are collections of data processing elements (neurons). These elements are highly interconnected to each other and build into the network structure. The structures of Neural Networks have been inspired by cerebral of human brain [90]. In the Figure 3, a typical biological neuron and its connections are presented.

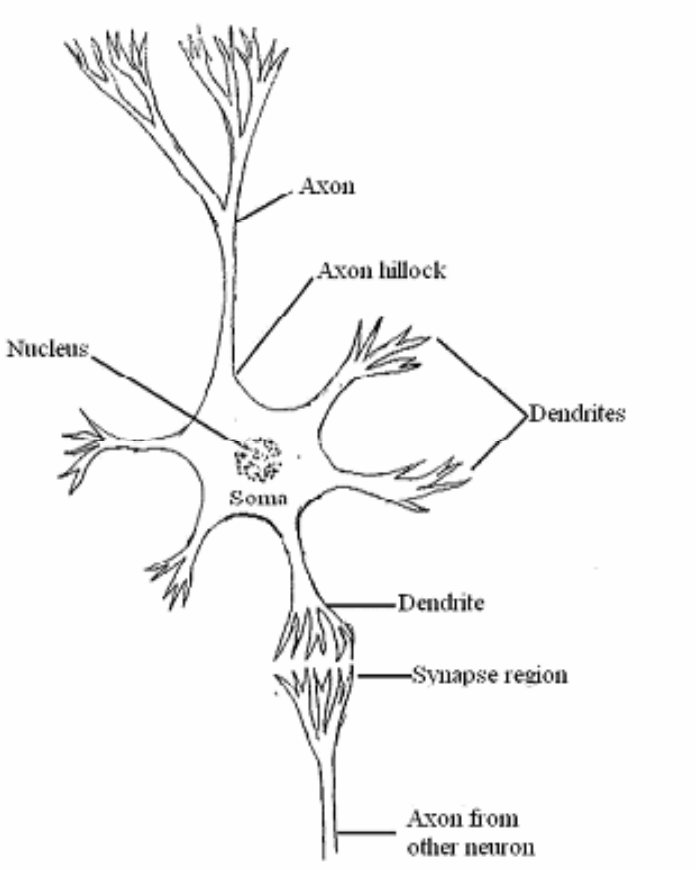

Figure 3- A typical biological neuron and its connections [91]

The neurons in human brain are working based on their chemical state. The electronic signals travel trough the neural networks by passing the connections between neurons. In 1943 McCulloch and Pitts proposed the first mathematical model for a neural network. That could be easily implemented in hardware.

Neuron is the most fundamental processing element of neural networks is a neuron, hence a simple neuron model is used as building block for all of the networks. In this model the weights present the synaptic strength of each neural connection [91]. Figure 4 shows the mathematical model of a neuron.

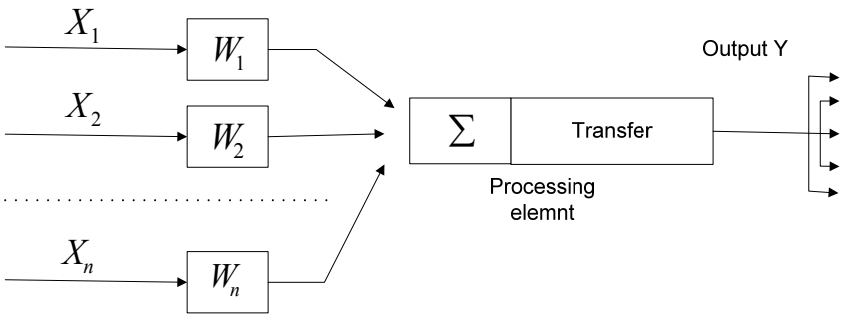

Figure 4- Mathematical model of a neuron [91]

In more recent models the bias parameter is also added. Equation 3-1 shows the mathematical presentation of such a model [92].

$$Y_i = F(\sum_{i=1}^{n} X_i W_i + \theta) \tag{3-1}$$

The state of neurons depends on Activation function (Transfer function). Activation function works as a threshold function that allows the signal to pass only if the summation of the activities of the neurons reaches a certain level. Different types of activation functions are proposed like linear and nonlinear activation function, discrete or continues activation function, etc.[92].

Figure 5 shows some examples of activation functions that are widely used in Neural Networks. These functions are developed in neural network toolbox of Matlab. Command of each function is mentioned under its corresponding graph.

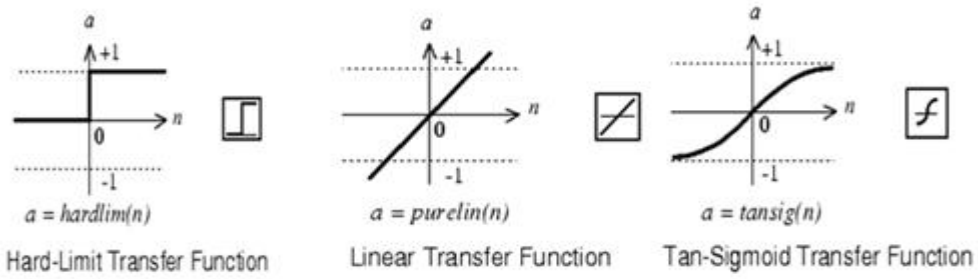

Figure 5- Activation Function Examples [93]

3.2 An introduction to Hopfield Neural Network

Hopfield neural networks were first presented by john Hopfield for the first time [24]. Hopfield Neural Networks are Recurrent Neural Networks i.e. Network with Cycles in their structures. Hopfield Neural Networks have a multiple-loop feedback system in their structure [86]. Each neuron in Hopfiled Neural Network is connected to all the other neurons but does not have a self feedback. All the neurons are input and output neurons [92]. Hopfield neural networks are placed in associative memory category. Associative memory or content addressable memory is a type of memory with function of retrieving stored pattern when it feed with portion or noisy version of pattern [86].

With respect to the activation function of the neurons in a neural network to be continues or discrete, Hopfield neural networks are recognized [94], and is denoted as Discrete Hopfield NN and continue Hopfield NN. Figure 6 present the structure of Hopfield Neural Networks.

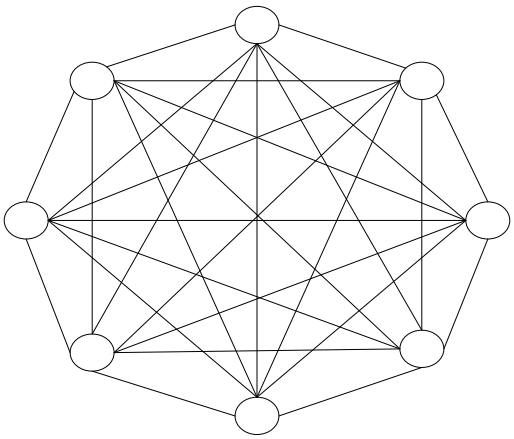

Figure 6- The Associative Network. All neurons are both input and output neurons [92]

All of the neural networks which are categorized as associative memory like Hopfield Neural Networks would be considered as dynamic systems [86]. Hopfield Neural Networks functionality are based on Lyapunov energy function. The Lyapunov energy function of Hopfield neural network is a monotonically decreasing function over time [86]. It means that by applying an input to the network, it tries to reach a stable point by decreasing its energy. Reaching stable points is guaranteed by following prerequisites of Lyapunov energy function theorem. Some of these prerequisite are like:

- There shouldn't be any self feedback in network (diagonal elements of weight matrix should be zero - weight matrix W, is matrix of numbers that each number $W_{i,j}$ present the weight of connection between neurons *i* and *j*)
- Weight matrix of network should be symmetric.
- Neurons should have nonlinear activation function
- The inverse of nonlinear activation function of neurons should exist

Before the network converges to a stable point the output of the network is not predictable [1].

3.3 Our Proposed Hopfield Neural Network as PRNG

We have developed a HNN considering the following conditions to guarantee non-convergence:

- The structure of the proposed HNN is fully connected, that is, self-feedback does exist. The structure of our Hopfield networks is presented in Figure 7. By connecting each neuron to itself we break one of the stability condition of Lyapunov energy function for Hopfield neural networks.
- We use a nonlinear function, $tanh(x)$ as the activation function for our neurons where x is the summation of all inputs of the neuron. Nonlinear activation function increases the complexity of neural network output (compare to linear activation function).

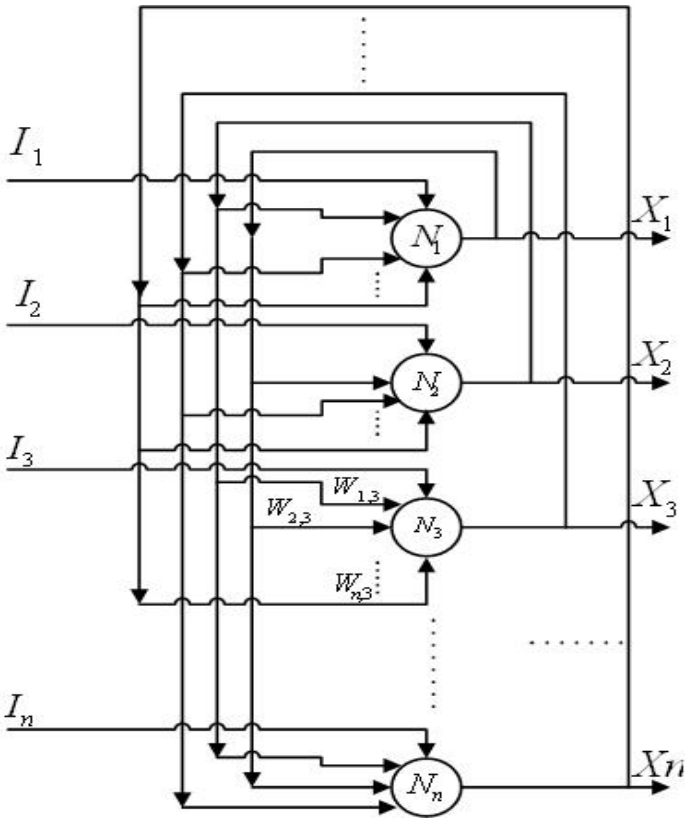

Figure 7 - Our Hopfield Neural Network Structure

- Weight matrix of our HNN is asymmetric where the upper triangle of weight matrix contains positive numbers and lower triangle of matrix contains negative numbers. By selecting asymmetric weight matrix we break one of the stability conditions of Lyapunov energy function for Hopfield neural network
- Our weight matrix is strongly diagonal dominated - Strongly diagonal weight matrix is shown in the following (3-2)

$$W_{jj} \geq \sum_{i \neq j} |w_{ij}| + \propto_j \tag{3-2}$$

Where $\propto_j$ is positive constant number, Hayken used same definition but instead of $\propto_j$ he used $\propto$, [86].

So the diagonal elements of the weight matrix contain large positive numbers compare to other elements of our weight matrix.

An example of used weight matrix is represented in Table 5.

- We use large number (100) of neurons.

If, for neuron i, $X_i \geq 1$ then, in the next iteration that neuron amplifies itself by the weight of the corresponding branch regardless of other inputs of the neuron. This is also valid for output $X_i \leq 1$ with diminishing impact on the neuron. In other words, this makes the neuron to be always fired with its output ≥ 1 or ≤ 1 in all iterations and accordingly amplifying or diminishing itself. To avoid this, Anderson put upper bound and lower bund for output of each neuron [95], but we set a condition (3-3) for each neuron j :

$$-1 \leq \sum_{i=1}^{n} w_{ij} \leq +1 \tag{3-3}$$

The output of each neuron in each iteration is calculated by (3-4)

$$X_i^{new} = \tanh\left(\sum_{j=1}^{n} W_{ji}X_j - \theta_i + I_i\right) \quad (3\text{-}4)$$

where I, X, W, θ, and n denote input, output, weight, threshold and the total number of neurons. In our HNN, θ is zero and in the first iteration I is 1.

Table 5- Example of Weight Matrix

Neuron Number	1	2	3	4	. . . . .	100
1	0.076686	0.000422	0.001538	0.007308	. . . . .	0.004925
2	-0.008889	0.069388	0.002479	0.009263	. . . . .	0.004435
3	-0.001074	-0.007553	0.063972	0.007767	. . . . .	0.007692
4	-0.009621	-0.001368	-0.004166	0.063631	. . . . .	0.004925
5	-0.005496	-0.007301	-0.002342	-0.002859	. . . . .	0.002267
6	-0.002744	-0.003131	-0.002483	-0.002923	. . . . .	0.008264
7	-0.004450	-0.003277	-0.005654	-0.007336	. . . . .	0.009012
8	-0.006538	-0.000811	-0.000223	-0.004170	. . . . .	0.006523
.	. . . .	. . . .	. . . .	. . . .	. . . .	. . . .
100	-0.004566	-0.006484	-0.007189	-0.003859	-0.008364	0.054545

3.3.1 Convergence Problem in Hopfield Neural Network

Convergence in a HNN is achieved when the output of each neuron reaches a stable state or oscillates between a limited number of states [89].

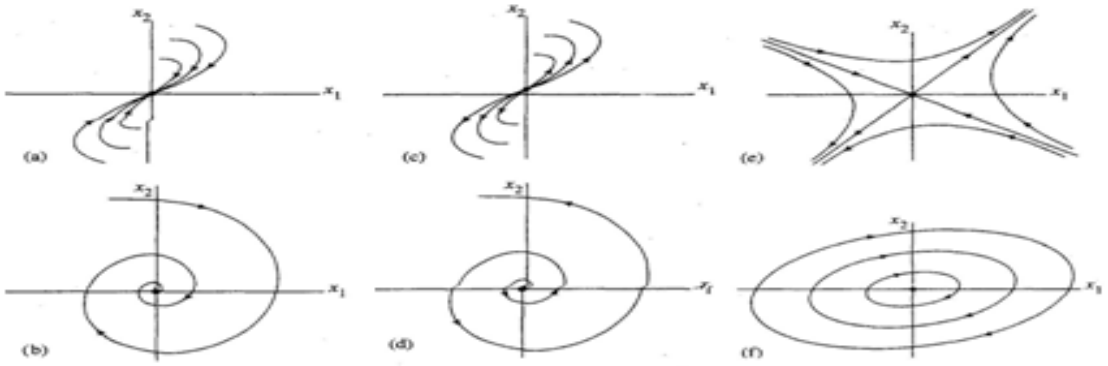

Figure 8 - Different Equilibrium Stats in a Second Order Dynamic System [86]

With respect to the definition of the state space of a dynamic systems, the dynamic system could be in varies states. For example Figure 8 shows different Equilibrium states (situations) that could happen in second order dynamic system[86].

