Cryptanalysis of Key Exchange Method in Wireless Communication

Arindam Sarkar, Jyotsna Kumar Mandal
(Corresponding author: Arindam Sarkar)
Department of Computer Science & Engineering, University of Kalyani, Kalyani-741235, W.B, India
(Email: arindam.vb@gmail.com)
(Received Oct. 5, 2012; revised and accepted Jan. 10 & July 24, 2014)

Abstract

In this paper, a cryptanalysis of key exchange method using multilayer perceptron (CKE) has been proposed in wireless communication of data/information. In this proposed CKE technique both sender and receiver uses an identical multilayer perceptrons for synchronization between them. After achieving the full synchronization weights vectors of both the parties' becomes identical and this identical weight vector is used as a secret session key for encryption/decryption. Different types of possible attacks during synchronization phase are introduced in this paper. Among different types of attacks some of them can be easily prevented by increasing the synaptic depth L. But few attacks are also there which has a great success rate. Parametric tests have been done and results are compared with some existing classical techniques, which show comparable results for the proposed technique.

Keywords: Cryptanalysis, encryption, wireless communication

1 Introduction

Cryptanalysis is the technique through which procedure of breaking the security can be analysed. Eavesdroppers can be reside anywhere in the network and always try to attack on the communication. In recent times wide ranges of techniques are developed to protect data and information from eavesdroppers [4, 6, 7, 8, 9, 10, 11, 14, 15]. These algorithms have their virtue and shortcomings. For Example in DES, AES algorithms [4] the cipher block length is nonflexible. In NSKTE [6], NWSKE [7], AGKNE [8], ANNRPMS [9] and ANNRBLC [10] technique uses two neural network one for sender and another for receiver having one hidden layer for producing synchronized weight vector for key generation. Now attacker can get an idea about sender and receiver's neural machine because for each session architecture of neural machine is static. In NNSKECC algorithm [11] any intermediate blocks throughout its cycle taken as the encrypted block and this number of iterations acts as secret key. Here if n number of iterations are needed for cycle formation and if intermediate block is chosen as an encrypted block after n/2th iteration then exactly same number of iterations i.e. n/2 are needed for decode the block which makes easier the attackers life. In this paper CKE technique has been proposed to analyzed variety of attacks that can be possible in key generation phase using multilayer perceptron and also provides some way out from these attacks.

The organization of this paper is as follows. Section 2 of the paper deals with structure of multilayer perceptron. Different types of attacks in CKE have been discussed in Section 3. Complexity analysis of the technique is given in Section 4. Experimental results are described in Section 5. Analysis of the results presented in Section 6. Analysis regarding various aspects of the technique has been presented in Section 7. Conclusions and future scope are drawn in Section 8 and that of references at end.

2 Structure of Multilayer Perceptron

In multilayer perceptron synchronization scheme secret session key is not physically get exchanged over public insecure channel. At end of neural weight synchronization strategy of both parties' generates identical weight vectors and activated hidden layer outputs for both the parties become identical. This identical output of hidden layer for both parties can be use as one time secret session key for secured data exchange. A multilayer perceptron synaptic simulated weight based undisclosed key generation is carried out between recipient and sender. Figure 1 shows multilayer perceptron based synaptic simulation system. Sender and receivers multilayer perceptron select same single hidden layer among multiple hidden layers for a particular session. For that session all other hidden layers goes in deactivated mode means hidden (processing) units of other layers do nothing with the incoming input. Either synchronized identical weight vector of sender and receivers' input layer, activated hidden

layer and output layer becomes session key or session key can be form using identical output of hidden units of activated hidden layer. The key generation technique and analysis of the technique using random number of nodes (neurons) and the corresponding algorithm is discussed in Subsections 2.1 to 2.5 in details.

Sender and receiver multilayer perceptron in each session acts as a single layer network with dynamically chosen one activated hidden layer and K no. of hidden neurons, N no. of input neurons having binary input vector, , discrete weights, are generated from input to output, are lies between -L and +L, where $i = 1, \cdots, K$ denotes the ith hidden unit of the perceptron and $j = 1, \cdots, N$ the elements of the vector and one output neuron. Output of the hidden units is calculated by the weighted sum over the current input values. So, the state of the each hidden neurons is expressed using Equation (1).

$$h_i = \frac{1}{\sqrt{N}} w_i x_i = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} w_{i,j} x_{i,j}. \quad (1)$$

Output of the ith hidden unit is defined in Equation (2).

$$\sigma_i = sgn(h_i). \quad (2)$$

But in case of $h_i = 0$ then $\sigma_i = -1$ to produce a binary output. Hence $\sigma_i = +1$, if the weighted sum over its inputs is positive, or else it is inactive, $\sigma_i = -1$. The total output of a perceptron is the product of the hidden units expressed in Equation (3).

$$\tau = \Pi_{i=1}^{K} \sigma_i. \quad (3)$$

The learning mechanism proceeds as follows ([8, 9]):

1) If the output bits are different, $\tau A \neq \tau B$, nothing is changed.

2) If $\tau A = \tau B = \tau$, only the weights of the hidden units with $\sigma_k^{A/B} = \tau^{A/B}$ will be will be updated.

