ABHINAV
NATIONAL MONTHLY REFEREED JOURNAL OF REASEARCH IN SCIENCE & TECHNOLOGY
www.abhinavjournal.com

CRYPTOGRAPHY IN STRUCTURE ADAPTABLE DIGITAL NURAL NETWORKS

Pratap Singh[1] and Dr. Harvir Singh[2]

[1]Lecture, IMS College, Roorkee
pspartapsingh@gmail.com
[2]Director & Principal, Hindu college of Engineering, Hariyana

ABSTRACT

Neural networks can be used to generate common secret key. The goal of any cryptographic system is the exchange of information among the intended users without any leakage of information to others who may have unauthorized access to it. A common secret key could be created over a public channel accessible to any opponent. In case of structure adaptable digital neural cryptography, both the communicating networks receive an identical input vector, generate an output bit and are trained based on the output bit. The two networks and their weight vectors exhibit a novel phenomenon, where the networks synchronize to a state with identical time-dependent weights. The generated secret key over a public channel is used for encrypting and decrypting the information being sent on the channel.

Keywords: Structure Adaptable Digital Neural Cryptography, Mutual Learning, Cryptographic System, Key Generation.

INTRODUCTION

The basic need to provide security is using cryptography. Nowadays information security has become an important aspect in every organization. In other words, the people have to be assured that the information to be read by only the sender and receiver. In our work we are combining structure adaptable digital neural network and cryptography.

Cryptography

Cryptography can also be defined as the conversion of data into a scrambled code that can be deciphered and sent across a public or private network. Cryptography is the practice and study of hiding information. It is an essential aspect for secure communication. Cryptography [1] not only protects data from theft or alternation but also can be used for user authentication. Cryptography uses two main styles or forms of encrypting data; symmetrical and asymmetrical. Symmetric encryptions, or algorithms, use the same key for encryption as they do for decryption. Other names for this type of encryption are secret-key, shared-key, and private-key. Asymmetric cryptography uses different encryption keys for encryption and decryption. In this case an end user on a network, public or private, has a pair of keys; one for encryption and one for decryption. These keys are labeled or known as a public and a private key.

Within the context of any application-to-application communication, there are some specific security requirements, including:

- Integrity: Assuring the receiver that the received message has not been altered in any way from the original.
- Non-repudiation: A mechanism to prove that the sender really sent this message.
- Authentication: The process of proving one's identity.
- Privacy/confidentiality: Ensuring that no one can read the message except the intended receiver.

Cryptography Definitions

Plaintext is either in a form that can be understood by a person (a document) or by a computer (executable code). Once it is not transformed into cipher text, human nor can machine properly process it until it is decrypted. This enables the transmission of confidential information over insecure channels without unauthorized disclosure. When this same sensitive information is sent over a network, it can no longer take these controls for granted, and the information is in a much more vulnerable state.

A system that provides encryption and decryption is referred to as a cryptosystem and can be created through hardware components or program code in an application. The cryptosystem uses an encryption algorithm, which determines how simple or complex the process will be. Most algorithms are complex mathematical formulas that are applied in a specific sequence to the plaintext. Most encryption methods use a secret value called a key (usually a long string of bits), which works with the algorithm to encrypt and decrypt the text, as shown in figure 1.

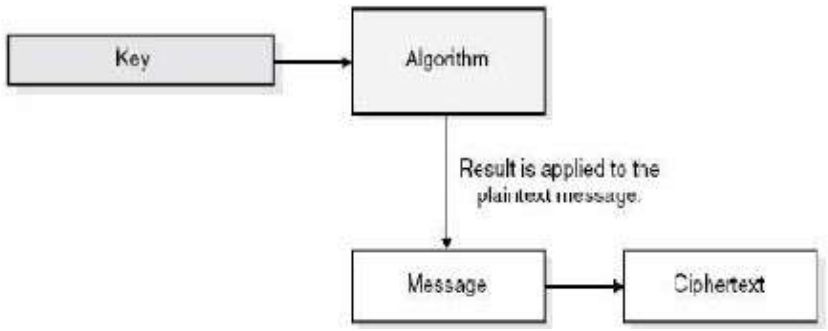

Figure 1: Use of a key

If an eavesdropper captures a message as it passes between two people, she can view the message, but it appears in its encrypted form and is therefore unusable. Even if this attacker knows the algorithm that the two people are using to encrypt and decrypt their information, without the key, this information remains useless to the eavesdropper. So we can say that key is very important.