(a) Stable node.

(b) Stable focus.

(c) Unstable node.

(d) Unstable focus.

(e) Saddle point.

(f) Center.

But generally speaking, Figure 9 shows some of the possible events in any dynamic system in motion. Although the Hopfield Neural Networks are dynamic systems, Figure 9 is valid for them.

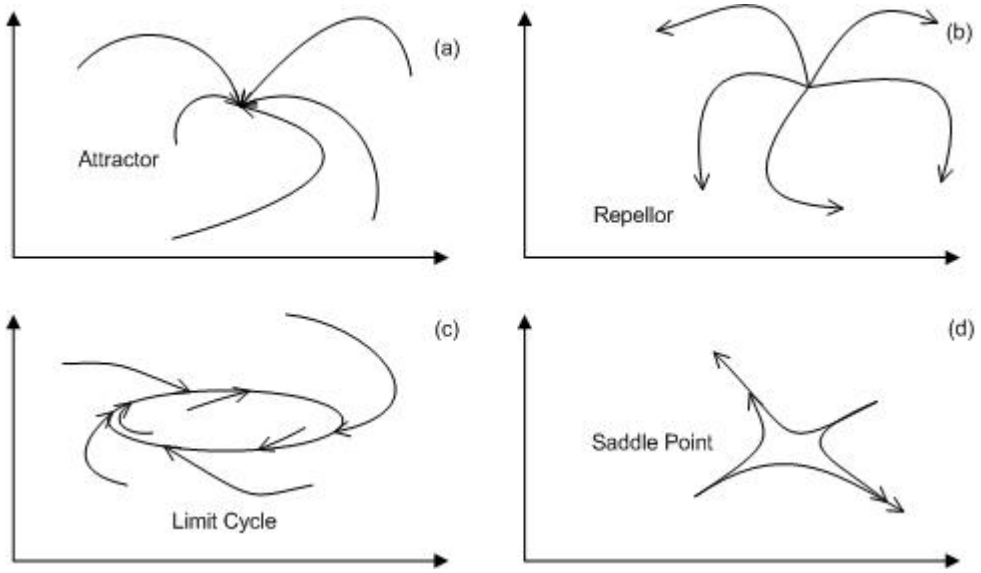

Figure 9 - General example of some events in any dynamic system [96]

In Figure 9 we have:

(a) Attractor states: These states have lower energy compare to their adjacent states and when the dynamic system is placed in these states the system becomes stable.

(b) Repellor states: These states have higher energy compare to their adjacent states. when the dynamic system is placed in these states it becomes unstable.

(c) Limit cycle state: By placing the dynamic systems in one of these types of states, it starts to travel among limited number of states which form a cycle. All the states which are member of a cycle have same energy level.

(d) Saddle states: These states have lower energy level compared to some of their adjacent states and have higher energy level compared to other adjacent states. When the dynamic system is placed in these states, it becomes unstable.

The definition of the state stability, is directly related to the degree of accuracy of our calculation. For example, if a state is stable with an accuracy of n digits after the decimal point, the very same state might not be stable with an accuracy of $m > n$ digits after decimal point. While digits with higher order of significance have converged, it is possible that other digits with lower order of significance exist t which have not converged yet and may converge in the following iterations. Therefore, any stable neuron in a HNN depending on the accuracy of the calculation might be unstable. This holds true for both cases of stability where there is, (i) one stable point, or (ii) a limited number of stable points [1].

Table 6 shows two possible events that could happen in a Hopfield Neural Network. The first scenario shows that by calculating 4 digits after decimal point, after 3 iterations the network converges to 0.5105, but if we increase the accuracy of our calculations to 8 digits after decimal point (first column + second column) it can be clearly seen that the network does not converge in iterations 3, 4 and 5 (it shows different values in all of these iterations).

Table 6-Example of two possible events in Hopfield Neural Networks

Scenario 1			Scenario 2		
	First 4 digits after decimal point	Second 4 digits after decimal point		First 4 digits after decimal point	Second 4 digits after decimal point
Iteration 1	0.5432	9865	Iteration 1	0.3279	5468
Iteration 2	0.5374	2101	Iteration 2	0.3204	2513
Iteration 3	0.5105	7635	Iteration 3	0.3203	6824
Iteration 4	0.5105	6463	Iteration 4	0.3204	2411
Iteration 5	0.5105	6432	Iteration 5	0.3203	6812

The second scenario in Table 6 shows the condition that network converges by repeating some specific states. Considering 4 digits after decimal point, the network repeats 0.3204 and 0.3203

(iteration 2, 4 and iteration 3, 5 are equal). But if we consider 8 digit after decimal then iterations 2, 4 (and iterations 3, 5) are not equal which means the network has not converged yet.

So for increasing the chance of avoiding convergence in our networks we increase the accuracy of our calculation to 15 digits after decimal point. 15 digits after the decimal point is the maximum accuracy that is possible by Matlab 2009a. By using symbolic toolbox accuracy, could be increased, but the speed of calculation is dramatically decreased. As fact that HNN PRNGs rely heavily on mathematical computation it is not feasible time vise to use symbolic processing toolbox.

3.3.2 Our Bit Samplings Mechanism

We have devised a sampling mechanism for the output of the HNN whereby random numbers can be acquired from. To this end we remove the 15^{th} digit after decimal point in the output of each neuron, and extracting remaining digits from the end. If the extracted digit is between 0 to 7 we convert it to a 3-bit binary format and if it is 8 or 9 we discard it. We claim that the collection of such 3-bit binary numbers is used as a Random Sequence of bits [1].

3.4 An Introduction to Our Proposed Fuzzy Hopfield Neural Networks as PRNG

HNN PRNG that is presented in [1][22][23] relies on HNN weight matrix that works as a seed for PRNG. Users by selecting different seeds (weight matrix) produce different random bit sequences. Due to the fact that weight matrix is the combination of 100*100 numbers, it is not easy for users to select 10000 different numbers to generate a random sequence of bits. On the other hand, the size of the weight matrix is tightened with number of neurons in HNN. So for decreasing the size of the weight matrix, we should decrease the number of neurons. However, by

decreasing the number of neurons, the HNN easily converges and it is no more suitable to work as PRNG [22] [23].

We present a specific type of Fuzzy HNN that would be used as PRNG. In effect, we have combined fuzzy logic approach and HNN to get a Fuzzy Hopfield Neural Networks (FHNN) which work as a PRNG. This system has a small weight matrix so that a user could easily use it as PRNG.

3.5 An Introduction to Fuzzy systems and Fuzzy Hopfiled Neural Networks

Fuzzy set A defined on the universe of discourse U is fully characterized by its membership function $\forall x \in U,\ \mu_F(x) \in [0,1]$, that assigns a number from unit interval [0,1] to each element of the universe of discourse. $\mu_F(x)$ represents the degree of membership of x in the fuzzy set A. The 1 extreme represents full membership while the 0 extreme represent full exclusion [25].

Most of the researches in fuzzy logic deal with the problem of mapping from fuzzy set to a fuzzy set. On the other hand whenever we want to apply Fuzzy Logic to engineering problems, we have to deal with the problem of mapping number to number. Most of the desired systems for engineering problems need to accept real numbers as an input and produce real numbers as an output, such as different controlling systems. Therefore fuzzifier used in beginning of fuzzy logic and defuzzifier is placed at end of fuzzy logic, the result denoted as Fuzzy Logic System. In general, Fuzzy Logic Systems prepare nonlinear mapping from vector of input data to vector of output data. Moreover, handling simultaneously numerical data and linguistic variable is the biggest characteristic of fuzzy logic systems [97].

The fuzzy logic theory provides a framework that enables efficient approach to the analysis and synthesis problems with complex nonlinear nature. By combining fuzzy logic systems and neural

networks different types of fuzzy Neural Networks are developed. Recently, convergence problem is studied in different types of fuzzy neural networks [98] [99] [100] [101].

3.6 Our Proposed Fuzzy Hopfield Neural Network (FHNN) as PRNG

We have developed a FHNN to be used as PRNG. The aim of the design is to reduce the number of neurons, therefore the number of weights to the system that would be used as seeds decrease [2].

In the following we have mentioned the specification of our design:

- We have used few numbers of neurons (5) in our system. Increasing the number of neuron means increasing the number of input of each neuron and its fuzzy Logic systems (fuzzy Logic systems will be explained in the next section). Increasing the number of input increases the complexity of Fuzzy Logic systems dramatically and makes the process time consuming, and this is not feasible.
- The structure of the proposed FHNN is fully connected, it should be noted that, self-feedback exists in our structure. Figure 10 depicts the structure of our proposed FHNN presented.
- All of the neurons in our system are exactly the same and there is no priority between them. All of them have the same activation function with the same number of input and output. For activation function, we have used the nonlinear function, *tanh(x)* where *x* is output of fuzzy system that is embedded in each neuron.

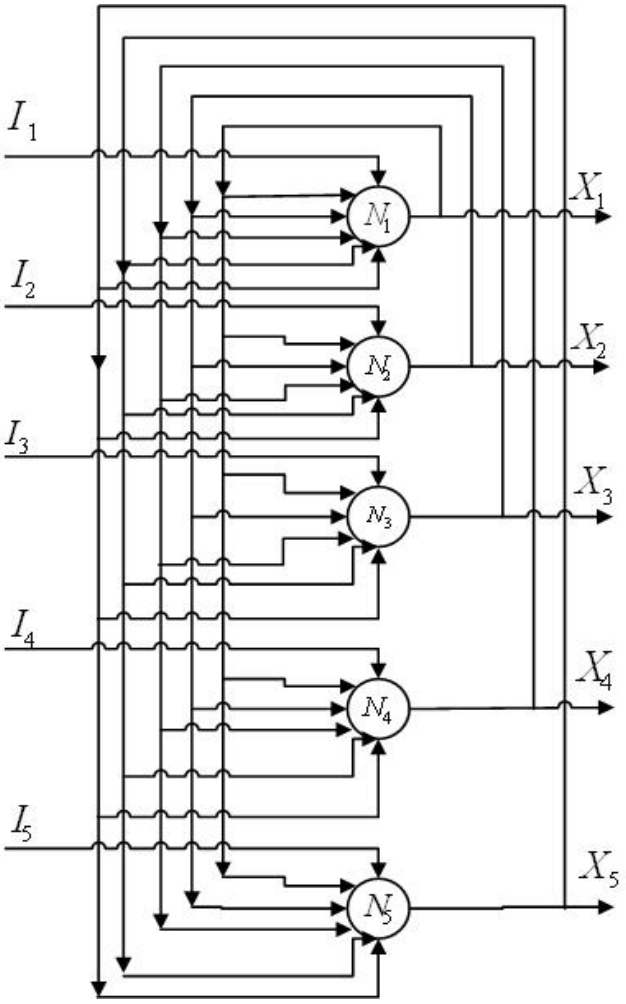

Figure 10 - Fuzzy Hopfield Neural Networks Structure (FHNN)

- Each neuron has a fuzzy system in it, which perform some specific process on its inputs. The final output of neuron would be calculated by applying *tanh* function on the output of the fuzzy system.
- Weight matrix of our FHNN is asymmetric. It contains only positive numbers. Moreover, the diagonal elements of the matrix are all set to +1. In our FHNN θ is zero and in the first iteration all inputs are 1.

3.6.1 Fuzzy Logic System embedded each neurons of our FHNN

Each neuron of our FHNN system works as a standalone Fuzzy system. All inputs of each neuron are outputs of other neurons that are multiplied by their corresponding weight. Each input of a neuron is fuzzified using membership functions depicted in the Figure 11 which are also used as the (de)fuzzification membership functions. We have 7 different membership functions in our system. In fuzzifying process with the input x, we consider membership function i in which $\mu_{mfi}(x) \neq 0$. For example if we have real input of 0.1 in our system, our input will be fuzzified to mf1 with $\mu_{mf1}^{(0.1)} = 0.25$ and mf2 with $\mu_{mf2}^{(0.1)} = 0.65$.

Defuzzifying process uses the same membership functions. So the output of each neurons is set to be [0, 1]. The output of each neuron will be feedback to itself. Hence, each neuron would be aware of what its previous output has been, so in the next iteration it will try to avoid generating the previous output.

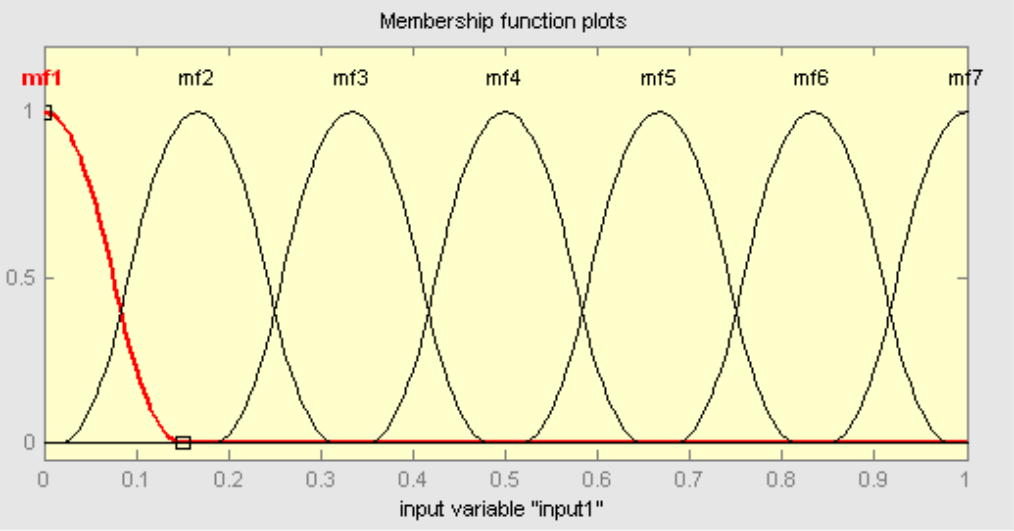

Figure 11- (De)Fuzzification Membership Function

In Fuzzy inference system the rules should be designed in a way that they cover all the possibilities. To do so, we have 5 inputs and one output, and each could acquire up to 3 of the 7 different membership functions. To cover all of the possibilities we need 7^5= 16807 rules, which is very large which makes our system very slow. To compensate for it, we have divided our input into two categories, the inputs from other neurons (4 inputs) and the self feedback of neuron (1 input), we have designed our rule in a way that each rule contains two antecedent and one consequence. First antecedent is the neuron self feedback and the second is from the other neurons output. Hence our Fuzzy inference system in each neuron contains 1*4*7*7=196 rules. As it can be seen in Table 7, rules are defined in a way that the output of each rule is different from its input and the feedback. In Table 7, each value in each cell presents the membership function that is activated, first column is the rule number and the following 5 columns are 5 inputs of each neurons, the first of these 5 inputs belong to the feedback input. The 7^{th} column shows the output value. In our fuzzy system we have implemented conjunction in the antecedent of rules with min t-norm. Implication is implemented and we used product and algebraic sum function [25] is used for aggregating outputs of the rules.