3) The weight vector of this hidden unit is adjusted using any of the following learning rules:
Anti-Hebbian:

$$W_k^{A/B} = W_k^{A/B} - \tau^{A/B} x_k \theta(\sigma_k \tau^{A/B})(\tau^A \tau^B). \quad (4)$$

Hebbian:

$$W_k^{A/B} = W_k^{A/B} + \tau^{A/B} x_k \theta(\sigma_k \tau^{A/B})(\tau^A \tau^B).$$

Random walk:

$$W_k^{A/B} = W_k^{A/B} + x_k \theta(\sigma_k \tau^{A/B})(\tau^A \tau^B).$$

During Step (2), if there is at least one common hidden unit with $\sigma k = \tau$ in the two networks, then there are 3 possibilities that characterize the behavior of the hidden nodes:

1) An attractive move: if hidden units at similar k positions have equal output bits, $\sigma_k^A = \sigma_k^B = \tau^{A/B}$.

2) A repulsive move: if hidden units at similar k positions have unequal output bits, $\sigma_k^A \neq \sigma_k^B$.

3) No move: when $\sigma_k^A = \sigma_k^B \neq \tau^{A/B}$, the distance between hidden units can be defined by their mutual overlap,

$$\rho_k = \frac{w_k^A w_k^B}{\sqrt{w_k^A w_k^A}\sqrt{w_k^B w_k^B}}$$

where $0 < \rho k < 1$, with $\rho k = 0$ at the start of learning and $\rho k = 1$ when synchronization occurs with the two hidden units having a common weight vector.

2.1 Multilayer Perceptron Simulation Algorithm

Input: Random weights, input vectors for both multilayer perceptrons.

Output: Secret key through synchronization of input and output neurons as vectors.

Method:

Step 1. Initialization of random weight values of synaptic links between input layer and randomly selected activated hidden layer.

$$w_{i,j} \in \{-L, -L+1, \cdots, +L\}. \quad (5)$$

Step 2. Repeat Steps 3 to 6 until the full synchronization is achieved, using Hebbian-learning rules.

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

Step 3. Generate random input vector X. Inputs are generated by a third party or one of the communicating parties.

Step 4. Compute the values of the activated hidden neurons of activated hidden layer using Equation (6).

$$h_i = \frac{1}{\sqrt{N}} w_i x_i = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} w_{i,j} x_{i,j}. \quad (6)$$

Step 5. Compute the value of the output neuron using

$$\tau = \Pi_{i=1}^{K} \sigma_i.$$

Compare the output values of both multilayer perceptron by exchanging the system outputs.

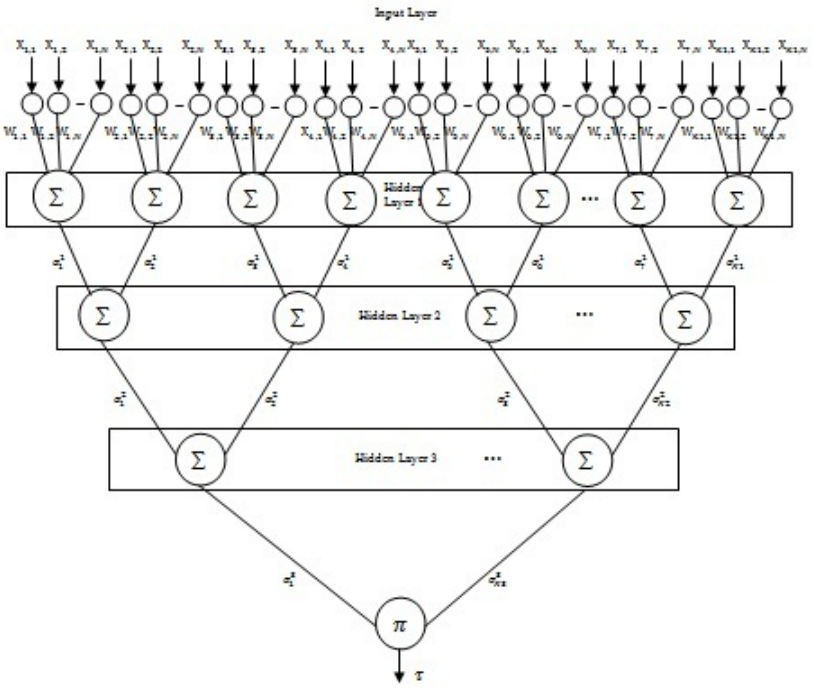

Figure 1: A multilayer perceptron with 3 hidden layers

if Output (A) ≠ Output (B), Go to Step 3

else if Output (A) = Output (B) then one of the suitable learning rule is applied

only the hidden units are trained which have an output bit identical to the common output.

Update the weights only if the final output values of the perceptron are equivalent. When synchronization is finally achieved, the synaptic weights are identical for both the system.

2.2 Multilayer Perceptron Learning Rule

At the beginning of the synchronization process multilayer perceptron of A and B start with uncorrelated weight vectors. For each time step K, public input vectors are generated randomly and the corresponding output bits A/B are calculated. Afterwards A and B communicate their output bits to each other. If they disagree, $A \neq B$, the weights are not changed. Otherwise learning rules suitable for synchronization is applied. In the case of the Hebbian learning rule [12] both neural networks learn from each other.

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

The learning rules used for synchronizing multilayer perceptron share a common structure. That is why they can be described by a single Equation (4).

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

with a function $f(\sigma_i, \tau^A, \tau^B)$, which can take the values -1, 0, or +1. In the case of bidirectional interaction it is given by

$$f(\sigma_i, \tau^A, \tau^B) = \theta(\sigma\tau^A)\theta(\tau^A\tau^B)\begin{cases}\sigma, & \text{Hebbian learning}\\ -\sigma & \text{anti-Hebbian learning}\\ 1 & \text{Random walk learning}\end{cases}$$

The common part $\theta(\sigma\tau^A)\theta(\tau^A\tau^B)$ of $f(\sigma_i, \tau^A, \tau^B)$ controls, when the weight vector of a hidden unit is adjusted. Because it is responsible for the occurrence of attractive and repulsive steps [8].

