Volume 3, Issue 5, May 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Neural Cryptography for Secret Key Exchange and Encryption with AES

Dr. Ajit Singh* *CSE, SES, BPSMV India.*
Aarti nandal *CSE, SES, BPSMV India.*

Abstract— Cryptography is the art of mangling information into apparent unintelligibility in a manner allowing a secret method of unmangling. The basic service provided by cryptography is the ability to send information between participants in a way that prevents others from reading it. In order to protect the content against an opponent, sender encrypts her message using a fast symmetric encryption algorithm. But now receiver needs to know sender's key for reading her message, one can achieve this by using a key-exchange protocol. Diffie Hellman key exchange protocol was introduced for key exchange protocol. But Diffie Hellman is prune to man in middle attack so neural cryptography is used for key exchange. Secret keys are generated by synchronization of Three Parity machines which is running on both sites. Generated key is used for AES encryption.

Keywords— Cryptography, Secret key, neural network, Three Parity Machine, Synchronization.

I. INTRODUCTION

Cryptography is the art of mangling information into apparent unintelligibility in a manner allowing a secret method of unmangling [1]. The basic service provided by cryptography is the ability to send information between participants in a way that prevents others from reading it. A message in its original form is known as **plaintext** or **cleartext.** The mangled information is known as **ciphertext**. The process for producing ciphertext from plaintext is known as **encryption.** The reverse of encryption is called **decryption.** While cryptographers invert clever secret codes, cryptanalysis attempts to break these codes. Figure 1 illustrates the cryptography process [2].

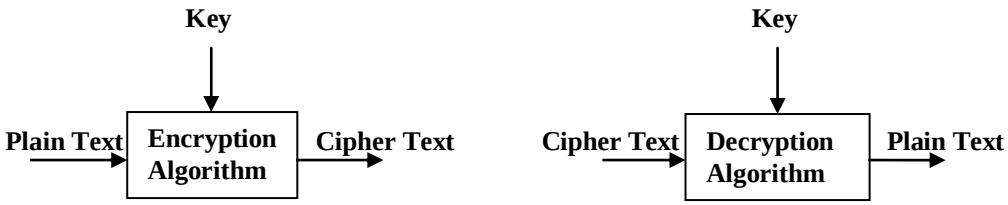

Fig. 1 Cryptography Process

Two partners Alice and Bob want to exchange secret messages over a public channel. In order to protect the content against an opponent Eve, A encrypts her message using a fast symmetric encryption algorithm. But now B needs to know A's key for reading her message. This situation is depicted in figure 2. In fact, there are three possible solutions for this key-exchange problem. First A and B could use a second private channel to transmit the key, e. g. they could meet in person for this purpose. But usually this is very difficult or just impossible. Alternatively, the partners can use public-key cryptography [10].

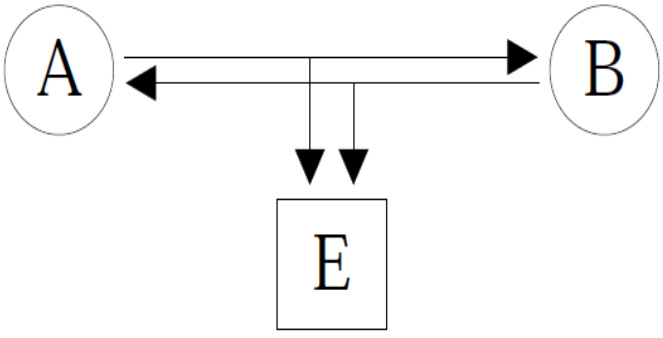

Fig. 2 Attacker obtaining secret message

© 2013, IJARCSSE All Rights Reserved

Here an asymmetric encryption algorithm is employed so that the public keys of A's and B's key pair can be exchanged between the partners without the need to keep them secret. But asymmetric encryption is much slower than symmetric algorithms. That is why it is only used to transmit a symmetric session key. However, one can achieve the same result by using a key-exchange protocol. Diffie Hellman key exchange protocol was introduced for key exchange protocol.

II. DIFFIE HELLMAN KEY EXCHANGE

Diffie Hellman key exchange protocol was introduced by Whitfield Diffie and Martin Hellman in 1976, for key exchange over insecure channel [3].