Table 7- FHNN Fuzzy Inference Rules

Rule Number	1st input	2nd input	3rd input	4th input	5th input	Output	Rule Number	1st input	2nd input	3rd input	4th input	5th input	Output	Rule Number	1st input	2nd input	3rd input	4th input	5th input	Output	Rule Number	1st input	2nd input	3rd input	4th input	5th input	Output
1	1	1	0	0	0	5	52	1	0	3	0	0	5	103	1	0	0	5	0	7	154	1	0	0	0	7	3
2	1	2	0	0	0	7	53	1	0	4	0	0	6	104	1	0	0	6	0	6	155	2	0	0	0	1	4
3	1	3	0	0	0	5	54	1	0	5	0	0	7	105	1	0	0	7	0	3	156	2	0	0	0	2	7
4	1	4	0	0	0	6	55	1	0	6	0	0	6	106	2	0	0	1	0	4	157	2	0	0	0	3	5
5	1	5	0	0	0	7	56	1	0	7	0	0	3	107	2	0	0	2	0	7	158	2	0	0	0	4	6
6	1	6	0	0	0	6	57	2	0	1	0	0	4	108	2	0	0	3	0	5	159	2	0	0	0	5	7
7	1	7	0	0	0	3	58	2	0	2	0	0	7	109	2	0	0	4	0	6	160	2	0	0	0	6	4
8	2	1	0	0	0	4	59	2	0	3	0	0	5	110	2	0	0	5	0	7	161	2	0	0	0	7	5
9	2	2	0	0	0	7	60	2	0	4	0	0	6	111	2	0	0	6	0	4	162	3	0	0	0	1	6
10	2	3	0	0	0	5	61	2	0	5	0	0	7	112	2	0	0	7	0	5	163	3	0	0	0	2	7
11	2	4	0	0	0	6	62	2	0	6	0	0	4	113	3	0	0	1	0	6	164	3	0	0	0	3	5
12	2	5	0	0	0	7	63	2	0	7	0	0	5	114	3	0	0	2	0	7	165	3	0	0	0	4	6
13	2	6	0	0	0	4	64	3	0	1	0	0	6	115	3	0	0	3	0	5	166	3	0	0	0	5	7
14	2	7	0	0	0	5	65	3	0	2	0	0	7	116	3	0	0	4	0	6	167	3	0	0	0	6	1
15	3	1	0	0	0	6	66	3	0	3	0	0	5	117	3	0	0	5	0	7	168	3	0	0	0	7	5
16	3	2	0	0	0	7	67	3	0	4	0	0	6	118	3	0	0	6	0	1	169	4	0	0	0	1	7
17	3	3	0	0	0	5	68	3	0	5	0	0	7	119	3	0	0	7	0	5	170	4	0	0	0	2	6
18	3	4	0	0	0	6	69	3	0	6	0	0	1	120	4	0	0	1	0	7	171	4	0	0	0	3	6
19	3	5	0	0	0	7	70	3	0	7	0	0	5	121	4	0	0	2	0	6	172	4	0	0	0	4	2
20	3	6	0	0	0	1	71	4	0	1	0	0	7	122	4	0	0	3	0	6	173	4	0	0	0	5	1
21	3	7	0	0	0	5	72	4	0	2	0	0	6	123	4	0	0	4	0	2	174	4	0	0	0	6	2
22	4	1	0	0	0	7	73	4	0	3	0	0	6	124	4	0	0	5	0	1	175	4	0	0	0	7	1
23	4	2	0	0	0	6	74	4	0	4	0	0	2	125	4	0	0	6	0	2	176	5	0	0	0	1	3
24	4	3	0	0	0	6	75	4	0	5	0	0	1	126	4	0	0	7	0	1	177	5	0	0	0	2	7
25	4	4	0	0	0	2	76	4	0	6	0	0	2	127	5	0	0	1	0	3	178	5	0	0	0	3	1
26	4	5	0	0	0	1	77	4	0	7	0	0	1	128	5	0	0	2	0	7	179	5	0	0	0	4	2
27	4	6	0	0	0	2	78	5	0	1	0	0	3	129	5	0	0	3	0	1	180	5	0	0	0	5	7
28	4	7	0	0	0	1	79	5	0	2	0	0	7	130	5	0	0	4	0	2	181	5	0	0	0	6	3
29	5	1	0	0	0	3	80	5	0	3	0	0	1	131	5	0	0	5	0	7	182	5	0	0	0	7	1
30	5	2	0	0	0	7	81	5	0	4	0	0	2	132	5	0	0	6	0	3	183	6	0	0	0	1	3
31	5	3	0	0	0	1	82	5	0	5	0	0	7	133	5	0	0	7	0	1	184	6	0	0	0	2	4
32	5	4	0	0	0	2	83	5	0	6	0	0	3	134	6	0	0	1	0	3	185	6	0	0	0	3	1
33	5	5	0	0	0	7	84	5	0	7	0	0	1	135	6	0	0	2	0	4	186	6	0	0	0	4	1

Rule Number	1st input	2nd input	3rd input	4th input	5th input	Output	Rule Number	1st input	2nd input	3rd input	4th input	5th input	Output	Rule Number	1st input	2nd input	3rd input	4th input	5th input	Output	Rule Number	1st input	2nd input	3rd input	4th input	5th input	Output
34	5	6	0	0	0	3	85	6	0	1	0	0	3	136	6	0	0	3	0	1	187	6	0	0	0	5	2
35	5	7	0	0	0	1	86	6	0	2	0	0	4	137	6	0	0	4	0	1	188	6	0	0	0	6	2
36	6	1	0	0	0	3	87	6	0	3	0	0	1	138	6	0	0	5	0	2	189	6	0	0	0	7	3
37	6	2	0	0	0	4	88	6	0	4	0	0	1	139	6	0	0	6	0	2	190	7	0	0	0	1	4
38	6	3	0	0	0	1	89	6	0	5	0	0	2	140	6	0	0	7	0	3	191	7	0	0	0	2	5
39	6	4	0	0	0	1	90	6	0	6	0	0	2	141	7	0	0	1	0	4	192	7	0	0	0	3	5
40	6	5	0	0	0	2	91	6	0	7	0	0	3	142	7	0	0	2	0	5	193	7	0	0	0	4	1
41	6	6	0	0	0	2	92	7	0	1	0	0	4	143	7	0	0	3	0	5	194	7	0	0	0	5	1
42	6	7	0	0	0	3	93	7	0	2	0	0	5	144	7	0	0	4	0	1	195	7	0	0	0	6	3
43	7	1	0	0	0	4	94	7	0	3	0	0	5	145	7	0	0	5	0	1	196	7	0	0	0	7	4
44	7	2	0	0	0	5	95	7	0	4	0	0	1	146	7	0	0	6	0	3							
45	7	3	0	0	0	5	96	7	0	5	0	0	1	147	7	0	0	7	0	4							
46	7	4	0	0	0	1	97	7	0	6	0	0	3	148	1	0	0	0	1	5							
47	7	5	0	0	0	1	98	7	0	7	0	0	4	149	1	0	0	0	2	7							
48	7	6	0	0	0	3	99	1	0	0	1	0	5	150	1	0	0	0	3	5							
49	7	7	0	0	0	4	100	1	0	0	2	0	7	151	1	0	0	0	4	6							
50	1	0	1	0	0	5	101	1	0	0	3	0	5	152	1	0	0	0	5	7							
51	1	0	2	0	0	7	102	1	0	0	4	0	6	153	1	0	0	0	6	6							

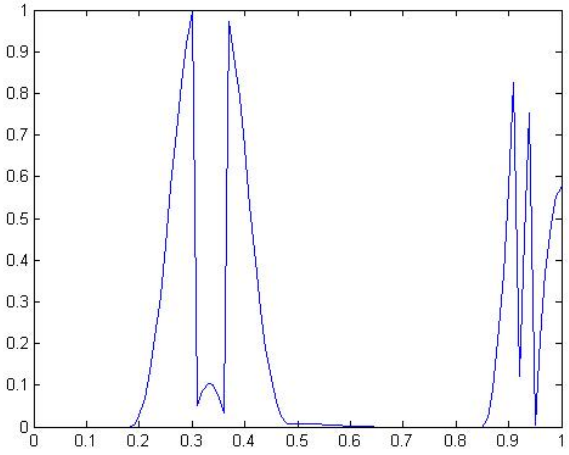

Figure 12 - Example of Aggregation Phase Result

As we mentioned previously, each rule in our fuzzy inference system contains 2 inputs which one of them is previous output of the system. The output of each rule is set in a way that it avoids previous output. So after aggregation phase (summation of the output of all rules) we have values that spread in all of our domain, except the part that presents previous output. Figure 12 and Figure 13 are two examples of our values after aggregation phase.

For defuzzification process, we used Centroid (3-5),

$$y_c(x) = \frac{\sum_{n=1}^{\infty} y_i \ \mu_\beta(y_i)}{\sum_{n=1}^{\infty} \mu_\beta(y_i)} \tag{3-5}$$

which is calculated only for bigger region in aggregation result. For example in Figure 12 two regions, one [0.2-0.6] and the other one [0.85-1] would be recognized, and the defuzzification process considers the first which is [0.2 -0.6]. Whilst in Figure 13 shows only one region exists and the defuzzification process will be applied on [0.35-1].

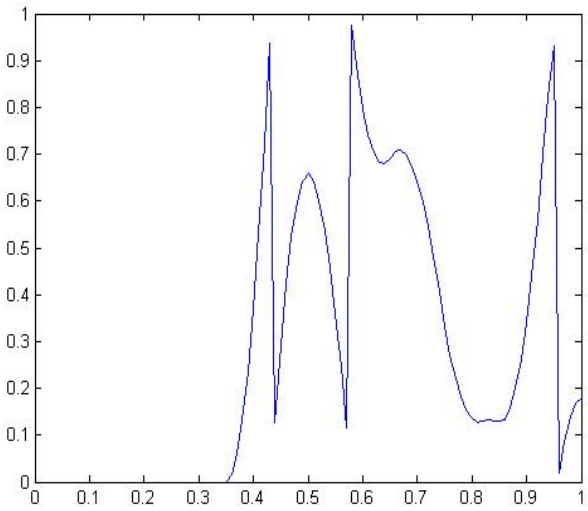

Figure 13- Example of Aggregation Phase Result

3.6.2 Reseeding mechanism for FHNN

By applying *tanh(x)* on output of the fuzzy inference system we acquire the output of each neuron. By considering 15 digit precision in our calculation we improve the accuracy of our system.

After each 5 iterations we change the weight matrix of our FHNN. We reverse the system outputs in 5 previous iterations and use them as a new weight matrix for our system. But the diagonal values of weight matrix will remain unchanged which is 1.

The main feature of this system is that by changing the weight matrix in each 5 iteration the chance of convergence decreases, because after each five iteration we move the state of our FHNN to a new point in the state space, so our FHNN should try to move toward a (new) stable point. In other words, our FHNN has only 5 iterations to converge and it is very hard. More over the fuzzy logic approach that is used in our FHNN makes our system, not to converge even with few numbers of iterations, this is due to the fact that our fuzzy logic approach does not allow a neuron to repeat its output [2].

A foolish builder builds a house on sand. Test and verify the randomness of your random number generator.
~ Casimir Klimasauskas

Chapter 4

Experiments

4.1 Testing HNN PRNG by NIST Test Suite

We implemented a PRNG based HNN with 100 neurons with HNN weights playing the role of PRNG seed. To reduce the possibility of convergence, the $100 * 100$ weights matrix of HNN was filled as is explained in chapter 3, section 3-3. Furthermore as discussed in chapter 3, by considering 15 digit precision we both increased the accuracy of our calculation and as well decreased the chance of convergence of the network. We also devised a sampling mechanism whereby random numbers would be acquired from the outputs of network. This was done by removing the 15^{th} digit of the output of each neuron, and extracting the rest of the digits from the end. If the extracted digit is between 0 to 7 then we convert it to a 3-bit binary format and if it is 8 or 9 we discard it. A collection of such 3-bit binary numbers is used as a Random Sequence of bits.

In our test module we set the α parameter to 0.01 (i.e. the level of confidence of test is set to 99%) and the sequence length to 1,000,000. Based on NIST PRNG Test Module the minimum proportion pass rate for each statistical test is 0.96015. We conduct 1-digit selection test for 3 times and the results are tabulated in Table 8.

Investigating the columns of Table 8, reveals that for most of the tests e.g. Frequency and Approximate Entropy tests, P-Values do not show good results. Moreover with respect to proportion results, the HNN PRNG could not pass most of the tests e.g. Cumulative Sums-

Forward/Reverse, Runs, Rank and Longest Run tests according to the proportion results of those tests.

Table 8- Results of 1-digit selection HNN

PRNG Test	**1-digit selection Experiment 1**		**1-digit selection Experiment 2**		**1-digit selection Experiment 3**	
	P-Value	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*
Frequency	0.0589	0.91*	0.0066	0.88*	0.0126	0.90*
Block Frequency	0.2896	0.97	0.0109	0.95*	0.7597	0.98
Cumulative Sums-Forward	0.1453	0.91*	0.0004	0.88*	0.0428	0.89*
Cumulative Sums-Reverse	0.3504	0.93*	0.0029	0.89*	0.0155	0.88*
Runs	0.0269	0.91*	0.0909	0.92*	0.0308	0.88*
Longest Run	0.4559	0.89*	0.3345	0.91*	0.0909	0.89*
Rank	0.4372	0.95*	0.4559	0.97	0.0000	0.93*
Non Overlapping Template	0.3102	0.92*	0.2485	0.92*	0.1902	0.90*
Overlapping Template	0.0487	0.89*	0.0251	0.92*	0.0308	0.88*
Universal	0.0456	0.89*	0.0757	0.91*	0.1025	0.90*
Approximate Entropy	0.1223	0.90*	0.0004	0.85*	0.0022	0.87*
Random Excursions	0.3295	0.99	0.2643	0.99	0.6490	0.98
Random Excursions Variant	0.3413	0.99	0.4265	0.99	0.5165	0.98
Serial(m=5)	0.0647	0.06*	0.1504	0.91*	0.0057	0.87*
Linear Complexity	0.7399	0.94*	0.2368	0.94*	0.2896	0.92*

(*) indicates that our HNN PRNG has not passed its corresponding test
(*P-Value)* is the result of the test
(*Prop)* is proportion of the results for 100 sequences

The obtained results imply that there might be some kind of patterns among the digits in the output of neurons in some iterations, i.e. by selecting one digit from the output of each neuron in the HNN PRNG, the final bit Sequence does not show good statistical characteristic to pass the PRNG Test. The question that we will investigate is, "What is the influence of increasing the number of digits to be selected from output of each neuron on the PRNG test results?"