The equation consists of two parts:

1) $\theta(\sigma\tau^A)\theta(\tau^A\tau^B)$: This part is common between the three learning rules and it is responsible for the attractive and repulsive effect and controls when the weight vectors of a hidden unit is updated. Therefore, all three learning rules have similar effect on the overlap.

2) $(\sigma, -\sigma, 1)$: This part differs among the three learning rules and it is responsible for the direction of the weights movement in the space. Therefore, it changes the distribution of the weights in the case of Hebbian and anti-Hebbian learning. For the Hebbian rule, A's ad B's multilayer perceptron learn their own output and the weights are pushed towards the boundaries at $-L$ and $+L$. In contrast, by using the anti- Hebbian rule, sender's and receiver's multilayer perceptron learn the opposite of their own outputs.

Consequently, the weights are pulled from the boundaries $\pm L$. The random walk rule is the only rule that does not affect the weight distribution so they stay uniformly distributed. In fact, at large values of N, both Hebbian and anti-Hebbian rules do not affect the weight distribution. Therefore, the proposed algorithm is restricted to use either random walk learning rule or Hebian or anti-Hebbian learning rules only at large values of N. The random walk learning rule is chosen since it does not affect the weights distribution regardless of the value of N.

2.3 Weight Distribution of Multilayer Perceptron

In case of the Hebbian rule Equation (5), A's and B's multilayer perceptron learn their own output. Therefore the direction in which the weight $w_{i,j}$ moves is determined by the product $\sigma_i x_{i,j}$. As the output σ_i is a function of all input values, $x_{i,j}$ and σ_i are correlated random variables. Thus the probabilities to observe $\sigma_i x_{i,j} = +1$ or $\sigma_i x_{i,j} = -1$ are not equal, but depend on the value of the corresponding weight $w_{i,j}$ [2, 3, 5, 13].

$$P(\sigma_i x_{i,j} = 1) = \frac{1}{2}[1 + erf(\frac{e_{i,j}}{\sqrt{NQ_i - w_{i,j}^2}})].$$

According to this equation, $\sigma_i x_{i,j} = sgn(w_{i,j})$ occurs more often than the opposite, $\sigma_i x_{i,j} = -sgn(w_{i,j})$. Consequently, the Hebbian learning rule pushes the weights towards the boundaries at $-L$ and $+L$. In order to quantify this effect the stationary probability distribution of the weights for $t \to \infty$ is calculated for the transition probabilities. This leads to [13].

$$P(w_{i,j} = w) = P_0 \prod_{m=1}^{|w|} \frac{1 + erf[\frac{m-1}{\sqrt{NQ_i-(m-1)^2}}]}{1 - erf[\frac{m}{\sqrt{NQ_i-m^2}}]}.$$

Here the normalization constant P_0 is given in Equation (7), the constant should be expressed as

$$P_0 = [\sum_{w=-L}^{L} \prod_{m=1}^{|w|} \frac{1 + erf[\frac{m-1}{\sqrt{NQ_i-(m-1)^2}}]}{1 - erf[\frac{m}{\sqrt{NQ_i-m^2}}]}]^{-1}. \quad (7)$$

In the limit $N \to \infty$ the argument of the error functions vanishes, so that the weights stay uniformly distributed. In this case the initial length of the weight vectors is not changed by the process of synchronization.

$$\sqrt{Q_i(t=0)} = \sqrt{\frac{L(L+1)}{3}}.$$

But, for finite N, the probability distribution itself depends on the order parameter Q_i. Therefore its expectation value is given by the solution of the following equation:

$$Q_i = \sum_{w=-L}^{L} w^2 P(w_{i,j} = w).$$

2.4 Order Parameters

In order to describe the correlations between two multilayer perceptron caused by the synchronization process, one can look at the probability distribution of the weight values in each hidden unit. It is given by $(2L + 1)$ variables.

$$P_{a,b}^i = P(w_{i,j}^A = a \wedge w_{i,j}^B = b)$$

which are defined as the probability to find a weight with $w_{i,j}^A = a$ in A's multilayer perceptron and $w_{i,j}^B = b$ in B's multilayer perceptron. In both cases, simulation and iterative calculation, the standard order parameters, which are also used for the analysis of online learning, can be calculated as functions of $P_{a,b}^i$ [1].

$$\begin{aligned} Q_i^A &= \frac{1}{N} w_i^A w_i^A = \sum_{a=-L}^{L} \sum_{b=-L}^{L} a^2 P_{a,b}^i \\ Q_i^B &= \frac{1}{N} w_i^B w_i^B = \sum_{a=-L}^{L} \sum_{b=-L}^{L} b^2 P_{a,b}^i \\ R_i^{AB} &= \frac{1}{N} w_i^A w_i^B = \sum_{a=-L}^{L} \sum_{b=-L}^{L} ab P_{a,b}^i \end{aligned}$$

Then the level of synchronization is given by the normalized overlap between two corresponding hidden units

$$\rho_i^{AB} = \frac{w_i^A w_i^B}{\sqrt{w_i^A w_i^A}\sqrt{w_i^B w_i^B}} = \frac{R_i^{AB}}{\sqrt{Q_i^A Q_i^B}}.$$