Structure Adaptable Digital Nural Network

Structure adaptable digital neural networks are parallel adaptive networks consisting of simple nonlinear computing elements called neurons which are intended to abstract and model some

of the functionality of the human nervous system in an attempt to partially capture some of its computational strengths. Neural networks [2] [3] are non-linear statistical data modeling tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data. A phenomenon of neural network is applied in cryptography systems. This is used for generating secret key over public channel.

Structure adaptable digital neural networks with their remarkable ability to derive meaning from complicated or imprecise data can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques [4] [5]. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

Background

In cryptography pseudorandom number generators (PRNG's) were used to generate secret keys between two communicating parties. These typically start with a "seed" quantity and use numeric or logical operations to produce a sequence of values. A typical pseudo-random number generation technique is known as a linear congruence pseudo-random number generator. These are the mechanisms used by real-world secure systems to generate cryptographic keys, initialization vectors, \random" nonce's and other values assumed to be random. But here there are some possible attacks against PRNG's [6]. Here an attacker may cause a given PRNG to fail to appear random, or ways he can use knowledge of some PRNG outputs (such as initialization vectors) to guess other PRNG outputs (such as secret key). Hence to overcome this disadvantage neural network is used in cryptography to generate the secret key. Here the generated secret key is random [7] [8].

Neural Cryptography

Interacting Neural Network and Cryptography

Two identical dynamical systems, starting from different initial conditions, can be synchronized by a common external-signal which is coupled to the two systems. Two networks which are trained on their mutual output can synchronize to a time ependent state of identical synaptic weights [9]. This phenomenon is also applied to cryptography [10]. Structure adaptable digital neural networks learn from examples. This concept has extensively been investigated using models and methods of statistical mechanics [11] - [12]. A "teacher" network is presenting input/output pairs of high dimensional data, and a student" network is

being trained on these data. Training means, that synaptic weights adopt by simple rules to the input/output pairs [13]. After the training phase the student is able to generalize: It can classify – with some probability – an input pattern which did not belong to the training set. In this case, the two partners A and B do not have to share a common secret but use their identical weights as a secret key needed for encryption. In structure adaptable digital neural network an attacker E who knows all the details of the algorithm and records any communication transmitted through this channel finds it difficult to synchronize with the parties, and hence to calculate the common secret key. We assume that the attacker E knows the algorithm, the sequence of input vectors and the sequence of output bits. Initial weight vectors and calculate the ones which are consistent with the input/output sequence. It has been shown, that all of these initial states move towards the same final weight vector, the key is unique [14]. However, this task is computationally infeasible. In principle, E could start from all of the Synchronization by mutual learning (A and B) is much faster than learning by listening (E). structure adaptable digital neural cryptography is much simpler than the commonly used algorithms [15] [16].

Algorithm

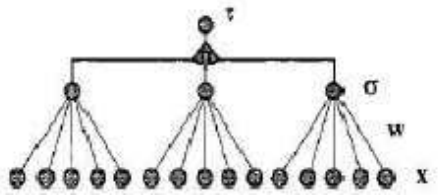

Figure 2: Tree Parity Machine

Here is a simple neural network as shown in figure 2. It consists of an input vector x, a hidden layer sigma, a weights coefficients w between input vector and the hidden layer which is an activation procedure that counts the result value t. Such a neural network is called as neural machine. It is described by three parameters: K-the number of hidden neurons, N-the number of input neurons connected to each hidden neuron, and L-the maximum value for weight {-L...+L}. Two partners have the same neural machines. Output value is calculated by

$$\tau = \pi \prod_{i=1}^{K} SIGN\left[\sum_{j=1}^{N} w_{i,j}\ x_{i,j}\right]$$

We update the weights only if the output values of neural machines are equal. There are three different rules like Hebbian learning rule [17] [18], Anti-Hebbian learning rule and Random-walk learning rule

- Hebbian learning rule:

$$w_{i,j}^{+} = g(\, w_{i,j} \, + \, x_{i,j} \, \tau\theta(\, \sigma_i \tau \,)\theta \, (\tau^A \, \tau^B \,))$$

- Anti-Hebbian learning rule:

$$w_{i,j}^{+} = g(\, w_{i,j} \, - \, x_{i,j} \, \tau\theta(\, \sigma_i \tau \,)\theta \, (\tau^A \tau^B \,))$$

- Random-walk learning rule:

$$w_{i,j}^{+} = g(\, w_{i,j} \, + \, x_{i,j} \, \theta(\sigma_i \tau)\theta(\tau^A \tau^B))$$

Secret Key Generation

Key Generation

The different stages in the secret key generation procedure which is based on neural networks can be stated as follow [19]: as shown in figure 3.