4.2 Effect of Digit Sampling Mechanism on the HNN as PRNG

In 1-digit selection test, we noticed that in most of the tests, our HNN PRNG did not provide good *P-Values* and *Proportions*.

In subsequent experiments, we increased the number of extracted digits from the output of each neuron. and we tested HNN PRNG with 3,5,7 and 9 digits extracted from the output of our HNN neurons. In each case, we have conducted the experiment 3 times. We could then examine the influence of the number of extracted digits from the output of each neuron in PRNG test results.

The results of experiments with 3,5,7 and 9 digit selection are respectively shown in Table 9-12.

Table 9- Results of 3 digit selection HNN

PRNG Test	**3-digit selection Experiment 1**		**3-digit selection Experiment 2**		**3-digit selection Experiment 3**	
	P-Value	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*
Frequency	0.6371	0.96	0.5749	0.95*	0.4190	0.94*
Block Frequency	0.3838	0.96	0.2368	0.99	0.9114	0.96
Cumulative Sums-Forward	0.9114	0.95*	0.5749	0.95*	0.3838	0.94*
Cumulative Sums-Reverse	0.0006	0.95*	0.6993	0.94*	0.0219	0.94*
Runs	0.1372	0.97	0.4749	0.97	0.6786	0.94*
Longest Run	0.2022	0.94*	0.5749	0.96	0.9942	0.96
Rank	0.6786	0.99	0.2896	1	0.4943	0.98
Non Overlapping Template	0.4225	0.96	0.4199	0.95*	0.3669	0.94*
Overlapping Template	0.4190	0.97	0.0456	0.95*	0.8343	0.94*
Universal	0.8343	0.97	0.6163	0.97	0.1223	0.95*
Approximate Entropy	0.4559	0.93*	0.0519	0.95*	0.0057	0.92*
Random Excursions	0.2899	0.98	0.2669	0.99	0.5122	0.99
Random Excursions Variant	0.3648	0.99	0.3195	0.99	0.4968	0.99
Serial(m=5)	0.3504	0.97	0.4144	0.93*	0.2985	0.94*
Linear Complexity	0.4943	0.97	0.8977	1	0.4559	0.99

(*) indicates that our HNN PRNG has not passed its corresponding test
(*P-Value)* is the result of the test
(*Prop)* is proportion of the results for 100 sequences

Table 10- Results of 5 digit selection HNN

PRNG Test	**5-digit selection Experiment 1**		**5-digit selection Experiment 2**		**5-digit selection Experiment 3**	
	P-Value	*Prop*	*P-Value*	*P-Value*	*Prop*	*P-Value*
Frequency	0.2757	0.95*	0.0308	0.2757	0.95*	0.0308
Block Frequency	0.5544	0.93*	0.6993	0.5544	0.93*	0.6993
Cumulative Sums-Forward	0.3669	0.96	0.0308	0.3669	0.96	0.0308
Cumulative Sums-Reverse	0.0167	0.94*	0.1372	0.0167	0.94*	0.1372
Runs	0.1025	0.91*	0.8513	0.1025	0.91*	0.8513
Longest Run	0.1916	0.96	0.7791	0.1916	0.96	0.7791
Rank	0.9463	0.97	0.9357	0.9463	0.97	0.9357
Non Overlapping Template	0.1913	0.92*	0.4029	0.1913	0.92*	0.4029
Overlapping Template	0.3669	0.92*	0.1916	0.3669	0.92*	0.1916
Universal	0.3669	0.97	0.5341	0.3669	0.97	0.5341
Approximate Entropy	0.0219	0.87*	0.9114	0.0219	0.87*	0.9114
Random Excursions	0.6230	0.98	0.2211	0.6230	0.98	0.2211
Random Excursions Variant	0.4189	0.99	0.2045	0.4189	0.99	0.2045
Serial(m=5)	0.0024	0.88*	0.4206	0.0024	0.88*	0.4206
Linear Complexity	0.2133	0.99	0.0219	0.2133	0.99	0.0219

Table 11- Results of 7 digit selection HNN

PRNG Test	**7-digit selection Experiment 1**		**7-digit selection Experiment 2**		**7-digit selection Experiment 3**	
	P-Value	*Prop*	*P-Value*	*P-Value*	*Prop*	*P-Value*
Frequency	0.5955	0.97	0.0117	0.5955	0.97	0.0117
Block Frequency	0.4190	0.99	0.7399	0.4190	0.99	0.7399
Cumulative Sums-Forward	0.2622	0.97	0.0401	0.2622	0.97	0.0401
Cumulative Sums-Reverse	0.3041	0.97	0.0109	0.3041	0.97	0.0109
Runs	0.2492	0.98	0.0179	0.2492	0.98	0.0179
Longest Run	0.8343	0.99	0.5955	0.8343	0.99	0.5955
Rank	0.0487	0.99	0.7981	0.0487	0.99	0.7981
Non Overlapping Template	0.5044	0.98	0.5108	0.5044	0.98	0.5108
Overlapping Template	0.5141	0.99	0.4190	0.5141	0.99	0.4190
Universal	0.6371	1	0.2896	0.6371	1	0.2896
Approximate Entropy	0.2896	0.96	0.0965	0.2896	0.96	0.0965
Random Excursions	0.4499	0.99	0.4710	0.4499	0.99	0.4710
Random Excursions Variant	0.4638	0.99	0.5086	0.4638	0.99	0.5086
Serial(m=5)	0.1662	0.95*	0.0537	0.1662	0.95*	0.0537
Linear Complexity	0.3838	1	0.8343	0.3838	1	0.8343

Table 12- Results of 9 digit selection HNN

PRNG Test	**9-digit selection Experiment 1**		**9-digit selection Experiment 2**		**9-digit selection Experiment 3**	
	P-Value	*Prop*	*P-Value*	*Prop*	*P-Value*	*Prop*
Frequency	0.6163	0.97	0.5141	0.95*	0.7399	0.99
Block Frequency	0.3190	0.98	0.2622	1	0.2622	1
Cumulative Sums-Forward	0.0329	0.96	0.9463	0.95*	0.9114	0.99
Cumulative Sums-Reverse	0.6786	0.95*	0.0269	0.96	0.4372	0.99
Runs	0.4943	0.98	0.3838	0.94*	0.2022	0.97
Longest Run	0.5544	1	0.2896	0.98	0.4011	1
Rank	0.5749	1	0.1537	1	0.1153	1
Non Overlapping Template	0.4142	0.96	0.4476	0.96	0.5004	0.98
Overlapping Template	0.5544	0.99	0.2757	0.97	0.9114	0.99
Universal	0.6579	0.98	0.1153	1	0.7197	0.97
Approximate Entropy	0.1372	0.92*	0.3190	0.92*	0.7791	0.97
Random Excursions	0.4682	0.98	0.3344	0.97	0.3656	0.98
Random Excursions Variant	0.2955	0.98	0.3274	0.99	0.3912	0.97
Serial(m=5)	0.4127	0.92*	0.4404	0.955*	0.7357	0.975
Linear Complexity	0.5341	0.99	0.8343	1	0.1453	0.99

(*) indicates that our HNN PRNG has not passed its corresponding test
(*P-Value)* is the result of the test
(*Prop)* is proportion of the results for 100 sequences

In most of the tests, like Frequency Test, Runs Test and Longest Run test, P-values in 3-digit selection in comparison with their corresponding in 1-digit selection are larger -

Table 9. Some tests such as Linear Complexity with 1-digit selection were failed, but passed in 3-digit selection tests. However, 3-digit selection test of HNN PRNG could not pass some tests like Cumulative Sums-Forward and Cumulative Sums-Reverse, and 1-digit selection was unable to pass too.

Experiment 2 of 5-digit selection in Table 10, has passed all the tests except Approximate Entropy and Serial test. It seems we are moving toward a better PRNG, by increasing the number of extracted digits. Be noticed that in experiments 1 and 3 in 5-digit selection test- Table 10 - PRNG could not pass some tests. However, not the same tests failed in the both experiments.

Noticeably, in Table 11, Experiment 2 of 7-digit selection tests shows HNN PRNG has passed all tests with higher P-values and Proportions than previous tests. However in experiments 1 and 3, HNN PRNG failed Serial Test with a close number to the pass/fail threshold of proportion result. Experiment 3 of 9-digit selection test shown in Table 12, has also passed all the tests but in experiments 1 and 2 some tests have failed. *P-Values* in 9-digit selection experiments in comparison with 7-digit Selection columns of Table 11 and Table 12, shows increase. Due to the great chance of convergence, we did not conduct 10- or 11-digit selection tests.

In each experiment of 1-digit selection, 12 tests were failed - Table 8 - while in 9-digit selection tests, in worst case – experiment 2 – 5 tests were failed, and in the best case – experiment 3 - all test were passed. This means that HNN PRNG by 9-digit selection mechanism produces much more randomly numbers. Moreover, p-values in all of the tests in 9-digit selection experiments Table 12 - show better results comparing with the corresponding tests in 1- digit selection experiment - Table 8 – noting that the larger p-value indicates better PRNG, the experiments show that the PRNG by increasing the numbers of digits that are acquired from output of each neuron, PRNG not only passes more tests but also passes them with better results.

4.3 FHNN as PRNG

As we described in chapter 3 section 3.6, we have also implemented a PRNG based on FHNN with 5 neurons in which HNN weight matrix plays the role of PRNG seed. By considering 15 digit precision we have both increased the accuracy of our calculation and decreased the chance of convergence of the network. We also used the sampling mechanism explained previously.

The results are summarized in Table 13,

Table 13- Results of Fuzzy Hopfield NN PRNG

PRNG Test	**FHNN Experiment**	
	P-Value	*Prop*
Frequency	0.2248	0.99
Block Frequency	0.4749	0.99
Cumulative Sums-Forward	0.4559	0.98
Cumulative Sums-Reverse	0.8343	0.99
Runs	0*	0*
Longest Run	0.7597	0.98
Rank	0.6993	1
Non Overlapping Template	0.0721	0.77*
Overlapping Template	0.1153	0.99
Universal	0*	0.71*
Approximate Entropy	0*	0*
Random Excursions	0.2943	0.97
Random Excursions Variant	0.4025	0.99
Serial(m=5)	0*	0*
Linear Complexity	0.2492	1

(*) indicates that our HNN PRNG has not passed its corresponding test
(*P-Value)* is the result of the test
(*Prop)* is proportion of the results for 100 sequences

By analyzing Table 13, we observed that in some of the tests for e.g. Runs and Approximate Entropy tests, P-Values do not indicate good results. Moreover our FHNN PRNG could not pass some of the tests like Non Overlapping Template according to the proportion results of those tests. Runs, Non Overlapping Template, Universal, Approximate Entropy and serial test have low P-Value and also proportion result of these tests are below the acceptance rate of the test.

By comparing our results with the results of 9-digit selection experiment 1 in Table 12- medium case among 3 experiments - we found that in Block Frequency, Cumulative Sums-Forward/Reverse we have slightly better results, but the problem is in Runs, Non Overlapping Template, Universal, Approximate Entropy and serial tests, where the P-value has fallen to zero.

HNN PRNG is a good PRNG system but according to number of weight that user should select as a seed for this system, makes it difficult to work with it. We try to reduce the size of the matrix and came up with FHNN with only 5 neurons. By comparing the run test result between HNN and FHNN it shows that the oscillation rate between zero and one in our FHNN PRNG is much lower than HNN PRNG. This could be a direct consequence of using fewer numbers of neurons in our FHNN system - the less the number of neurons, the less the number of parameters (inputs) for each neuron, that participates in generating the output.

In [1] [22] [23], a large numbers of neurons are used to avoid the convergence problem, however, by applying FHNN we tried to reduce the number of neurons to have a smaller weight matrix and avoiding the convergence problem. We showed that the numbers of neurons do not only related to convergence problem also it has a big influence in number of patterns that appear in the bit sequences.

4.4 Comparing our HNN PRNG and FHNN PRNG with other PRNGs

In this section we will compare our proposed HNN and FHNN PRNGs with other existing PRNGs. To do the test we set the α parameter to 0.01 and the sequence length to 1000000. Linear Congruential, Quadratic Congruential 1 and 2, Cubic Congruential, Micali-Schnorr, Blum Blum Shub, Modular Exponentiation and G using SHA-1 are PRNG algorithms that we have tested them with NIST PRNG Test Package. All of these algorithms except our proposed HNN PRNG and FHNN PRNG are implimented by NIST. These PRNGs are publicly available in the NIST PRNG Test package that anybody can execute them and compare their results with his/her PRNG.

The detailed results of the tests performed on different PRNGs are tabulated in the Appendix, however Table 14 summarize the tests outcome. This clearly shows how good our Proposed PRNG is. Among these PRNGs some of them could not pass some tests of the NIST random test suite. Table 14 presents the tests that each PRNG could not passed. Whenever, there is a star in any cell of Table 14, it indicates that the corresponding algorithm could not passed the related test. We could say as a point of view of that specific test, that PRNG is not random.