Here is an explanation which includes the encryption's mathematics: The simplest, and original, implementation of the protocol uses the multiplicative group of integers modulo *p*, where *p* is prime and *g* is primitive root mod *p*. Here is an example of the protocol [4][8]:

TABLE I

DIFFIE HELLMAN KEY EXCHANGE

Alice				**Bob**		
Secret	**Public**	**Calculates**	**sends**	**Calculate**	**Public**	**Secret**
a	p, g		p, g $\longrightarrow$			b
a	p, g, A	$g^a \bmod p = A$	A $\longrightarrow$		p, g	b
a	p, g, A		$\longleftarrow$ ***B***	$g^b \bmod p = B$	p, g, A, B	b
a, s	p, g, A, B	$B^a \bmod p = s$		$A^b \bmod p = s$	p, g, A, B	b, s

A. Man in Middle Attack

The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack. In this attack, an opponent Eve intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Eve substitutes it with his own and sends it to Alice. Eve and Alice thus agree on one shared key and Eve and Bob agree on another shared key. After this exchange, Eve simply decrypts any messages sent out by Alice or Bob, and then reads and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the other party [5].

As an example of what can be done with such an attack, consider the case where Alice and Bob use a shared secret key obtained in a DH protocol for symmetric encryption. Suppose Alice sends a message m to Bob and that ENCK(x) represents the symmetric encryption (e.g. DES) of x using the secret key K.

1. A sends $ENC_g^{xy'}$ (m).
2. E intercepts $ENC_g^{xy'}$ (m) and decrypts it (which he can do since he knows $g^{xy'}$).
3. E replaces this message with $ENC_g^{x'y}$ (m') which he sends to B. Note that m' can be set to any message.

The encryption scheme is thus clearly compromised as message privacy is violated [8].

III. NEURAL CRYPTOGRAPHY

The answer to above problem is to build **neural** networks, one for each [6]. Then, they should synchronize their networks, and the weights will be the secret key Tree Parity Machines, which are used by partners and attackers in neural cryptography, are multi-layer feed-forward networks. Their general structure is shown in figure 3.Two Tree Parity Machines, one for A and one for B respectively, start with random initial weights, which are kept secret [14].

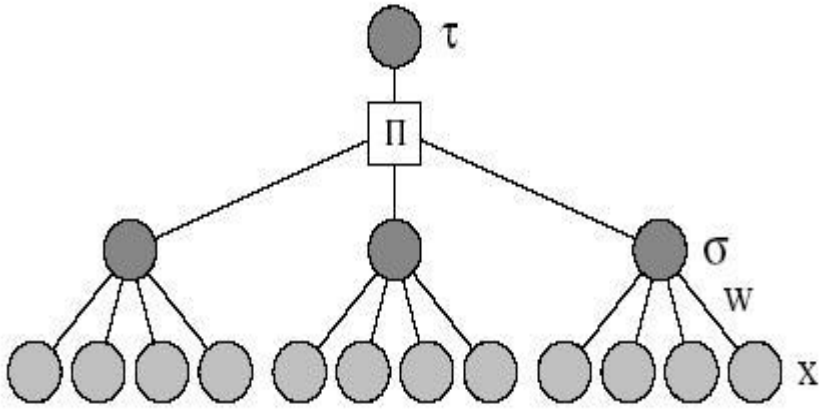

Fig. 3 Feed Forward neural network

© 2013, IJARCSSE All Rights Reserved

Here is a simple neural network developed by Rosenblat in 1968. It consists of an **input vector X**, a **hidden layer Sigma s**, a **weights coefficients W** between the input vector and the hidden layer, and an **activation procedure that** counts the **result value τ**. Let's call such a neural network a **neural machine**. It can be 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. To count the output value, we use a simple method:

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

How do we update the weights? We update the weights only if the output values of the neural machines are equal. There are three different rules [6][12]:

$w_{i,j}^{+} = g(w_{i,j} + x_{i,j}\tau\Theta(\sigma_i\tau)\Theta(\tau^A\tau^B))$	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
$w_{i,j}^{+} = g(w_{i,j} + x_{i,j}\Theta(\sigma_i\tau)\Theta(\tau^A\tau^B))$	Random-walk learning ruleτ

Here, Theta is a special function. Theta(a, b)=0 if a<>b; else Theta=1. The g(...) function keeps the weight in the range -L..+L. x is the input vector and w is the weights vector. After the machines are synchronized, their weights are equal: we can use them for constructing a shared key. In [14], there is a lot of information about the attacks on this algorithm. I just want to say here that it is impossible to hack it.