2.5 Hidden Layer as a Secret Session Key

At end of full weight synchronization process, weight vectors between input layer and activated hidden layer of both multilayer perceptron systems become identical. Activated hidden layer's output of source multilayer perceptron is used to construct the secret session key. This session key is not get transmitted over public channel because receiver multilayer perceptron has same identical activated hidden layer's output. Compute the values of the each hidden unit by

$$\begin{aligned} \sigma_i &= sgn(\sum_{j=1}^{N} w_{i,j} x_{i,j}) \\ sgn(x) &= \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases} \end{aligned} \quad (8)$$

For example consider 8 hidden units of activated hidden layer having absolute value $(1, 0, 0, 1, 0, 1, 0, 1)$ becomes an 8 bit block. This 10010101 become a secret session key for a particular session and cascaded XORed with recursive replacement encrypted text. Now final session key based encrypted text is transmitted to the receiver end. Receiver has the identical session key i.e. the output of the hidden units of activated hidden layer of

receiver. This session key used to get the recursive replacement encrypted text from the final cipher text. In the next session both the machines started tuning again to produce another session key. Identical weight vector derived from synaptic link between input and activated hidden layer of both multilayer perceptron can also becomes secret session key for a particular session after full weight synchronization is achieved.

3 Different Types of Attacks on Multilayer Perceptron

The security of multilayer perceptron based key generation protocol is based on a contest between attractive and repulsive forces. Two multilayer perceptrons interacting with each other synchronize much faster than an attacker network only trained with their inputs and outputs. The dissimilarity between the two parties and the attacker is that the two parties synchronize in a polynomial time of synaptic depth L, while the complexity of the attacker scales exponentially. However, the process is stochastic and depends on the random attractive and repulsive forces. As a result, there is a small probability that an attacker succeeds to synchronize with one of the parties. The difficulty an attacker faces with the organization of multilayer perceptron is the lack of information about the internal representation of A's or B's machine. Most of attacks depend on estimating the state of the hidden units. Following are the different possible attacks on multilayer perceptron during key generation phase.

3.1 Type-1 Attack

In this type1attack replicate a huge population of multilayer perceptrons with the identical arrangement as the two parties, and teach them with the same inputs. At each stage about half the replicated networks produces an output of $+1$, and half produces an output of -1. Successful multilayer perceptrons whose outputs imitate those of the two parties raise and multiply, while unsuccessful multilayer perceptrons gets ruled out. Attack starts with one network with haphazardly chosen weights. At each step a population of networks grow according to 3 potential scenarios:

- A and B have dissimilar outputs $\tau A \neq \tau B$, and therefore do not change their weights. Then all the attacker's networks stay unaffected as well.

- A and B have the equivalent outputs $\tau A = \tau B$, and the sum of attacking networks is lesser than some predefined limit. In this case there are 4 possible combinations of the hidden outputs agreeing with the final output. So, the attacker replaces each network N from the population by 4 variants of itself, $\{N1, ..., N4\}$ which are the results of updating N with the standard learning rule but pretending that the hidden outputs were equal to each one of these combinations.

- A and B have the identical outputs $\tau A = \tau B$ but the total number of simulated networks is larger than predefined value. In this case the attacker computes the outputs of all the networks, deletes the unsuccessful networks whose output is different from τA, and updates the weights in the successful networks by using the standard learning rule with the actual hidden outputs of the perceptrons.

3.2 Type-2 Attack

In Type-2 attack the attacker imitates one of the parties, but if attacker output disagrees with the imitated party's output $\tau c \neq \tau A$, attacker certainly knows that either one or all three of his hidden units are mistaken. In order to get $\tau c = \tau A$ attacker negates the sign of one of attacker's hidden units. As $\sigma = sgn(h)$ the unit most likely to be wrong is the one with the minimal $|h|$, therefore that is the unit which is negate. This policy results a immense enhancement in the attacker's achievement. It can be seen that the success rate is quite high for all L values presented, but it drops exponentially as L increases. On the other hand parties' synchronization time increases like L^2, and therefore it can be conclude that in the boundary of large L values the proposed technique is secure against Type-2 attack. Each input can be viewed as K random hyperplanes $(X_1, \cdots, X_K)$ corresponding to K hidden units. Each X_i is a hyperplane $f_i(z_1, \cdots, z_n) = \sum_{j=i}^{N} x_{ij} z_j = 0$ in the N-dimensional discrete space $U = \{-L, \cdots, L\}^N$. The weights of a network could be also viewed as K points $W_1, \cdots, W_K$ in U, $W_i = \{w_{i1}, \cdots, w_{ik}\}$, while the i-th hidden output is just the side of the half-space (with respect to X_i) which contains W_i. Consider an attacking network E that is close enough to the unknown network A but has a different output for a given input. In fact they have either 1 or 3 different hidden outputs. The second case is less likely to occur so we assume that only one hidden output of the network E is different from the corresponding hidden output of A. Consequently, only one pair (W_i^A, W_I^E) is separated by the known input hyperplane X_i. Of course, we are interested in detecting its index i. If the points W_i^E and W_i^A are separated by X_i then the distance between them is greater than the distance from W_i^E to the hyperplane X_i. W_i^E and W_i^A are close to each other, so the distance from W_i^E to X_i has to be small. On the other hand, if W_i^E and W_i^A are in the same half-space with respect to X_i then they are more likely to be far away from the random input X_i (even though we know that they are close to each other). We thus guess that the index of the incorrect hidden output is the i for which W_i^E is closest to the corresponding hyperplane X_i, where we compute the distance by $\rho(W_i^E, X_i) = |f_i(W_i^E)|$. Formally, the attacker constructs a single neural network E with the same structure as A and B, and randomly initializes its

weights. At each step attacker's trains E with the same input as the two parties, and updates its weights with the following rules:

- If A and B have different outputs $\tau A \neq \tau B$, then the attacker doesn't update E.
- If A and B have the same outputs $\tau A = \tau B$ and $\tau E = \tau A$, then the attacker updates E by the usual learning rule.
- If A and B have the same outputs $\tau A = \tau B$ and $\tau E \neq \tau A$, then the attacker finds $i_0 \in \{1, \cdots, K\}$ that minimizes $|\sum_{j=0}^{N} w_{ij}^{E} x_{ij}|$. The attacker negates $\tau_{i_0}^{E}$ and updates E assuming the new hidden bits and output τA.

3.3 Type-3 Attack

In this Type-3 attack a huge collection of M attackers work together. The Type-2 attacker's likelihood to supposition correctly A's interior representation is some function $Pcorrect(\beta)$ of its overlap β with A, starting from $Pcorrect(\beta = 0) = 0.25$. Assume there are group of M independent Type-2 attackers, each having overlap β with A. They will split into 4 groups, one for each possible internal representation. Since $Pcorrect > 0.25$ for all $\beta > 0$, the number of attackers having the correct internal representation, M. $Pcorrect$ will be bigger than the number of attackers in the other 3 groups, for all $\beta > 0$. Therefore, the internal representation resulted from the majority discussion of M independent Type-2 attackers would always be the correct one! From this argument we conclude that the attack should use $M \gg 2k-1$ attackers, which would simultaneously develop an overlap with the parties, trying to remain as independent as possible.

3.4 Type-4 Attack

The Type-4 attack procedure is to start from independent Type-2 attackers and let them act disjointedly for some preliminary number of time steps. Then, the majority procedure is applied: we count how many attackers have each of the 4 possible internal representations, and assign the majority's internal representation to all the M attackers. To prevent the similarity between the attackers from developing too quickly, this majority procedure is applied only on even time steps. However, the attackers make many coherent moves, and unavoidable overlap is developed between them as well. Therefore we do not have a group of independent attackers, but of attackers with an overlap between them. This overlap diminishes the efficiency of the attack, and it is not always successful as a majority attack of M independent attackers would be.

3.5 Type-5 Attack

It is much easier to predict the position of a point in a bounded multidimensional box after several moves in its random walk than to guess its original position. A simple way to do it is to consider each coordinate separately, and to associate with each possible value i in the interval $\{-L, \cdots, L\}$ of the probability $p_t(i) = Pr[X_t = i]$. Initially $\forall i, p_0(i) = \frac{1}{2L+1}$ and after each move $p_{t+1}(i) = \sum_j p_t(j)$, where j are such that if $x_t = j$ then $x_{t+1} = i$. Applying this technique to the original scheme we face the problem that the moves are not known to the attacker does not know which perceptrons are updated in each round. Fortunately, if we know the distribution of the probabilities $P_{k,n,i} = Pr[w_{k,n} = i]$ then using dynamic programming we can calculate the distribution of $\vec{w_k}\vec{x_k}$ for a given vector $\vec{x_k}$ and thus the probabilities $u_k(s) = Pr[\tau_k = s]$. Using these probabilities we can calculate the conditional probabilities

$$\begin{aligned} U_k &= Pr[\tau_k = 1|\tau], \\ &= \frac{\sum_{(\alpha_1,\cdots,\alpha_k):\prod_i \alpha_i = \tau, \alpha_k = 1} \prod_i \mu_i(\alpha_i)}{\sum_{(\alpha_1,\cdots,\alpha_k):\prod_i \alpha_i = \tau} \prod_i \mu_i(\alpha_i)} \end{aligned}$$

because τ is publicly known. We can now update the distribution of the weights: $P_{k,n,i}^{t+1} = \sum_j P_{k,n,j}^{t} Pr[w_{k,n}^{t} = j \Rightarrow w_{k,n}^{t+1} = i]$ is calculated using U_k. Experiments show that in most cases, when A and B converge to a common $\hat{w_{k,n}}$ the probabilities $Pr[w_{k,n} = \hat{w_{k,n}} \approx 1$ and thus the adversary can easily find $\hat{w_{k,n}}$ when A and B decide to stop the protocol.

3.6 Type-6 Attack

To provide a brute force attack, an attacker has to test all possible keys (all possible values of weights). By K hidden neurons, $K \times N$ input neurons and boundary of weights L, this gives $(2L+1)KN$ possibilities. For example, the configuration $K = 3$, $L = 3$ and $N = 100$ gives us 3×10253 key possibilities, making the attack impossible with today's computer power.

3.7 Type-7 Attack

Here the attacker E's neural network has the same structure of A's and B's. All what E has to do is to start with random initial weights and to train with the same inputs transmitted between A and B over the public channel. Then, the attacker E learns the mutual output bit $\tau^{A/B}$ between them and applies the same learning rule by replacing τ^{E} with $\tau^{A/B}$, i.e.

$$W_k^E = W_k^E - \tau^{A/B} x_k \theta(\sigma_k^E \tau^{A/B})(\tau^A \tau^B).$$

One of the basic attacks can be provided by an attacker, who owns the same tree parity machine as the parties A and B. He wants to synchronize his tree parity machine with these two parties. In each step there are three situations possible:

- Output (A) $\neq$ Output (B): None of the parties updates its weights.