Table 14 - Pass/Fail Results of PRNGs tested with NIST test module

NIST Tests / Pseudo Random Number Generator	Frequency	Block Frequency	Cumulative Sums-Forward	Cumulative Sums-Reverse	Runs	Longest Run	Rank	Non Overlapping Template	Overlapping Template	Universal	Approximate Entropy	Random Excursions	Random Excursions Variant	Serial(m=5)	Linear Complexity
Hopfield NN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Fuzzy Hopfield NN	✓	✓	✓	✓	×	✓	✓	×	✓	×	×	✓	✓	×	✓
Linear Congruential	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Quadratic Congruential-1	×	✓	×	×	×	✓	✓	×	✓	✓	✓	×	✓	×	✓
Quadratic Congruential-2	×	✓	×	×	✓	✓	✓	✓	✓	✓	✓	×	×	×	✓
Cubic Congruential	×	×	×	×	×	✓	✓	✓	✓	✓	×	✓	✓	×	✓
Micali Schnorr	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Blum Blum Shub	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Modular Exponentiation	×	✓	×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓
G function using SHA1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

In Figure 14- Figure 28 we have compared the results of all the PRNGs that we named in previous page. To simplify the following figures, we have associated a specific color to each PRNG and the results of their tests are shown using the corresponding color,

- Violet ↔ HNN PRNG with 9 digit selection (experiment 3Table 12)
- Dark Blue ↔ FHNN PRNG
- Blue ↔ Linear Congruential PRNG
- light Blue ↔ Quadratic Congruential 1 PRNG
- Aquamarine ↔ Quadratic Congruential 2 PRNG
- Green ↔ Cubic Congruential PRNG
- Yellow ↔ Micali-Schnorr PRNG
- Orange ↔ Blum Blum Shub PRNG
- Red ↔ Modular-Exponentiation PRNG
- Dark Red ↔ G using SHA-1 PRNG

In all of these figures the vertical axes present P-Value.

4.4.1 Frequency and Block Frequency test result

In Figure 14 the result for Frequency test is presented and surprisingly HNN PRNG has a very good result compared to Blum Blum shub algorithm. It is noticeable that its result is higher than all the other PRNGs. Also the result of the test on FHNN PRNG is comparable with Quadratic Congruential 1 & 2, Cubic Congruential, Micali-Schnorr and Modular-Exponentiation PRNG. It is important to note that the frequency test is a very important test that most of the PRNGs could not pass or pass it with poor results - Micali-Schnorr, is an example for the later.

Figure 15 shown the results of Block Frequency Test. HNN PRNG and FHNN PRNG show acceptable results compare to other PRNGs. FHNN outperform HNN in this test, which is due to

the fact that the output of FHNN in each iteration is less than the block that was to be tested (in test module, the length of each block is 128).

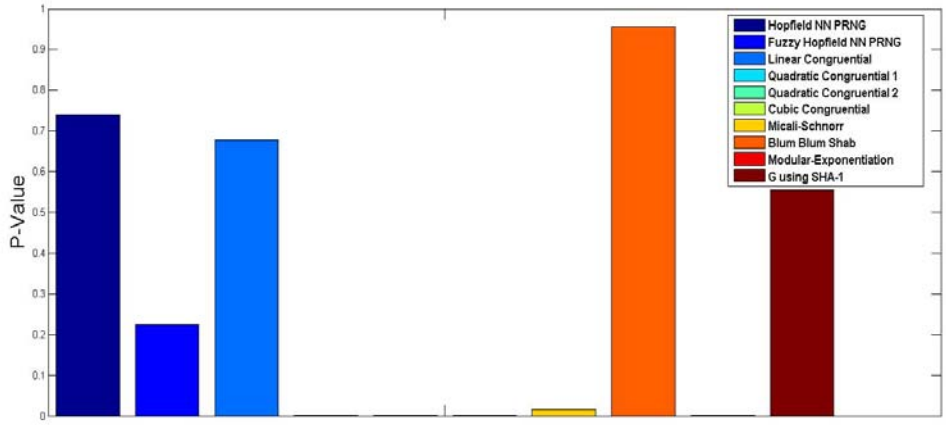

Figure 14 - Frequency test result for different PRNGs

Micali-Schnorr and G using SHA1 show high *P-Values*, in comparison Modular-Exponentiation and Quadratic Congruential 1 show low *P-Values*. Cubic Congruential is the only PRNG that could not pass Block Frequency test.

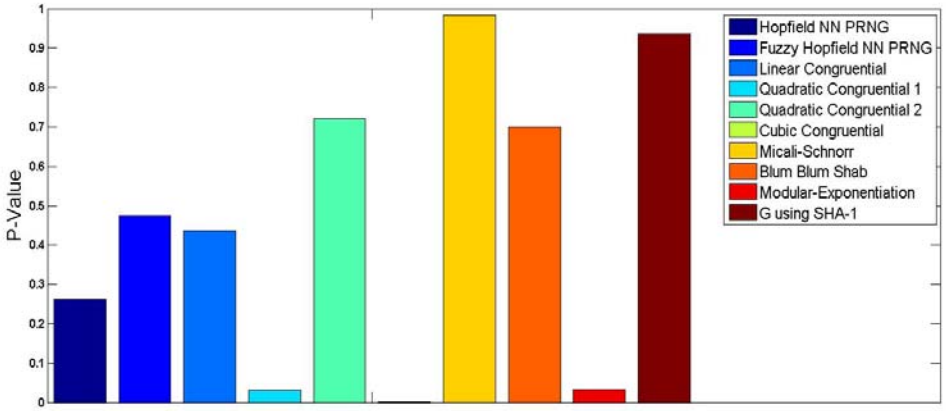

Figure 15 - Block Frequency test result for different PRNGs

4.4.2 Cumulative Sum-Forward/Reverse test result

Figure 16 represents the results of Cumulative Sums-Forward Test that HNN PRNG has the best result comparing to others. It is followed by G using SHA1 PRNG and FHNN PRNG, Micali-Schnorr, Blum Blum Shub has also good results but with considerable difference with HNN PRNG.

In Figure 17 the results of Cumulative Sums-Reverse Test are reported and FHNN has the best result. Moreover, FHNN has a very good result comparable to others. The difference in the results of Micali-Schnorr, Blum Blum Shub and G using SHA-1 between Figure 16 and Figure 17 are considerable.

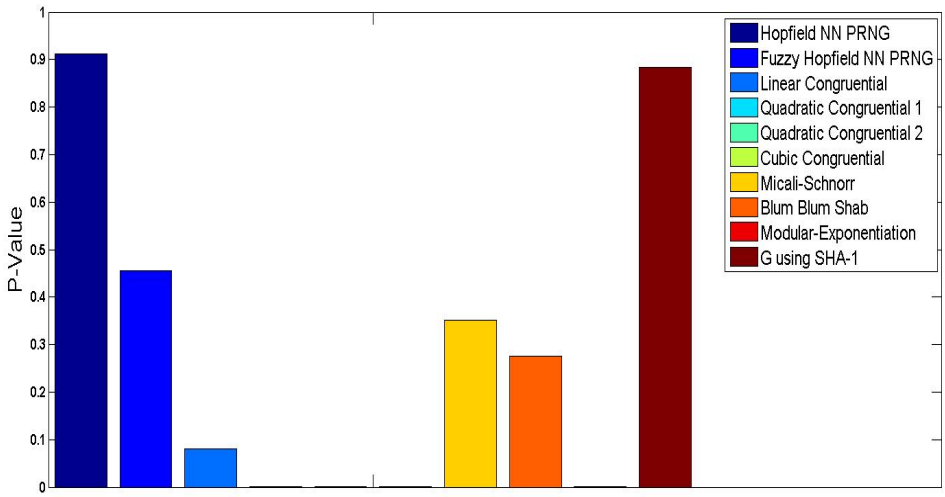

Figure 16 - Cumulative Sum-Forward test result for different PRNGs

Figure 16 and Figure 17 show that Quadratic Congruential 1 and Quadratic Congruential 2, Cubic Congruential and Modular-Exponentiation have poor results in this test.

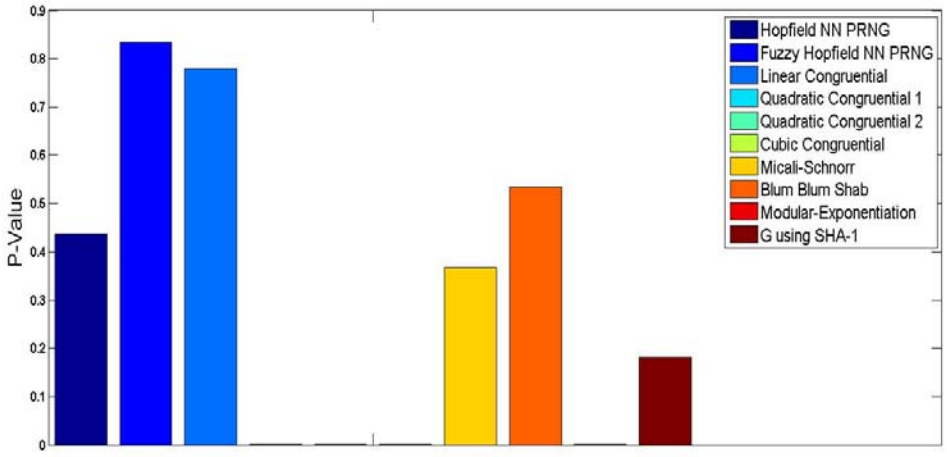

Figure 17 - Cumulative Sum-Reverse test result for different PRNGs

4.4.3 Run and Longest Run test result

Figure 18 represents the results of Run Test. HNN has an acceptable result but FHNN like Cubic Congruential has not passed the test. Most of the PRNGs have satisfactory result in this test. Linear Congruential, Modular Exponentiation and Quadratic Congruential 1 PRNGs are the leaders in this test. Micali-Schnorr presents the lowest *P-Value* among other PRNGs which passed this test.

Figure 19 shows the results for Longest Run test. All the PRNGs have passed this test, and in particular HNN and FHNN have acceptable *P-Values*. Surprisingly, FHNN shows considerable Improvement between Run test and Longest Run test than HNN. Longest Run focuses on Longest Run of the same bit in random number sequence. Figure 18 and Figure 19 clearly showed that result of Run test and Longest Run test on the same sequence of numbers could have totally different result.

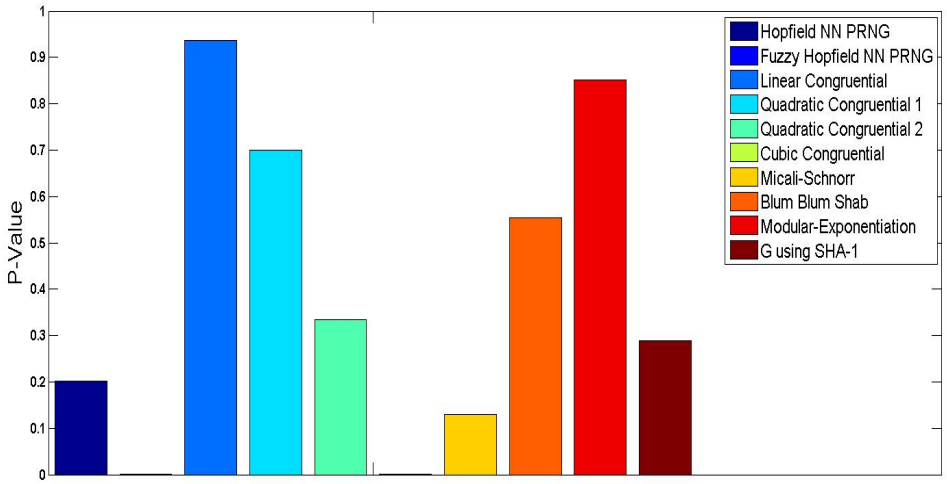

Figure 18- Runs test result for different PRNGs

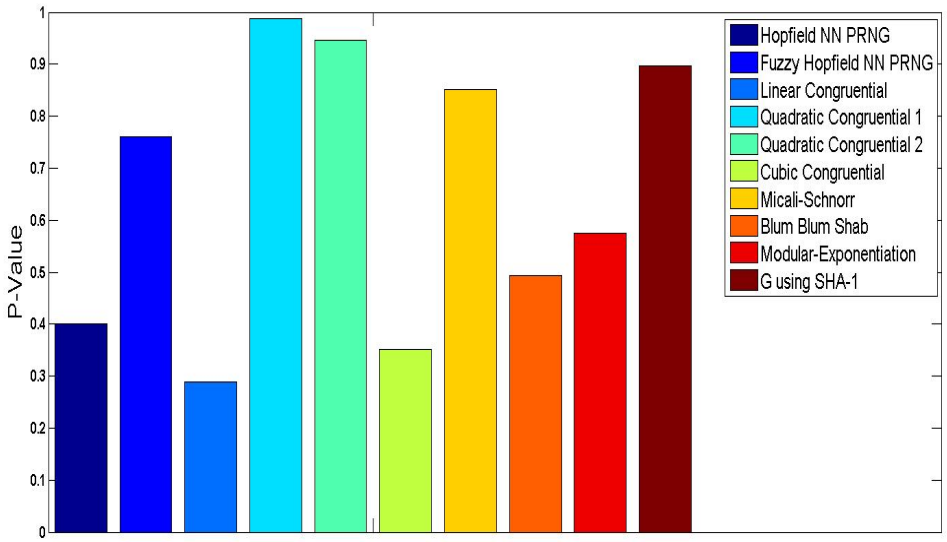

Figure 19 - Longest Run test result for different PRNGs

4.4.4 Rank test result

Rank Test result is shown in Figure 20. Modular Exponentiation has the best result among all of the PRNGs and in particular FHNN and Quadratic Congruential 1 have also got good results. All PRNGs have passed this test, but the results of Micali-Schnorr, G using SHA-1 and HNN PRNGs are not that good.

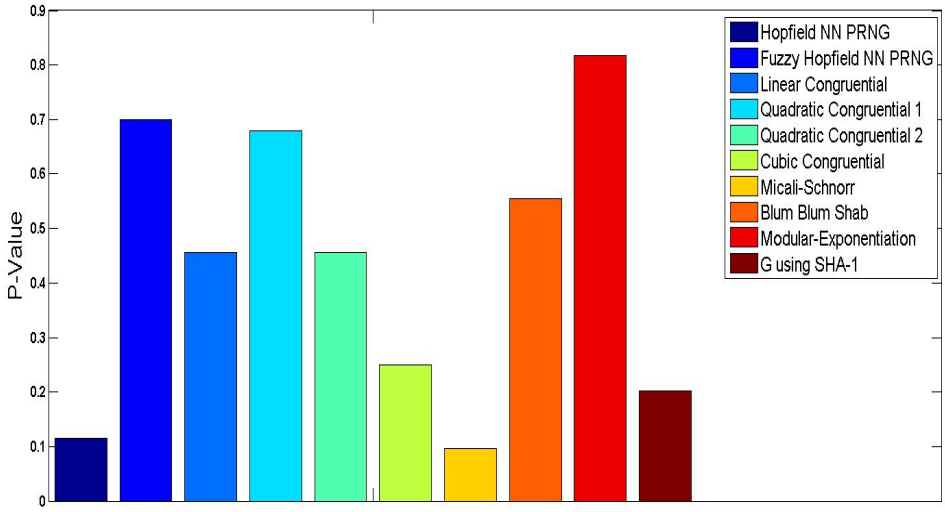

Figure 20- Rank test for different PRNGs

4.4.5 (Non) Overlapping Template test result

(Non) Overlapping Template Test Results are shown in Figure 21 and Figure 22. All of the PRNGs have passed Overlapping Template test. FHNN and Quadriatic Congruential 1 did not pass Non Overlapping Template test (Table 14),whilst others have passed this test.