1. Determination of neural network parameters: k, the number of hidden layer units n, the input layer units for each hidden layer unit l, the range of synaptic weight values is done by the two machines A and B.
2. The network weights to be initialized randomly.
3. The following steps are repeated until synchronization occurs.
4. Inputs are generated by a third party (say the key distribution centre).
5. The inputs of the hidden units are calculated.
6. The output bit is generated and exchanged between the two machines A and B.
7. If the output vectors of both the machines agree with each other then the corresponding weights are modified using the Hebbian learning rule, Anti-Hebbian learning rule and Random-walk learning rule [20].
8. When synchronization is finally occurred, the synaptic weights are same for both the networks. And these weights are used as secret key.

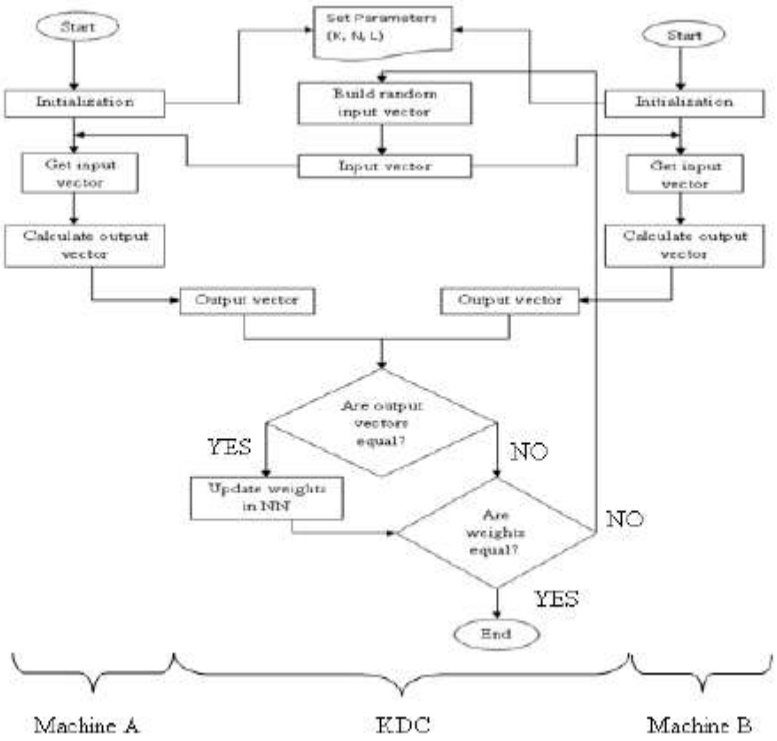

Figure 3: Key Generation

Implementation

In this section we will describe how to program neural machines and will show how to use MATLAB. The following figure 4 shows how to synchronize the two machines.

The main function used here is tTPM (TPMTree Parity Machine [21]). It contains vectors: H and W. 'H' is used for internal operations during result value counting. 'W' contains weights. There are also four integer values: K, L, N, and TPOutput. Here in every iteration, we should produce the input vector, the count output value by using functions tInputVector() and CountResult(). Whenever the output of the two machines is same then the weights are updated using UpdateWeight() function. The FormRandomVector() function is used to find the random input vectors by the key distribution centre. To find the random bit the randi function from MATLAB is used which uniformly distributes pseudorandom integers.

RESULT

Table 1. Result table for Neutral Network Key Generator

Sr.No.	Different issues	With NN	Without NN
1	Synchronization time	Required	Not required
2	Randomness	More	No
3	Security	More	Less

Table 1 shows analysis made for neural network key generator.

Synchronization Time

The data set obtained for synchronization time by varying number of input units (n) is shown in figure 4. The number of iterations required for synchronization by varying number of input units (n) is shown in figure 5. The two figures show that as the value of n increases, the synchronization time and number of iterations also increases. The %iterations (actual/max) required for synchronization by varying number of input units (n) is shown in figure 6. The %synchronization time per iteration required for synchronization by varying number of input units (n) is shown in figure 7. In these two figures it is shown that as the value of n increases, the %iterations (actual/max) and %synchronization time per iteration is decreased.