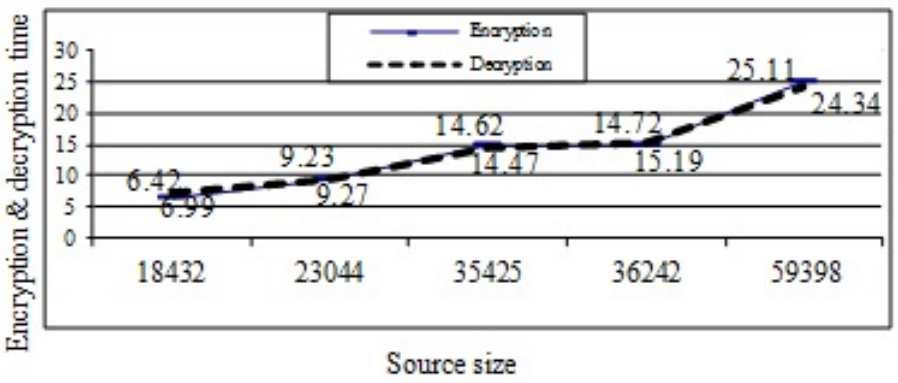

Figure 2: Encryption decryption time against stream size

- Output (A) = Output (B) = Output (E): All the three parties update weights in their tree parity machines.
- Output (A) = Output (B) ≠ Output (E): Parties A and B update their tree parity machines, but the attacker cannot do that. Because of this situation his learning is slower than the synchronization of parties A and B.

It has been proven, that the synchronization of two parties is faster than learning of an attacker. It can be improved by increasing of the synaptic depth L of the neural network. That gives this protocol enough security and an attacker can find out the key only with small probability. Changing this parameter increases the cost of a successful attack exponentially, while the effort for the users grows polynomially. Therefore, breaking the security of neural key exchange belongs to the complexity class NP.

4 Complexity Analysis

The complexity of the Synchronization technique will be $O(L)$, which can be computed using following three steps.

Step 1. To generate a MLP guided key of length N needs $O(N)$ Computational steps. The average synchronization time is almost independent of the size N of the networks, at least up to $N = 1000$. Asymptotically one expects an increase like $O(logN)$.

Step 2. Complexity of the encryption technique is $O(L)$.

Step 2.1. Recursive replacement of bits using prime nonprime recognition encryption process takes $O(L)$.

Step 2.2. MLP based encryption technique takes $O(L)$ amount of time.

Step 3. Complexity of the decryption technique is $O(L)$.

Step 3.1. In MLP based decryption technique, complexity to convert final cipher text into recursive replacement cipher text T takes $O(L)$.

Step 3.2. Transformation of recursive replacement cipher text T into the corresponding stream of bits $S = s_0 s_1 s_2 s_3 s_4 \cdots s_{L-1}$, which is the source block takes $O(L)$ as this step also takes constant amount of time for merging $s_0 s_1 s_2 s_3 s_4 \cdots s_{L-1}$.

5 Experiment Results

In this section the results of implementation of the proposed CKE encryption/decryption technique has been presented in terms of encryption decryption time, Chi-Square test, source file size vs. encryption time along with source file size vs. encrypted file size.

The results are also compared with existing RSA [4] technique, existing ANNRBLC [10] and NNSKECC [11].

Table 1: Encryption/decryption time vs. file size

Encryption Time (s)			Decryption Time (s)		
Source Size (bytes)	CKE	NNSK-ECC [9]	Encrypted Size(bytes)	CKE	NNSK-ECC [9]
18432	6.42	7.85	18432	6.99	7.81
23044	9.23	10.32	23040	9.27	9.92
35425	14.62	15.21	35425	14.47	14.93
36242	14.72	15.34	36242	15.19	15.24
59398	25.11	25.49	59398	24.34	24.95

Table 1 shows encryption and decryption time with respect to the source and encrypted size respectively. It is also observed the alternation of the size on encryption.

In Figure 2 stream size is represented along X axis and encryption/decryption time is represented along Y-axis. This graph is not linear, because of different time requirement for finding appropriate CKE key. It is observed that the decryption time is almost linear, because there is no CKE key generation process during decryption.

Table 2 shows Chi-Square value for different source stream size after applying different encryption algorithms.

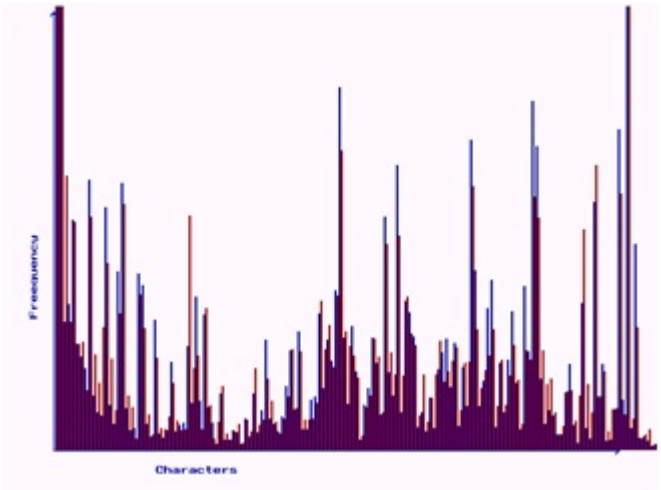

Figure 3: Chi-Square value against stream size

It is seen that the Chi-Square value of CKE is better compared to the algorithm ANNRBLC [10] and comparable to the Chi-Square value of the RSA algorithm. Figure 3 shows graphical representation of Table 2.