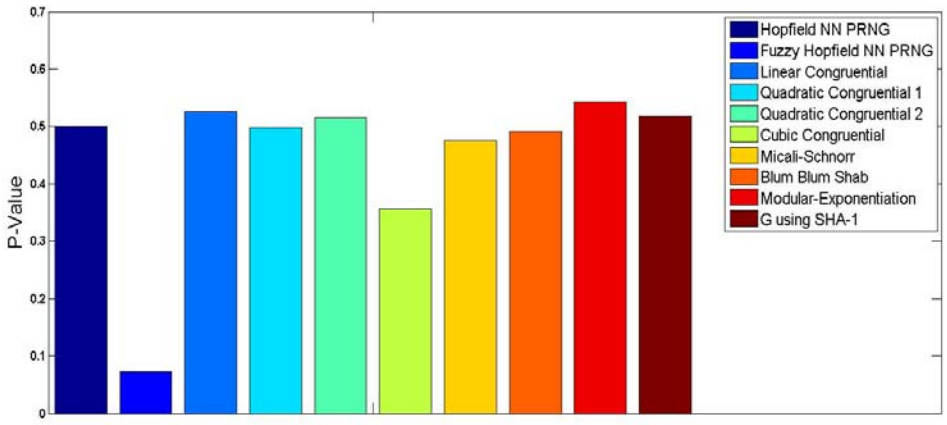

Figure 21 - Non Overlapping Template test result for different PRNGs

The window block that is used in (Non)Overlapping Template test consisted of 9 bits, which means the NIST PRNG test suite looks for the number of occurrences of specific patterns with the length of 9 bits. Non Overlapping Template test result shows that all PRNGs except FHNN got good result. In Overlapping Template test (Figure 22) HNN PRNG has the best result, and noticeably, the difference between HNN and Blum Blum Shub is considerable. FHNN in this test has a better result than Modular Exponentiation and G using SHA-1 PRNGs.

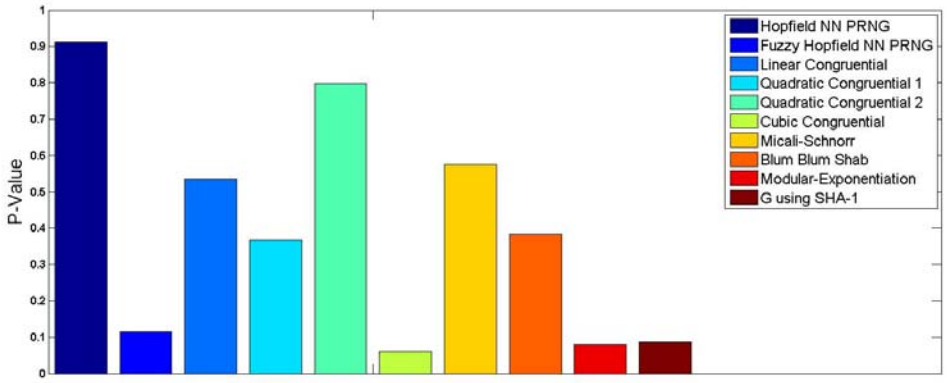

Figure 22 - Overlapping template test result for different PRNGs

4.4.6 Universal test result

Figure 23 depict Universal Test results. FHNN is the only PRNG that has not passed this test. Cubic Congruential, HNN and Blum Blum Shub PRNGs have very good result. Modular Exponentiation, G using SHA-1 and Micali-Schnorr showed low *P-Values*. Universal test is a very important test and acquiring high value in this test is a positive point. HNN has been acquired in this test.

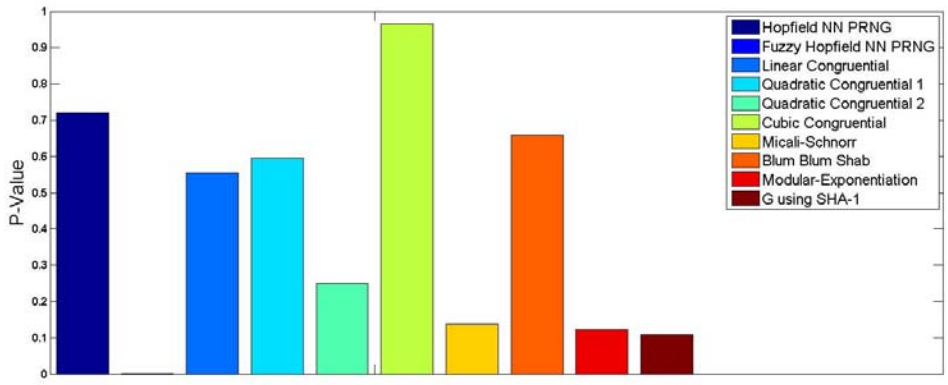

Figure 23 - Universal test result for different PRNGs

4.4.7 Approximate Entropy test result

In Figure 24 results of Approximate Entropy Test show that our HNN PRNG is the best in this test. FHNN and Cubic Congruential PRNGs on the other hand have the worst results and have not passed this test. This test checks the frequency of oscillation between two consecutive overlapping blocks with the length of (M, M+1), here M was set to 10 in NIST PRNG Test suite. It seems that in FHNN the number of oscillations between outputs of FHNN is not suitable, this could direct result of using few numbers of neurons compared to HNN that has best result in this test.

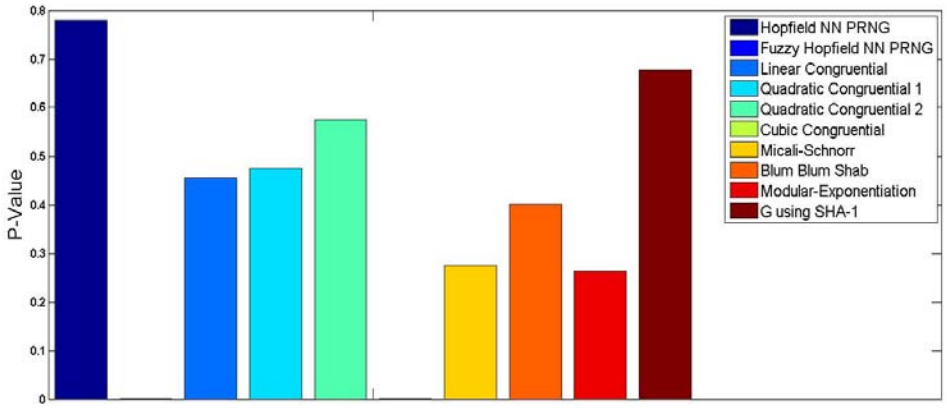

Figure 24 - Approximate Entropy test result for different PRNGs

4.4.8 Random Excursion (Variant) test result

In Figure 25 and Figure 26, Random Excursions and Random Excursion Variant Test Results are presented. All the PRNGs have passed these tests however HNN and FHNN have good *P-Values* comparing to the other PRNGs. Quadratic Congruential 1 & 2 PRNGs, although have acquired good P-values but have not passed this test (Table 14).

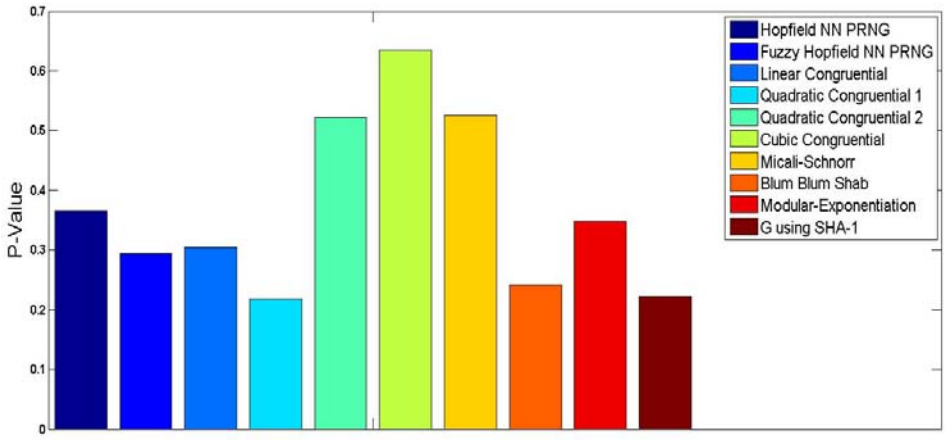

Figure 25 - Random Excursions test result for different PRNGs

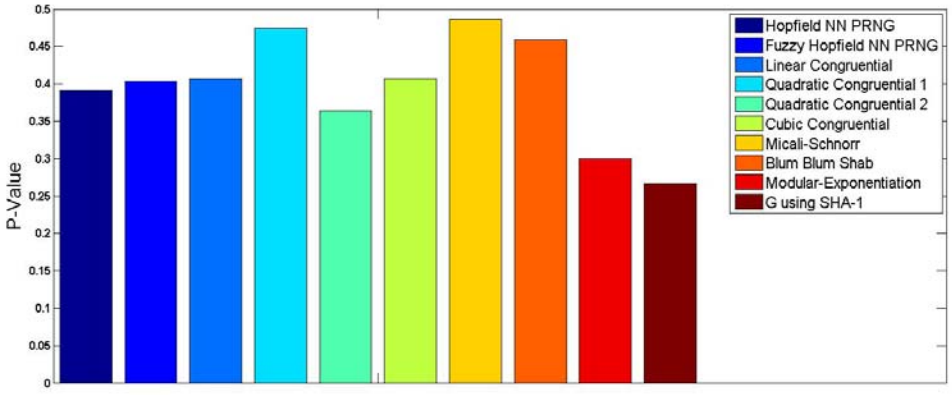

Figure 26 - Random Excursions Variant test result for different PRNGs

4.4.9 Serial test result

Figure 27 presents Serial Test Result. FHNN, Quadratic Congruential 1,2 and Cubic Congruential are the PRNGs, which have not passed this test. On the other hand HNN, Micali-Schnorr, Blum Blum Shub and G using SHA1 PRNGs show high P-values that indicates the strength of these PRNGs. All the PRNGs who passed this test, show acceptable result in serial test.

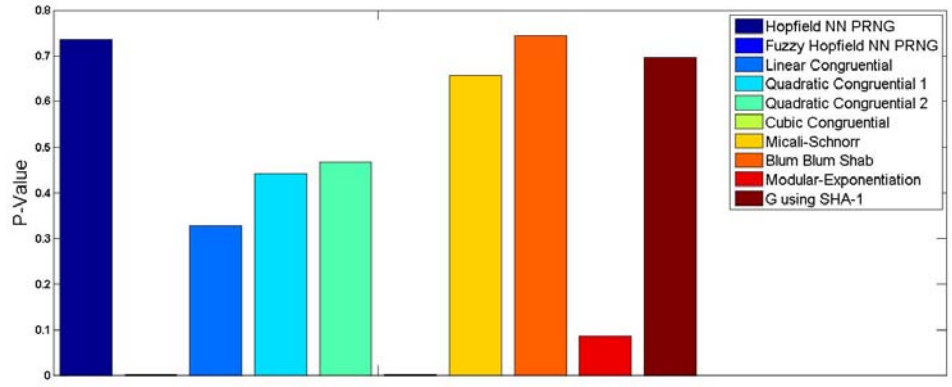

Figure 27 - Serial test result for different PRNGs

4.4.10 Linear Complexity test result

Linear Complexity is the last test of NIST PRNG test suite, that is discussed here. Figure 28 presents the linear complexity test results for the PRNGs, and all the PRNGs have passed. However, Blum Blum Shub PRNG has the best result among all other PRNGs in this test. It is followed by Micali-Schnorr and Linear Congruential PRNGs. The results obtained for FHNN and HNN are acceptable, but FHNN has better result than HNN.

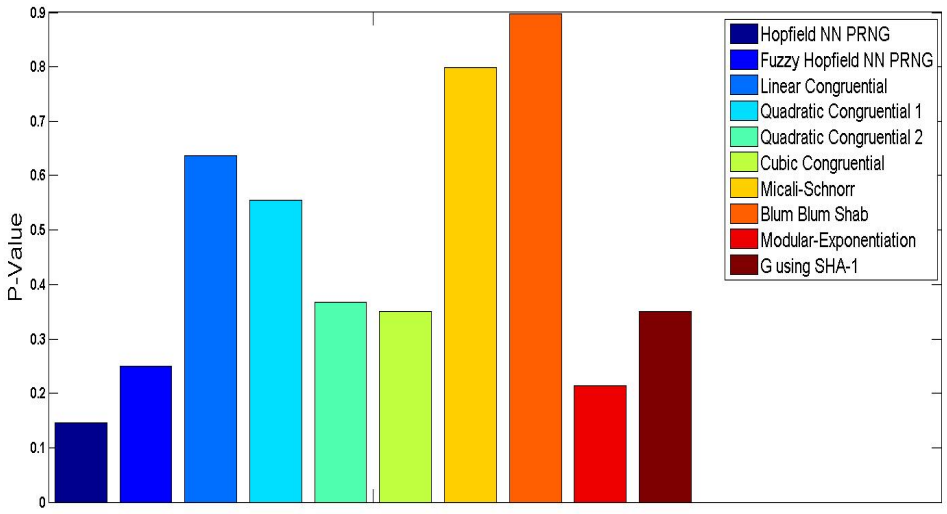

Figure 28 - Linear Complexity test result for different PRNGs

An ideal random number generator
is a fiction
Schindler and Killman

Chapter 5

Conclusion

In this research, we have integrated AI techniques, in particular neural network and fuzzy neural network with security issues. More precisely we have elaborated on generating random numbers, using neural networks. The contribution of the thesis would be enumerated as follows:

- We have examined the concept of Random number and its usage. We have investigated why random number generators with good quality of randomness are essential to all cryptographic applications. We have studied different types of RNGs, PRNG and TRNG, and have elaborated on their applicability and features in today's world.
- We have examined the usages of neural networks in different cryptosystems. We have tried to match the characteristic of PRNGs with the characteristics of the neural network (Hopfield NN) which is exploited for that problem. We have studied neural networks in cryptosystems. This showed us that there is a great possibility for using Neural Networks in cryptosystems and more over there are much more to be investigated both in neural network behavior and cryptosystems features. Specifically we considered the applications of neural networks as pseudo random number generator in the past years.
- We have investigated some of the standard PRNGs, their algorithms and their pros and cons.
- RNG test applications also have been studied and different RNG test suites are examined. We have carefully reviewed NIST PRNG test suite in particular.