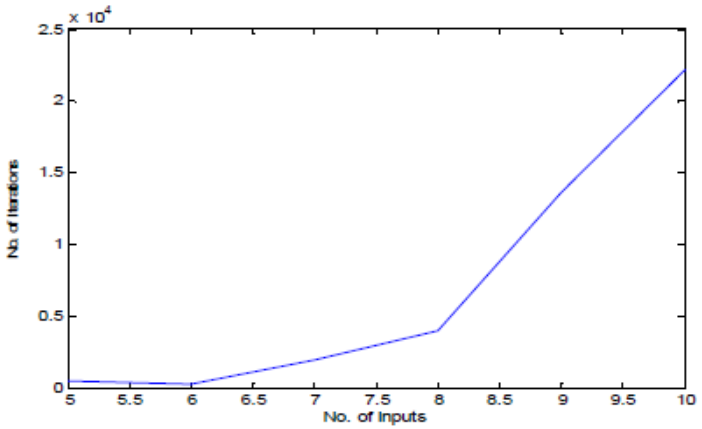

Figure 4: No of Input Units Vs No of Iteration

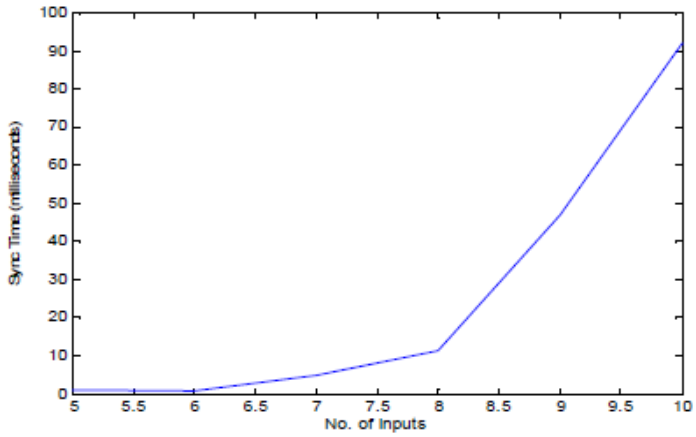

Figure 5: No of Input Units Vs Sync. Time

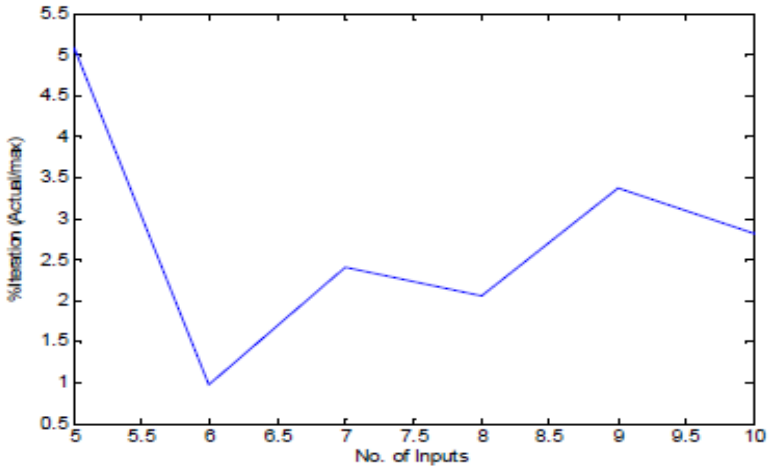

Figure 6: No of Input Units Vs %Iteration

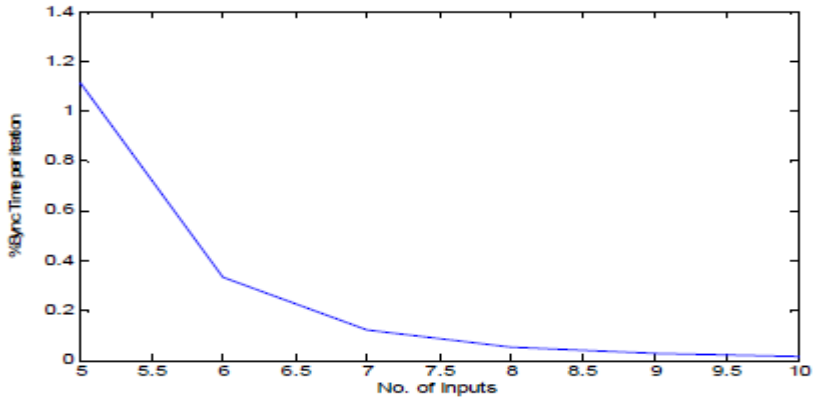

Figure 7: No of Input Units Vs Sync. Time per Iteration

Randomness

A random process is one whose consequences are unknown. Intuitively, this is why randomness is crucial in our work because it provides a way to create information that an adversary can't learn or predict. When speaking about randomness, we commonly mean a sequence of independent random numbers, where each number was obtained completely random and has absolutely no correlation between any other numbers inside the sequence. Here in our work in each iteration we are going to get the different keys, hence randomness is more as shown in figure 8. So here an attacker cannot predict the key.