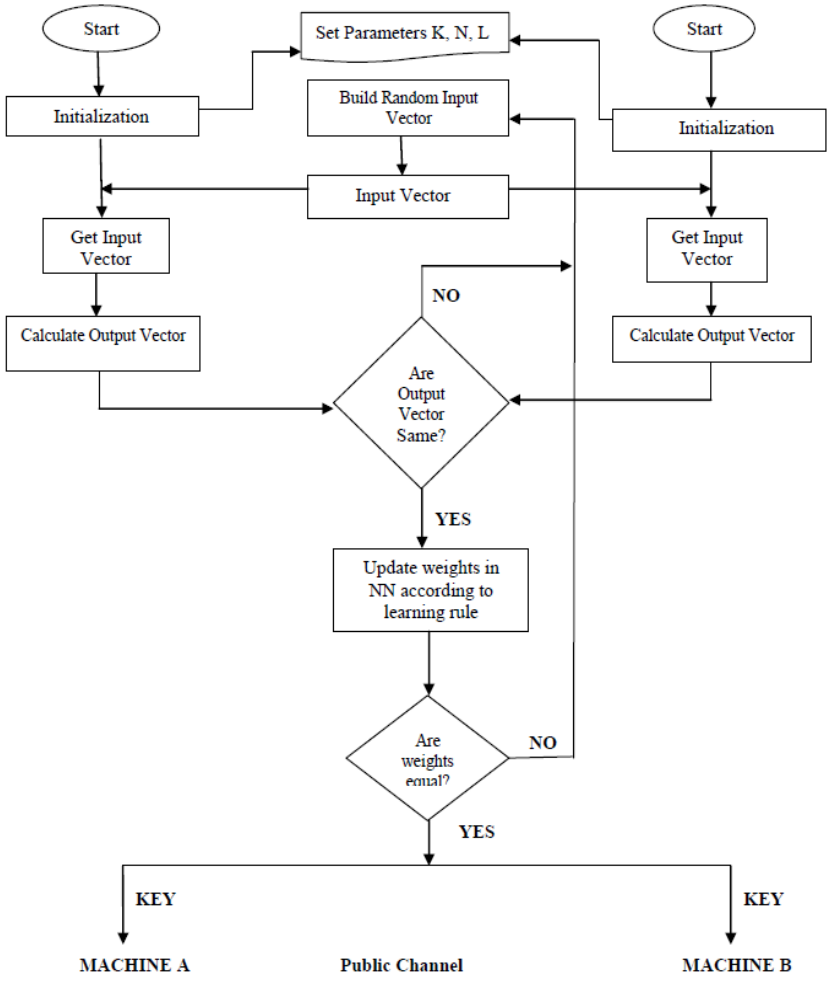

Fig. 4 Neural Key Exchange Algorithm

© 2013, IJARCSSE All Rights Reserved

IV. NEURAL KEY EXCHANGE ALGORITHM

We follow the following steps for neural key generation which is based on neural networks [7]:

1. First of all determine the neural network parameters i.e.: 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. Repeat 4 to 7 until synchronization occurs..
4. The inputs of the hidden units are calculated.
5. The output bit is generated and exchanged between the two machines A and B.
6. If the output vectors of both the machines are same i.e. $\tau_A = \tau_B$ then the corresponding weights are modified using the Hebbian learning rule, Anti-Hebbian learning rule and Random-walk learning rule.
7. After complete synchronization, the synaptic weights are same for both the networks. And these weights are used as secret key.

In this paper I increase the key size without increasing the weight range as a result we get maximum security with less synchronization time. After synchronization we append the auto generated key to increase key size.