- Inspired by the convergence problem of Hopfield Neural Networks, we examined using HNN as PRNG. We designed and developed a non-converging HNN based on which we produced and then evaluated the generated random sequence numbers using NIST PRNG Random test package. We showed that the number of digits that we extract from output of each neuron has a big influence in the performance of our HNN PRNG toward passing the PRNG tests. We showed that by increasing the number of digits extracted from the output of each neuron in HNN PRNG, a better PRNG would be achieved.
- We developed a Fuzzy Hopfield Neural Networks with few numbers of neurons. By reducing the number of neurons, we decreased the size of the weight matrix of FHNN. Consequently it would be much easier for the users setting the weight matrix which be used as a seed to the PRNG. The FHNN based PRNG were then put to test using NIST PRNG Random test package. We showed that the number of neurons has a big influence in the performance of our HNN PRNG for passing the PRNG tests.
- We tested other standard PRNGs and compared their results with our proposed HNN. We showed that our proposed PRNGs have good quality compared to other PRNGs.

The limitation of our work would be enumerated as:

- Our proposed HNN PRNG and FHNN PRNG are complex system compared to other PRNGs
- HNN PRNG is not good enough if inappropriate seed (weak seed) is selected by user.
- In FHNN PRNGs, the number of rules was tightening up with the number of neurons. By increasing the number of neurons the number of rules in fuzzy system increases that makes FHNN PRNG more complex and slow.

- In bit sampling mechanism, the upper bound of our accuracy in calculation is the size of processor registers (16 digits after decimal point). Using symbolic calculation toolbox of matlab to gain bigger level of accuracy made the system slow, and slow PRNG is not that useful.

This work would be continued in the following directions as future works:

- Investigating the relation of the number of neurons to the quality of generated random numbers in HNN PRNG.
- Implementing HNN with spiking neurons. The main problem with spiking neurons is complexity, HNN with spiking neurons as PRNGs shouldn't be that complex.
- Studying other activation functions or procedures instead of $tanh\,(x)$ for HNN and FHNN PRNG
- Studying other method of integrating fuzzy logic systems and neural networks for developing FHNN could be studied.
- Other mechanism of bit sampling could be studied and applied on HNN and FHNN PRNG

Appendix

This project was developed in Matlab 2009a platform. The NIST PRNGs test suite and all PRNGs except HNN and FHNN were ran on Linux machine. All the results that were shown in this thesis are based on the report of results obtained from NIST PRNG test suit for the PRNGs.

All the tests in this thesis executed with 99% level of confidence. All the PRNGs used to generate 100 sequences of 1000000 bits as random numbers. And these bit sequences tested by PRNGs with α parameter equal to 0.01.

In the following the results of NIST test suite for different PRNGs (except HNN and FHNN) presented in more details.

Table 16 - NIST Results of Linear Congruential

PRNG Test	**Linear Congruential**	
	P-Value	*Prop*
Frequency	0.6786	0.99
Block Frequency	0.4372	0.99
Cumulative Sums-Forward	0.0805	0.99
Cumulative Sums-Reverse	0.7791	0.99
Runs	0.9357	0.97
Longest Run	0.2896	1
Rank	0.4559	0.99
Non Overlapping Template	0.5254	0.989
Overlapping Template	0.5341	0.99
Universal	0.5544	0.99
Approximate Entropy	0.4559	0.99
Random Excursions	0.3040	0.993
Random Excursions Variant	0.4067	0.993
Serial(m=5)	0.3282	0.98
Linear Complexity	0.6371	0.99

Table 15 - NIST Results for Cubic Congruential

PRNG Test	**Cubic Congruential**	
	P-Value	*Prop*
Frequency	0*	0.67*
Block Frequency	0*	1
Cumulative Sums-Forward	0*	0.66*
Cumulative Sums-Reverse	0*	0.71*
Runs	0*	0*
Longest Run	0.3504	0.99
Rank	0.2492	0.99
Non Overlapping Template	0.3555	0.973
Overlapping Template	0.0589	0.99
Universal	0.9642	0.99
Approximate Entropy	0*	0.77*
Random Excursions	0.6340	0.987
Random Excursions Variant	0.4063	0.992
Serial(m=5)	0*	0.18*
Linear Complexity	0.3504	0.97

Table 17 - NIST Results for Quadratic Congruential 1

PRNG Test	Quadratic Congruential 1	
	P-Value	*Prop*
Frequency	0	0.55*
Block Frequency	0.0308	0.99
Cumulative Sums-Forward	0	0.61*
Cumulative Sums-Reverse	0	0.62*
Runs	0.6993	0.91*
Longest Run	0.9878	0.98
Rank	0.6786	0.99
Non Overlapping Template	0.4982	0.988*
Overlapping Template	0.3669	0.99
Universal	0.5955	0.99
Approximate Entropy	0.4749	0.99
Random Excursions	0.2180	0.985*
Random Excursions Variant	0.4743	0.9936
Serial(m=5)	0.4415	0.945*
Linear Complexity	0.5544	0.99

Table 18 - NIST Results of Quadratic Congruential 2

PRNG Test	Quadratic Congruential 2	
	P-Value	*Prop*
Frequency	0	0.74*
Block Frequency	0.7197	0.99
Cumulative Sums-Forward	0	0.76*
Cumulative Sums-Reverse	0	0.72*
Runs	0.3345	0.96
Longest Run	0.9463	0.99
Rank	0.4559	1
Non Overlapping Template	0.5156	0.9893
Overlapping Template	0.7981	1
Universal	0.2492	0.98
Approximate Entropy	0.5749	0.99
Random Excursions	0.5221	0.987*
Random Excursions Variant	0.3645	0.977*
Serial(m=5)	0.4678	0.965*
Linear Complexity	0.3669	0.98

Table 19- NIST Results of Blum Blum Shub

PRNG Test	**Blum Blum Shub**	
	P-Value	*Prop*
Frequency	0.9558	1
Block Frequency	0.6993	0.99
Cumulative Sums-Forward	0.2757	1
Cumulative Sums-Reverse	0.5341	1
Runs	0.5544	0.99
Longest Run	0.4943	0.98
Rank	0.5544	0.98
Non Overlapping Template	0.4906	0.9899
Overlapping Template	0.3838	0.98
Universal	0.6579	0.99
Approximate Entropy	0.4011	1
Random Excursions	0.2414	0.9920
Random Excursions Variant	0.4589	0.9964
Serial(m=5)	0.7442	0.995
Linear Complexity	0.8977	1

Table 20- NIST Results for Micali-Schnorr

PRNG Test	**Micali-Schnorr**	
	P-Value	*Prop*
Frequency	0.0167	0.96
Block Frequency	0.9834	0.99
Cumulative Sums-Forward	0.3504	0.98
Cumulative Sums-Reverse	0.3669	0.97
Runs	0.1296	1
Longest Run	0.8513	1
Rank	0.0965	0.98
Non Overlapping Template	0.4752	0.9897
Overlapping Template	0.5749	0.99
Universal	0.1372	0.98
Approximate Entropy	0.2757	0.98
Random Excursions	0.5263	0.9923
Random Excursions Variant	0.4858	0.9897
Serial(m=5)	0.6566	0.98
Linear Complexity	0.7981	0.97

Table 22- NIST Results of Modular-Exponentiation

PRNG Test	Modular Exponentiation	
	P-Value	*Prop*
Frequency	0	0.67*
Block Frequency	0.0329	0.98
Cumulative Sums-Forward	0	0.72*
Cumulative Sums-Reverse	0	0.69*
Runs	0.8513	0.96
Longest Run	0.5749	0.99
Rank	0.8165	0.97
Non Overlapping Template	0.5435	0.9899
Overlapping Template	0.0805	1
Universal	0.1223	1
Approximate Entropy	0.2622	1
Random Excursions	0.3481	0.9965
Random Excursions Variant	0.2995	0.9938
Serial(m=5)	0.0859	0.95*
Linear Complexity	0.2133	0.99

Table 21- NIST Results for G using SHA-1

PRNG Test	G using SHA-1	
	P-Value	*Prop*
Frequency	0.5544	0.98
Block Frequency	0.9357	0.98
Cumulative Sums-Forward	0.8831	0.98
Cumulative Sums-Reverse	0.1815	0.99
Runs	0.2896	1
Longest Run	0.8977	0.99
Rank	0.2022	1
Non Overlapping Template	0.5179	0.9902
Overlapping Template	0.0855	0.98
Universal	0.1087	0.99
Approximate Entropy	0.6786	1
Random Excursions	0.2226	0.9892
Random Excursions Variant	0.2669	0.9885
Serial(m=5)	0.6957	0.99
Linear Complexity	0.3504	0.99

References

[1] K. Tirdad and A. Sadeghian, "Hopfield Neural Networks as Pseudo Random Number Generators," in *North America Fuzzy Information Processing Society (NAFIPS)*, Toronto, July 2010, pp. 1-6.

[2] K. Tirdad and A. Sadeghian, "Fuzzy Hopfield Neural Networks as Pseudo Random Number Generators," in *Submitted to International Conference on Computer and Computational Intelligence (ICCCI)*, 2010.

[3] I. Woungang, A. Sadeghian, S. Wu, S. Misra, and M. Arvandi, "Wireless Web Security Using a Neural Network-Based Cipher," in *Web Services Security and E-Business*, G. Radhammani and G. S.V. Radha Krishna Rao (Eds.), Ed.: Idea Group Publishing Inc, ch. II, pp. 32-56.

[4] T. Schmidt, H. Rahnama and A. Sadeghian, "A Review of Applications of Artificial Neural Networks in Cryptosystems," in *International Symposium on Soft Computing for Industry (ISSCI)* , 2008.

[5] Fabio Pareschi, "Chaos-Based Random Number Generators:Monolightic Implementation, Testing and Applications," Universt`a Di Bologna, PhD Thesis 2006.

[6] R. Gennaro, "Randomness in cryptography," *IEEE Journal of Security & Privacy*, vol. 4, no. 2, pp. 64-67, April 2006.

[7] Lesley Adkins and Roy A. Adkins, *Handbook to life in ancient Rome*. US: Oxford University Press , 1998.

[8] Sarah Iles Johnston, *Religions of the ancient world*. US: Harvard University Press, 2004.

[9] Gaming Statutes and Relgulations. Nevada Gaming Commission and State Gaming Control Board. [Online]. http://gaming.nv.gov/stats_regs.htm

[10] Joan Daemen and Vincent Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*.: Springer-Verlag, 2002.

[11] National Institute of Standard and Technology, "Data Encription Standard," *Federal Information Processing Standard (FIPS) publication 43-3*, October 1999. [Online]. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[12] National Institute of Standard and Technology, "Advance Encription Standard," *Federal Information Processing Standard (FIPS) publication 197*, November 2001. [Online]. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[13] D. E. Eastlake, S. D. Crocker and J. I. Shiller, "RFC 1750: Randomness recommendation for security," *Internet Society Request for Comments, Internet Engineering Task Force*, December 1994.

[14] Bruce Schneier, *Applied Cryptography*, 2nd ed. Toronto, Canada: John Wiley & Sons, 1996.

[15] Security Requirements for Cryptographic Modules Approved Random Number Generators for FIPS PUB 140-2. (2009, July) NIST. [Online]. http://csrc.nist.gov/publications/fips/fips140-2/fips1402annexc.pdf

[16] J. von Neumann, "Various Technique Used in Connection with Random Digits," *in Applied Math Series, notes by G. E. Forsythe, National Bureau of Standards*, vol. 12, pp. 36–38,

1951.

[17] Donald. E. Knuth, *The Art of Computer Programming*, 3rd ed.: Addison-Wesley Professional, 1997, vol. Seminumerical Algorithms.

[18] B. Sunar, W. J. Martin, and D. R. Stinson, "A Provably Secure True Random Number Generator with Built-in Tolerance to Active Attacks," University ofWaterloo (Canada), Technical Report CACR 2005-20 2005.

[19] NIST. (2008, August) A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. [Online]. http://csrc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf

[20] F. Pareschi, R. Rovatti, and G. Setti, "Second-level NIST Randomness Tests for Improving Test Reliability," in *IEEE International Symposium on Circuits and Systems*, May 2007, pp. 1437-1440.

[21] G. Marsaglia and W. W. Tsang, "Some difficult-to-pass tests of randomness," *Journal of Statistical Software*, vol. 7, no. 3, pp. 1-8, January 2002.

[22] Y. Wang, G. Wang and H. Zhang, "Random Number Generator Based on Hopfield Neural Network and SHA-2 (512)," *Advancing Computing, Communication, Control and Management*, vol. 56, pp. 198-205, December 2009.

[23] Y. H. Wang, Z. D. Shen and H. G. Zhang, "Pseudo random number generator based on hopfield neural network," in *Proceeding of fifth International Conference on Machine Learning and Cybernetic*, August 2006, pp. 2810-2813.

[24] J. J. Hopfield, "Neural Network & Physical Systems with Emergent Collective

Computational Abilities," in *Proc Natl Acad Sci U.S.A*, April 1982, pp. 2554-2558.

[25] L.A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338-353, 1965.

[26] L. A. Zadeh, "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," in *IEEE Transactions on Systems, Man, and Cybernetics*, January 1973, pp. 28-44.

[27] E. Cox, "Fuzzy Fundamentals," *IEEE Spectrum*, pp. 58-61, October 1993.

[28] J. Hong, "The Fuzzy Engine for Random Number Generator in Crypto Module," *Lecture Notes in Computer Science-4th International Conference on Networking, ICN 2005*, vol. 3421, pp. 953-963, March 2005.

[29] Karl Entacher. (2000, June) A collection of classical pseudorandom number generators with linear structures - advanced version. [Online]. http://crypto.mat.sbg.ac.at/results/karl/server/

[30] S.K. Park and K.W. Miller, "Random Number Generators: Good Ones Are Hard To Find," *Communications of the ACM,* pp. 1192-1201, vol.31, 1998.