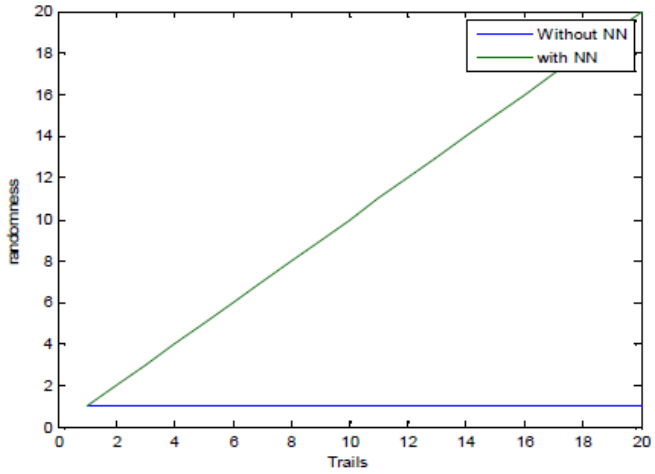

Figure 8: Randomness Vs No. of Trials

Here we can say that security is directly proportional to randomness, hence we have achieved security as well.

CONCLUSION

Interacting structure adaptable digital neural networks have been calculated analytically. At each training step two networks receive a common random input vector and learn their mutual output bits. A new phenomenon has been observed: Synchronization by mutual learning. The two partners can agree on a common secret key over a public channel. An opponent who is recording the public exchange of training examples cannot obtain full information about the secrete key used for encryption .This works if the two partners use multilayer networks, parity machines. We have shown graphs by which we can come to know that synchronization time will go on decreasing as the number of inputs increase. The opponent has all the information (except the initial weight vectors) of the two partners and uses the same algorithms. Nevertheless he does not synchronize. Here we have also achieved the randomness of key.

FUTURE WORK

The key distribution centre generated the secret key. Our future work is that the key distribution centre will distribute the generated key securely by some method.

REFERENCES

1. William Stallings, Cryptography and Network Security.
2. Jacek M. Zurada, Introduction to Artificial Neural Systems.
3. John A. Bullinaria: Introduction to Neural Networks, 2004.
4. Eric S Imsand, Deon Garrett, John A. Hamilton, Jr., Member, IEEE: User Identification Using GUI Manipulation Patterns and Artificial Neural Networks.
5. Henry A. Rowley, Shumeet Baluja, and Takeo Kanade: Neural Network-Based Face Detection, January 1998.

6. John Kelsey, Bruce Schneier, David Wagner, Chris Hall: Cryptanalytic Attacks on Pseudorandom Number Generators.
7. Peter Gutmann, David Naccache, Charles C. Palmer: Randomness in Cryptography, 2006.
8. Jorg Muhlbacher, DI Rudolf Hormanseder: Randomness in Cryptography, October 2007. [9] R. Metzler and W. Kinzel and 1. Kanter, Phys. Rev. E, 62, 2555 (2000).
9. Kaniter, W. Kinzel and E. Kanter, Europhys. Lett, 57,141-147 (2002).
10. M. Rosen-Zvi, E. Klein, 1. Kanter and W. Kinzel, "Mutual leamning in a treepanty machine and its application to cryptography", Phys. Rev. E (2002).
11. Neural Synchronization and Cryptography – Andreas Ruttor. PhD thesis, Bayerische Julius- Maximilians-Universitat Wurzburg, 2006.
12. Steve Lawrence, C. Lee Giles, Ah Chung Tsoi: Lessons in Neural Network Training: Overfitting May be Harder than Expected, 1997.
13. R. Urbanczik, private communication
14. C.P. Williams and S.H. Cleat-water, Explorations in Quantum Computing, Springer Verlag, 1998. [16] D. R. Stinson, Cryptography: Theory and Practice (CRC Press 2002).
15. Gal Chechik, Isaac Meilijson, Eytan Ruppin: E_ective Neuronal Learning with Ine_ective Hebbian Learning Rules, October 26, 2000.
16. Daniel A. Butts, Patrick O. Kanold, Carla J. Shatz: A Burst-Based ‘‘Hebbian’’ Learning Rule at Retinogeniculate Synapses Links Retinal Waves to Activity-Dependent Refinement, March 2007.
17. J. Hertz, A. Krogh, and R. G. Palmer: Introduction to the Theory of Neural Computation, (Addison Wesley, Redwood City, 1991).
18. Einat Klein, Rachel Mislovaty, Ido Kanter, Andreas Ruttor, Wolfgang Kinzel: Synchronization f neural networks by mutual learning and its application to cryptography.
19. Engel and C. Van den Broeck: Statistical Mechanics of Learning, (Cambridge University Press, 2001).