Table 2: Source size vs. Chi-Square value

Stream Size (bytes)	Chi-Square value (TDES) [1]	Chi-Square value (CKE)	Chi-Square value (ANNRBLC) [8]	Chi-Square value (RSA) [1]
1500	1228.5803	2856.2673	2471.0724	5623.14
2500	2948.2285	6582.7259	5645.3462	22638.99
3000	3679.0432	7125.2364	6757.8211	12800.355
3250	4228.2119	7091.1931	6994.6198	15097.77
3500	4242.9165	12731.7231	10572.4673	15284.728

Table 3 shows total number of iteration needed and number of data being transferred for CKE key generation process with different numbers of input(N) and activated hidden(H) neurons and varying synaptic depth(L). Figure 4 shows the snapshot of CKE key simulation process.

Table 3: Data exchanged and number of iterations for different parameters value

No. of Input Neurons(N)	No. of Activated Hidden Neurons (K)	Synaptic Weight (L)	Total No. of Iterations	Data Exchanged (Kb)
5	15	3	624	48
30	4	4	848	102
25	5	3	241	30
20	10	3	1390	276
8	15	4	2390	289

6 Analysis of Results

From results obtained it is clear that the technique will achieve optimal performances. Encryption time and decryption time varies almost linearly with respect to the block size. For the algorithm presented, Chi-Square value is very high compared to some existing algorithms. A user input key has to transmit over the public channel all the way to the receiver for performing the decryption procedure. So there is a likelihood of attack at the time of key exchange. To defeat this insecure secret key generation technique a neural network based secret key generation technique has been devised. The security issue of existing algorithm can be improved by using CKE secret session key generation technique. In this case, the two partners A and B do not have to share a common secret but use their indistinguishable weights or output of activated hidden layer as a secret key needed for encryption. The fundamental conception of CKE based key exchange protocol focuses mostly on two key attributes of CKE. Firstly, two nodes coupled over a public channel will synchronize even though each individual network exhibits disorganized behavior. Secondly, an outside network, even if identical to the two communicating networks, will find it exceptionally difficult to synchronize with those parties, those parties are communicating over a public network. An attacker E who knows all the particulars of the algorithm and records through this channel finds it thorny to synchronize with the parties, and hence to calculate the common secret key. Synchronization by mutual learning (A and B) is much quicker than learning by listening (E) [12]. For usual cryptographic systems, we can improve the safety of the protocol by increasing of the key length. In the case of CKE, we improved it by increasing the synaptic depth L of the neural networks.

For a brute force attack using K hidden neurons, $K \times N$ input neurons and boundary of weights L, gives $(2L+1)KN$ possibilities. For example, the configuration

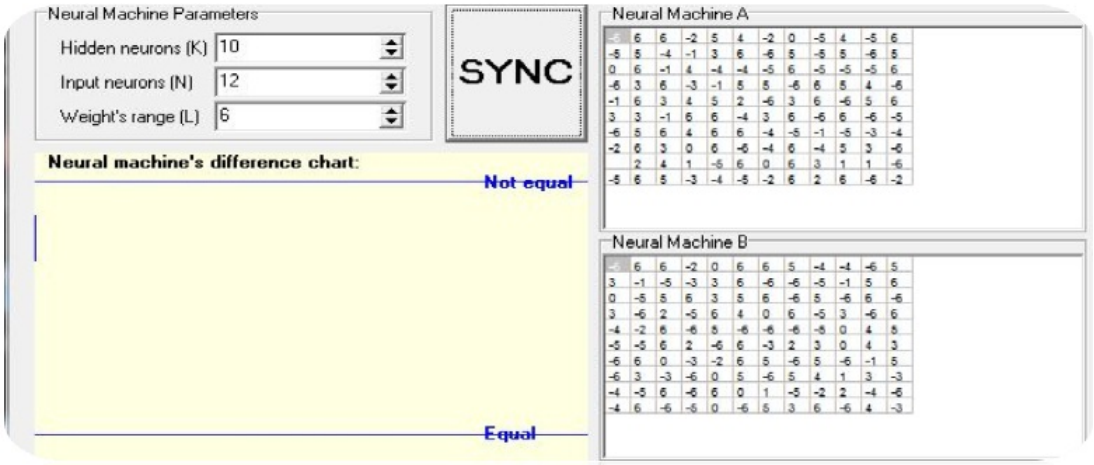

Figure 4: CKE Key Simulation Snapshot with N=12, K=10 and L=6

$K = 3$, $L = 3$ and $N = 100$ gives us 3×10253 key possibilities, making the attack unfeasible with today's computer power. E could start from all of the $(2L + 1)3N$ 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. This is not true for simple perceptron the most unbeaten cryptanalysis has two supplementary ingredients first; a group of attacker is used. Second, E makes extra training steps when A and B are quiet [1, 12, 13]. So increasing synaptic depth L of the CKE we can make our CKE safe.