[31] J. Stern, "Secret linear congruential generators are not cryptographically secure," in *28th Annual Symposium on Foundations of Computer Science* , Los Angeles, CA, USA, October 1987, pp. 421-426.

[32] D. Knuth, "Deciphering a linear congruential encryption," *IEEE Transactions on Information Theory*, vol. 31, no. 1, pp. 49-52, January 1985.

[33] J. Eichenauer and J. Lehn, "On the structure of quadratic congruential sequences," *manuscripta mathematica, Springer Berlin / Heidelberg*, vol. 58, no. 1-2, pp. 129-140,

March 1987.

[34] O. Goldreich , R. Israel and V. Rosen, "On the Security of Modular Exponentiation with Application to the Construction of Pseudorandom Generators," *journal of Cryptography*, vol. 16, Number 2, pp. 71-93, March 2008.

[35] Federal Information Processing Standards Publication 186. (1994, May) Appendix 3.3. [Online]. http://www.itl.nist.gov/fipspubs/fip186.htm

[36] A. Menezes, P. van Oorschot, and S. Vanstone. (CRC Press,1996, p. 175) Handbook of Applied Cryptography. [Online]. http://www.cacr.math.uwaterloo.ca/hac/about/chap5.pdf

[37] L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal on Computing*, vol. 15, pp. 364–383, May 1986.

[38] M. Geisler, M. Krøigård and A. Danielsen. (2004, December) About Random Bits. [Online]. http://www.cecm.sfu.ca/~monaganm/teaching/CryptographyF08/random-bits.pdf

[39] P. L'Ecuyer, "Software for uniform random number generation: distinguishing the good and the bad," *Winter Simulation Conference (WSC'01)*, vol. 2, pp. 95-105, December 2001.

[40] C. Kenny, Random Number Generators:An Evaluation and Comparison of Random.org and Some Commonly Used Generators, April, 2005, TRINITY COLLEGE DUBLIN,Management Science and Information Systems Studies, Project Report.

[41] G. Marsaglia. DIEHARD Statistical Tests. [Online]. http://www.stat.fsu.edu/pub/diehard

[42] J. Walker. A Pseudorandom Number Sequence Test Program. [Online]. http://www.fourmilab.ch/random/

[43] M. Rutti. (2004, Institute for Theoretical Physics, Zurich,) A Random Number Generator

Test Suite for the C++ Standard. [Online]. http://www.comp-phys.org/rngts/doc/main.pdf

[44] NIST. (2009, July) Approved Random Number Generators for FIPS PUB 140-2, Security Requirements for Cryptographic Modules. [Online]. http://www.csrc.nist.gov/publications/fips/fips140-2/fips1402annexc.pdf

[45] A. Poorghanad, A. Sadr and A. Kashanipour , "Generating High Quality Pseudo Random Number Using Evolutionary methods," in *International Conference on Computational Intelligence and Security, CIS '08.* , Suzhou, December 2008, pp. 331-335.

[46] F. Pareschi, G. Scotti, L. Giancane,R. Rovatti,G. Setti and A. Trifiletti, "Power analysis of a chaos-based Random Number Generator for cryptographic security," in *IEEE International Symposium on Circuits and Systems, ISCAS 2009.* , Taipei , May 2009, pp. 2858-2861.

[47] L. Cao, L. Min and H. Zang , "A Chaos-Based Pseudorandom Number Generator and Performance Analysis," in *International Conference on Computational Intelligence and Security, CIS '09.* , Beijing, December 2009, pp. 494-498.

[48] K. Ozdemir, S. Kilinc, and S. Ozoguz , "Random number generator design using continuous-time chaos," in *IEEE 16th Signal Processing, Communication and Applications Conference, SIU 2008.*, Aydin, April 2008, pp. 1-4.

[49] C. Y. Li, J. S. Chen and T. Y. Chang, "A chaos-based pseudo random number generator using timing-based reseeding method," in *IEEE International Symposium on Circuits and Systems, ISCAS 2006*, Island of Kos, September 2006, pp. 3276-3280.

[50] S. E. EI-Kbamy, M. Lotfy and A. H. Ali, "A New Fuzzy Logic Based Pseudo-Random Bit Generator for Secure DS-CDMA Systems," in *Twenty Second National Radio Science*

Conference (NRSC 2005), Cairo, Egypt, March 2005, pp. 377-384.

[51] R. Mislovaty, Y. Perchenok, I.Kanter and W.Kinzel, "A secure key-exchange protocol with an absence of injective function," *Physical Review E,* vol. 66, no. 6, June 2002.

[52] M. Volkmar and A.Schaumburg, "Authenticated tree parity machine key exchange," *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 421-427, November 2004.

[53] P. Li and Z. Yi, "Compound attack on synchronization based neural cryptography," in *Advances in Cognitive Neurodynamics,ICCN*, 2007, pp. 1019-1023.

[54] L. P. Yee and L. C. De Silva, "Application of multilayer perceptron networks in public key cryptogrraphy," in *International Joint Conference on Neural Network ,IJCNN*, 2002.

[55] M. Arvandi, S. Wu, A. Sadeghian, W. Melek, and I. Woungang, "Symmetric Cipher Design Using Recurrent Neura Networks," in *International Joint Conference on Neural Networks (IJCNN)*, 2006, pp. 2039-2046.

[56] 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.

[57] M. Arvandi and A. Sadeghian, "Chosen Plaintext Attack Against Neural Network-Based Symmetric Cipher," in *Int'l Joint Conf. on Neural Networks*, 2007, pp. 847-852.

[58] E. C. Laskari, G. C. Meletion, D. K. Tasoulis and M. N. Vrahatis, "Studing the performance of artificial neural networks on problems related to cryptography," in *Nonlinear Analysis: Real World Applications*, vol. 7, Issue 5, December 2006, pp. 937-942.

[59] B. Janson and K. Nakayama, "Neural network following a binary approach applied to the

integer prime-factorization problem," in *International Joint Conference on Neural Network ,IJCNN*, Montreal, 2005, pp. 2577-2582.

[60] S. Lian, J. Sun and Z. Wang, "Secure Hash Function based on neural network," *Neurocomputing*, vol. 69, no. 16-18, pp. 2346-2350, October 2006.

[61] S. Lian, J. Sun and Z. Wang, "one way Hash Function based on neural network," , July 2007, pp. 1-7.

[62] M. Naor and A. Shamir, "Visual Cryptography," in *Advances in Cryptography, EUROCRYPT'94*, Berlin, 1995, pp. 1-12.

[63] T. W. Yue and S. Chiang , "A neural network approach for visual cryptography," in *IEEE-INNS-ENNS International Joint Conference on Neural Network, IJCNN*, 2000, pp. vol.5, pp. 494-499.

[64] T. W. Yue and S. Chiang, "The general neural-network paradigm for visual cryptography," in *Connectopnist Model of Neurons, Learning Process and Artificial Intelligence*, 2001, pp. 196-206.

[65] T. W. Yue and S.Chiang, "The semipublic encryption for visual cryptography using Q'tron neural networks," *Journal of Network and Computer Application*, vol. 30, no. 1, pp. 24-41, January 2007.

[66] G. Song,and X. Miao C. Peng, "Visual cryptographyscheme using Pi-sigma neural networks," in *International symposium on information science and engineering, ISISE'08*, Dec 2008, pp. vol.2, pp.679-682.

[67] P. T. Yu, H. H. Tsai and J. S. Lin, "Digital watermarking based on neural networks for

color images," *Signal Processing*, vol. 81, no. 3, pp. 663-671, March 2001.

[68] M.S. Hwang, C. C. Chang and K. F. Hwang, "Digital watermarking of images using neural networks," *Journal of electronic imaging*, vol. 9, no. 4, pp. 548-555, October 2000.

[69] Er. A. Bansal, S. S. Bhadauria and R. Gupta, "Application of backpropagation neural network to generate fragmented watermarks and full watermark by union," *International Journal of computerscience and security, IJCSNS*, vol. 8, no. 10, pp. 191-199, October 2008.

[70] A. E. Hassanein, A. Abraham and C. Grosan, "Spiking neural network and wavelets for hiding iris data in digital images," *Journal soft computing - a fusion of fundations, methodologies and applications*, vol. 13, no. 4, pp. 401-416, 2008.

[71] K. Tsuruta and Y. Nishio, "Reversible watermarking using small-world cellular neural network," in *International workshop on nonlinear circuit and signal processing, NCSP'05*, March 2005, pp. 21-24.

[72] W. Zhenfei, Z. Guangqun and W. Nengchao, "Digital watermarking algorithm based on wavelet transform and neural network," *Wuhan University Journal of Natural Sciences*, vol. 11, no. 6, pp. 1667-1670, November 2006.

[73] J. Snag, X. Liao and M. S. Alam, "Neural-network-based zero-watermark scheme for digital images," *Optical engineering*, vol. 45, no. 9, 2006.

[74] J. Snag and M. S. Alam, "A neural network based lossless digital image watermarking in the spatial Domain," *Advances in Neural Networks – ISNN 2005*, vol. 3497, pp. 772-776, May 2005.

[75] L. Cao, X. Wang, Z. Wang and S. Bai, "Neural network based audio watermarking algorithm," *Proceedings of SPIE--the international society for optical engineering*, vol. 6041, pp. 175-179, 2005.

[76] H. H. Tsai, J.S.Cheng, P. T. Yu, "Audio Watermarking based on HAS ans Neural Networks in DCT Domain," *Journal on applied signal processing EURASIP*, vol. 3, pp. 252-263, 2003.

[77] L. Shaohui, Y. Hongxun and G. Wen, "Neural network based steganalysis in still images," in *International conference on Multimedia and Expo*, July 2003, pp. vol.2 pp.11-12.

[78] R. Benton and H. Chu, "Soft computing approach to steganalysis of LSB embedding in digital Images," in *3rd International Conference on Information Technology:Research and Education, ITRE2005*, June 2005, pp. 105-109.

[79] V. Sabeti, S. Samavi, M. Mahdavi and S. Shirani, "Steganalysis of embedding in difference of image pixel pairs by neural network," in *The ISC International Journal of Information Security*, January 2009, pp. vol.1, no.1, pp.17-26.

[80] M. Murshed, "A survey on neural network based steganalysis," in *7th International Conference on computer and information technology*, December 2004, pp. 26-28.

[81] Y. Q. Shi, G.Xuan, D.Zou and J.Gao, "Image steganalysis based on moments of characteristic functions using wavelet decomposition, prediction-error image, and neural network," in *In Proceedings of ICME'2005*, 2005, pp. 296-272.

[82] S. R. Baragada, S. Ramakrishna, M. S. Rao and S. Purushothaman, "Implementation of radial basis function neural network for image steganalysis," in *International Journal of*

Computer Science and Security, July 2008, pp. vol.2, Issue.1, pp.12-22.

[83] D. A. Karras and V. Zorkadis, "Overfitting in multilayer perceptron as a mechanism for (pseudo) random number generation in the desgin of secure electronic commerce systems," in *Information Systems for Enhanced Public Safety and Security, IEEE/AFCEA, EUROCOM*, Munich, 2000, pp. 345 - 349.

[84] J. M. Hughes, "Pesudu-random number generation usimg binary recurrent neural networks," Dartmouth, Kalamazoo College, http://hdl.handle.net/10090/6348, April 2007.

[85] B. J. Wang, H. J. Cao, Y. H. Wang and H. G. Zhang, "Random number generator of BP neural network based on SHA-2(512)," in *Proceeding of Sixth International Conference on Machine Learning and Cybernetic*, August 2007, pp. Vol. 5, pp.2708-2712.

[86] Simon Haykin, *Neural Networks and Learning Machines*, 3rd ed. New Jersey, US: Prentice Hall, 2009.

[87] M. B. Menhaj and N. Seifipour, "A new implementation of discrete-time Hopfield net with higher capacity and speed of convergence," in *International Joint Conference on Neural Networks*, 2001, pp. 436-441 vol.1.

[88] R. Ma, P. Chu and S. Zhan, "Stability Conditions for Discrete Delayed Hopfield Neural Networks," in *Third International Conference on Natural Computation*, August 2007, pp. 468-472.

[89] R. Ma, Y. Xie, S. Zhang and W. Liu , "Convergence of discrete delayed Hopfield neural networks," *Computers & Mathematics with Applications*, vol. 57, no. 11-12, pp. 1869-1876, June 2009.

[90] R.E. Uhrig, "Introduction to artificial neural networks," in *21st International Conference on Industrial Electronics, Control, and Instrumentation, IEEE IECON*, Orlando, FL, November 1995, pp. 33-37.

[91] TommyW S Chow and Siu-Yeung Cho, *Neural Networks and Computing-Learning Algorithms and Applications*, 7th ed. Singapore: Imperial College Press, 2007.

[92] Ben Krose and Patrick van der Smagt, *An Introduction to Neural Networks*. Amsterdam: The University of Amsterdam, November 1996.

[93] Howard Demuth,Mark Beale and Martin Hagan. (2010, March) Neural Network Tool Box 6 user Guide, Version 6.0.4. [Online]. http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf

[94] James A. Freeman and David M. Skapura, *Neural Networks - Algorithms, Applications and Programming Techniques*.: Addison-Wesley Publication Company, 1991.

[95] James. A. Anderson, *An Introduction to Neural Network*.: MIT Press, 1995.

[96] Colin Fyfe. (1996) Artificial Neural Networks. Department of Compputer and Information Systems, The University of Paisley.

[97] J. M. Mendel, "Fuzzy logic systems for engineering: a tutorial," in *Proceedings of the IEEE* , March 1995, pp. 345-377.

[98] C.L. Wang, F. Shilu and D. Qi, "The Stability of a Class of Generalized Fuzzy Hopfield Networks," in *International Conference on Apperceiving Computing and Intelligence Analysis, ICACIA*, July 2009, pp. 108-111.

[99] Li Sheng and Huizhong Yang, "Delay-dependent exponential stability analysis of fuzzy

delayed Hopfield neural networks: a fuzzy Lyapunov-Krasovskii functional approach," in *American Control Conference*, 2009, pp. 4296-4301.

[100] H. Huang, D. Ho and J. Lam, "Stochastic stability analysis of fuzzy Hopfield neural networks with time-varying delays," *IEEE Trans. Circuits and Systems-II*, vol. 52, pp. 251-255, 2005.

[101] X. Lou and B. Cui, "Robust asymptotic stability of uncertain fuzzy BAM neural networks with time-varying delays," *Fuzzy Sets and Systems*, vol. 158, pp. 2746-2756, 2007.