A. AES Encryption

This key is utilized to encrypt a sensitive message to be transmitted over an insecure channel using Rijndael AES encryption algorithm with key size 128 bit, 192 bit and 256 bit and block size 128 bit [9][15].

V. IMPLEMENTATION

In this paper Java language is used for implementation of neural key exchange algorithm. In this section we program neural machines and use the Delphi unit neurocrypt.pas [13]. The main object in this unit is **TPM** (tpm - tree parity machine. It contains two 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. Following figures shows that neural server and neural client obtains the same key after synchronization. Both client and server initialize their vector. After obtaining the key client will use AES for encrypting the message with neural key.

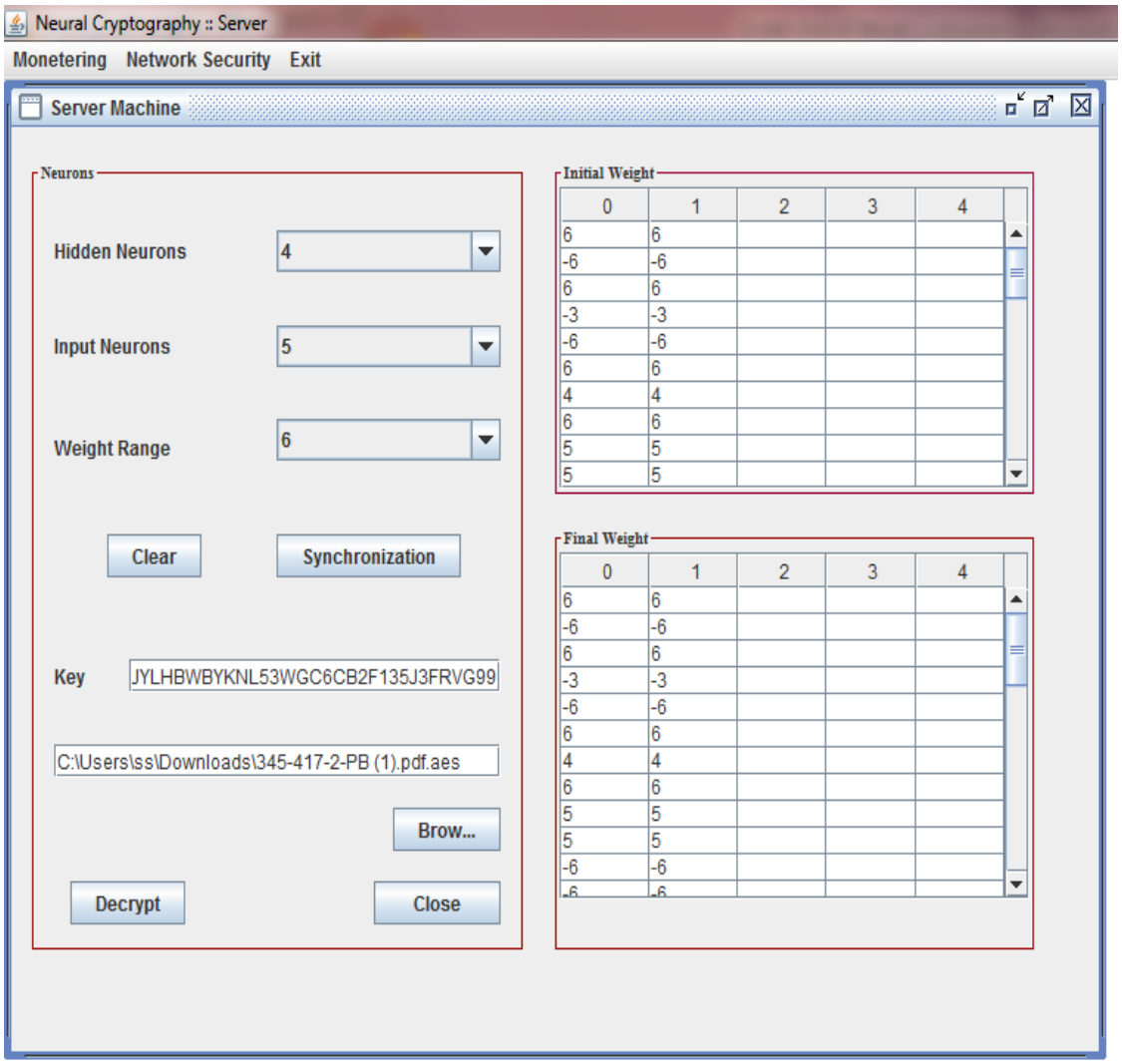

Fig. 5 Neural cryptography Server

© 2013, IJARCSSE All Rights Reserved

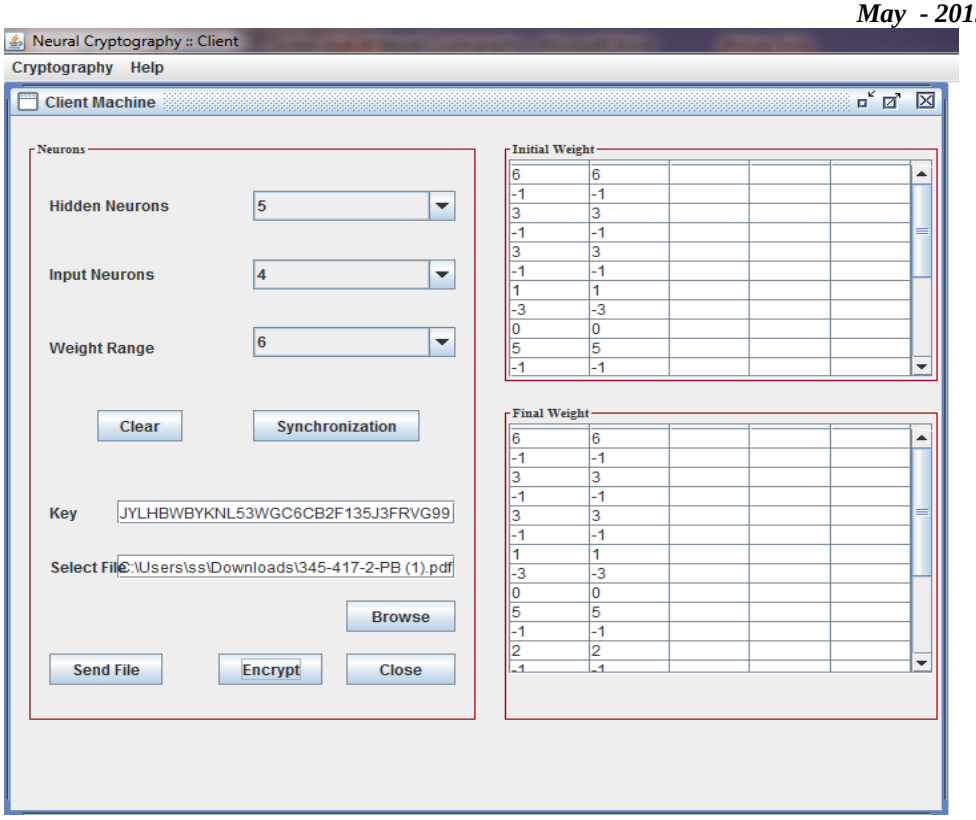

Fig. 6 Neural cryptography client

VI. RESULT

A. Synchronization time

The data set obtained for synchronization time by varying number of input units (n) is shown in figure 7. The number of iterations required for synchronization by varying number of input units (n).