7 Security Issue

The main difference between the partners and the attacker in CKE is that A and B are able to influence each other by communicating their output bits τ^A and τ^B while E can only listen to these messages. Of course, A and B use their advantage to select suitable input vectors for adjusting the weights which finally leads to different synchronization times for partners and attackers. However, there are more effects, which show that the two-way communication between A and B makes attacking the CKE protocol more difficult than simple learning of examples. These confirm that the security of CKE key generation is based on the bidirectional interaction of the partners. Each partener uses a seperate, but identical pseudo random number generator. As these devices are initialized with a secret seed state shared by A and B. They produce exactly the same sequence of input bits. Whereas attacker does not know this secret seed state. By increasing synaptic depth average synchronize time will be increased by polynomial time. But success probability of attacker will be drop exponentially Synchonization by mutual learning is much faster than learning by adopting to example generated by other network. Unidirectional learning and bidirectional synchronization. As E can't influence A and B at the time they stop transmit due to synchrnization. Only one weight get changed where, $\sigma_i = T$. So, difficult to find weight for attacker to know the actual weight without knowing internal representation it has to guess.

8 Conclusion

This paper presented a novel approach for cryptanalysis of key exchange using multilayer perceptron. This technique enhances the security features of the key exchange algorithm by increasing of the synaptic depth L of the CKE. Here two partners A and B do not have to exchange a common secret key over a public channel but use their indistinguishable weights or outputs of the activated hidden layer as a secret key needed for encryption or decryption. So likelihood of attack proposed technique is much lesser than the simple key exchange algorithm. Future scope of this technique is that this CKE model can be used in wireless communication and also in key distribution mechanism.

Acknowledgments

The author expresses deep sense of gratitude to the Department of Science & Technology (DST) , Govt. of India, for financial assistance through INSPIRE Fellowship leading for a PhD work under which this work has been carried out, at the department of Computer Science & Engineering, University of Kalyani.

References

[1] A. Engel and C. Van den Broeck, *Statistical Mechanics of Learning*, Cambridge University Press, Cambridge, 2001.

[2] T. Godhavari, N. R. Alainelu and R. Soundararajan, "Cryptography using neural network," in *IEEE Indicon'05*, Chennai, India, pp. 258–261, Dec. 2005.

[3] D. Hu, "A new service based computing security model with neural cryptography," in *Second Pacific-Asia Conference on Web Mining and Web-based Application (WMWA'09)*, pp. 154–156, 2009.

[4] A. Kahate, *Cryptography and Network Security*, Tata McGraw-Hill, 2006.

[5] W. Kinzel and L. Kanter, "Interacting neural networks and cryptography," in *Advances in Solid State Physics*, vol. 42, pp. 383, 2002.

[6] J. K. Mandal, A. Sarkar, "Multilayer perceptron guided key generation through mutation with recursive replacement in wireless communication," *International Journal on AdHoc Networking Systems*, vol. 2, no. 3, pp. 11-28, 2012.

[7] J. K. Mandal, A. Sarkar, "Neural weight session key based encryption for online wireless communication," in *Research and Higher Education in Computer Science and Information Technology (RHECSIT'12)*, pp. 90–95, Feb. 2012.

[8] J. K. Mandal, A. Sarkar, "An adaptive genetic key based neural encryption for online wireless communication," in *International Conference on Recent Trends In Information Systems (RETIS'11)*, pp. 62–67, Dec. 2011.

[9] J. K. Mandal, A. Sarkar, "An adaptive neural network guided secret key based encryption through recursive positional modulo-2 substitution for online wireless communication," in em International Conference on Recent Trends In Information Technology (ICRTIT'11), pp. 107–112, 2011.

[10] J. K. Mandal, A. Sarkar, "An adaptive neural network guided random block length based cryptosystem," in *2nd International Conference on Wireless Communications, Vehicular Technology, Information Theory And Aerospace & Electronic System Technology*, pp. 1–5, Feb. 2011.

[11] J. K. Mandal, A. Sarkar, "Neural network guided secret key based encryption through cascading chaining of recursive positional substitution of prime nonprime," in *International Confference on Computing and Systems (ICCS'10)*, pp. 291–297, Nov. 2010.

[12] R. Mislovaty, Y. Perchenok, I. Kanter, and W. Kinzel, "Secure key-exchange protocol with an absence of injective functions," *Physical Review E*, vol. 66, no. 6, Dec. 2002.

[13] A. Ruttor, W. Kinzel, R. Naeh, and I. Kanter, "Genetic attack on neural cryptography," *Physical Review E*, vol. 73, no. 3, pp.1–8, 2006.

[14] A. Sarkar, S. Karforma, J. K. Mandal, "Object oriented modeling of IDEA using GA based efficient key generation for E-governance security (OOMIG)," *International Journal of Distributed and Parallel Systems*, vol. 3, no. 2, pp. 171-183, Mar. 2012.

[15] A. Sarkar, J. K. Mandal, *Artificial Neural Network Guided Secured Communication Techniques: A Practical Approach*, LAP Lambert Academic Publishing, ISBN: 978-3-659-11991-0, 2012.

Arindam Sarkar INSPIRE FELLOW (DST, Govt. of India), MCA (VISVA BHARATI, Santiniketan, University First Class First Rank Holder), M.Tech (CSE, K.U, University First Class First Rank Holder).

Jyotsna Kumar Mandal M. Tech. (Computer Science, University of Calcutta), Ph.D.(Engg., Jadavpur University) in the field of Data Compression and Error Correction Techniques, Professor in Computer Science and Engineering, University of Kalyani, India. Life Member of Computer Society of India since 1992 and life member of cryptology Research Society of India. Dean Faculty of Engineering, Technology & Management, working in the field of Network Security, Steganography, Remote Sensing & GIS Application, Image Processing. 25 years of teaching and research experiences. Eight Scholars awarded Ph.D. and 8 are pursuing.