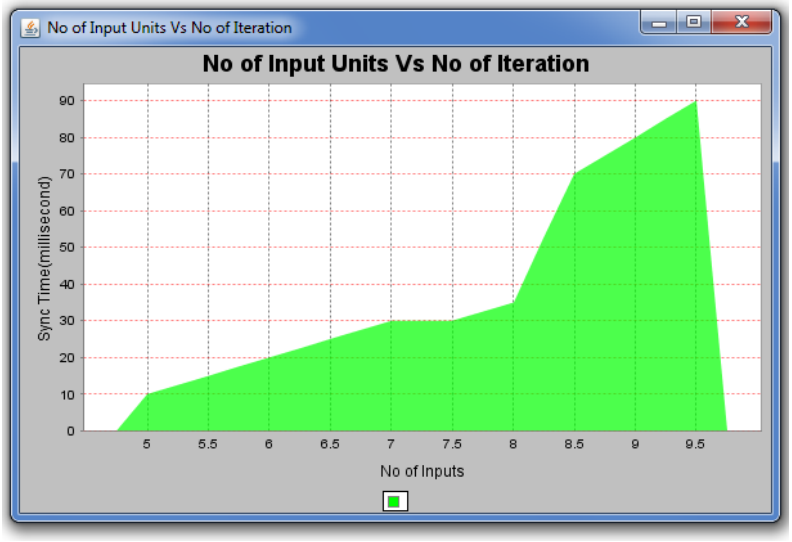

Fig. 7 No of Input Vs Iteration

VII. CONCLUSIONS

The neural key-exchange algorithm is an application of synchronization. Both partners A and B use a Tree Parity Machine with the same structure. The parameters K, L and N are public. Each TPM machine starts with randomly chosen weight vectors. These initial weight conditions are kept secret. During the synchronization process, only the input vectors and the output vectors are transmitted over the public channel. Therefore each participant just knows the internal representation of his own Tree Parity Machine. Keeping this information secret is essential for the security of the key-exchange protocol. After achieving full synchronization, A and B use the modified weight vectors as common secret key.

The main problem of the attacker E is that the internal representation of A's and B's Tree Parity Machines are not known to her. As the movement of the weights depends on σ_i it is important for a successful attack to guess the state of the hidden units correctly. Of course, most known attacks use this approach. But there are other possibilities and it is indeed possible that a clever attack method will be found, which breaks the security of neural cryptography completely. However, this risk exists for all cryptographic algorithms except the one-time pad.

© 2013, IJARCSSE All Rights Reserved

ACKNOWLEDGMENT

Author would like to give her sincere gratitude to her guide Mr. Ajit Singh who encouraged and guided her throughout this research.

REFERENCES

[1] A. Forouzan., "Cryptography *and Network Security* ", First Edition. McGraw-Hill, (2007), USA.
[2] Atul Kahate (2009), *Cryptography and Network Security*, second edition, McGraw-Hill.
[3] William Stalling, "Network Security Essentials (Applications and Standards)", Pearson Education, 2004.
[4] Maryam Ahmed, Baharan Sanjabi, Difo Aldiaz, Amirhossein Rezaei, Habeeb Omotunde "Diffie-Hellman and Its Application in Security Protocols" *International Journal of Engineering Science and Innovative Technology (IJESIT)* Volume 1, Issue 2, November 2012
[5] Jean-Fran¸cois Raymond1 and Anton Stiglic2, "Security Issues in the Diffie-Hellman Key Agreement Protocol", 2009.
[6] Vidushi Sharma, Sachin Rai, Anurag Dev, " A Comprehensive Study of Artificial Neural Networks", *International Journal of Advanced Research in Computer Science and Software Engineering 2 (10),* October-2012, pp.278-284
[7] R. M.Jogdand1 and Sahana S.Bisalapur2, "Design of an Efficient Neural Key Generation," *International Journal of Artificial Intelligence & Applications (IJAIA)*, Vol.2, No.1, pp. 60–69, January 2011.
[8] Maryam Ahmed, Baharan Sanjabi, Difo Aldiaz, Amirhossein Rezaei, Habeeb Omotunde "Diffie-Hellman and Its Application in Security Protocols" *International Journal of Engineering Science and Innovative Technology (IJESIT)* Volume 1, Issue 2, November 2012.
[9] Andreas Ruttor, "Neural Synchronization and Cryptography ", PhD thesis, Bayerische Julius MaximilianUniversity Wurzburg, 2006.
[10] Aseem jagadev, "Advanced Encryption Standard (AES) Implementation", M.Tech Thesis National Institute of Technology, Rourkela. May, 2009.
[11] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography, CRC Press, New York, 1997 p. 81-83.
[12] www.codeproject.com/Articles/39067/Neural-Cryptography .
[13] github.com/sagunms/NeuroCrypto.
[14] Neural_cryptography , en.wikipedia.org/wiki/ .
[15] Advanced_Encryption_Standard, en.wikipedia.org/wiki/.

© 2013, IJARCSSE All Rights Reserved