KEY SWAP OVER AMONG GROUP OF MULTILAYER PERCEPTRONS FOR ENCRYPTION IN WIRELESS COMMUNICATION (KSOGMLPE)

Arindam Sarkar¹ and J. K. Mandal²

¹Department of Computer Science & Engineering, University of Kalyani, W.B, India
arindam.vb@gmail.com

²Department of Computer Science & Engineering, University of Kalyani, W.B, India
jkm.cse@gmail.com

ABSTRACT

In this paper, a key swap over mechanism among group of multilayer perceptrons for encryption/decryption (KSOGMLPE) has been proposed in wireless communication of data/information. Two parties can swap over a common key using synchronization between their own multilayer perceptrons. But the problem crop up when group of N parties desire to swap over a key. Since in this case each communicating party has to synchronize with other for swapping over the key. So, if there are N parties then total number of synchronizations needed before swapping over the actual key is O(N²). KSOGMLPE scheme offers a novel technique in which complete binary tree structure is follows for key swapping over. Using proposed algorithm a set of N parties can be able to share a common key with only O(log₂ N) synchronization. Parametric tests have been done and results are compared with some existing classical techniques, which show comparable results for the proposed technique.

KEYWORDS

Multilayer Perceptron, Encryption, Swap Over, Wireless Communication.

1. INTRODUCTION

The most important hazard for private key cryptography is how to firmly swap over the shared secrets between the parties. As a result, key exchange protocols are mandatory for transferring private keys in a protected manner. The first available key exchange protocol is known as Diffie-Hellman key exchange and it depends on the stiffness of computing discrete logarithms [1]. In recent times wide ranges of techniques are developed to protect data and information from eavesdroppers [1, 2, 3, 4, 5, 6, 7, 8, 9]. These algorithms have their virtue and shortcomings. For Example in DES, AES algorithms [1] the cipher block length is nonflexible. In NSKTE [4], NWSKE [5], AGKNE [6], ANNRPMS [7] and ANNRBLC [8] 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 [9] 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. As the same time as key exchange protocols are developed for exchanging key between two parties, many applications do necessitate the need of swapping over a secret key among group of parties. A lot of proposals have been proposed to accomplish this goal. As Multilayer perceptron synchronization proposal is a fresh addition to the field of cryptography, it does not provide a group key exchange mechanism. To solve these types of problems in this paper we have proposed a key swap over mechanism among cluster of multilayer perceptrons for encryption/decryption (KSOCPE) in wireless communication of data/information. This KSOCPE scheme implements the key swap over algorithm with the help of complete binary tree which make the algorithm scales logarithmically with the number of parties participating in the protocol.

The organization of this paper is as follows. Section 2 of the paper deals with structure of multilayer perceptron. Proposed KSOGMLPE has been discussed in section 3. KSOGMLPE password based certificate generation scheme is given in section 4. Complexity analysis of the technique is given in section 5. Experimental results are described in section 6. Analysis of the results presented in section 7. Analysis regarding various aspects of the technique has been presented in section 8. Conclusions and future scope are drawn in section 9 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 the subsections 2.1 to 2.5 in details.

![Figure 1. A Multilayer Perceptron with 3 Hidden Layers](image-url)
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, \( x_j \in \{-1,+1\} \), discrete weights, are generated from input to output, are lies between -L and +L, \( w_{ij} \in [-L,-L+1,...,+L] \). Where \( i = 1,...,K \) denotes the \( i^{th} \) hidden unit of the perceptron and \( j = 1,...,N \) 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 (eq.1)

\[
h_i = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} w_{ij} x_i = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} w_{ij} x_{ij}
\]  

Output of the \( i^{th} \) hidden unit is defined as

\[
\sigma_i = \text{sng}(h_i)
\]  

But in case of \( h_i = 0 \) then \( \sigma_i = -1 \) to produce a binary output. Hence a, \( \sigma_i = +1 \), 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 (eq. 2)

\[
\tau = \prod_{i=1}^{K} \sigma_i
\]  

The learning mechanism proceeds as follows ([6, 7]):

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 updated.
3. The weight vector of this hidden unit is adjusted using any of the following learning rules:

   - **Anti-Hebbian:**
     \[
     W_k^{A/B} = W_k^{A/B} - \tau^{A/B} x_k \Theta(\sigma_k^{A/B}) (\tau^A \tau^B)
     \]

   - **Hebbian:**
     \[
     W_k^{A/B} = W_k^{A/B} + \tau^{A/B} x_k \Theta(\sigma_k^{A/B}) (\tau^A \tau^B)
     \]

   - **Random walk**
     \[
     W_k^{A/B} = W_k^{A/B} + x_k \Theta(\sigma_k^{A/B}) (\tau^A \tau^B)
     \]

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 behaviour of the hidden nodes:
1. **An attractive move:** if hidden units at similar $k$ positions have equal output bits, $\sigma^A_k = \sigma^B_k = \tau^{A/B}$

2. **A repulsive move:** if hidden units at similar $k$ positions have unequal output bits, $\sigma^A_k \neq \sigma^B_k$

3. **No move:** when $\sigma^A_k = \sigma^B_k \neq \tau^{A/B}$

The distance between hidden units can be defined by their mutual overlap, $\rho_k$,

$$\rho_k = \frac{w^A_kw^B_k}{\sqrt{w^A_kw^A_k}\sqrt{w^B_kw^B_k}}$$  \hspace{1cm} (7)

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:**

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

2. **Step 2.**

   Where, $w_{ij} \in \{-L, -L+1, \ldots, L\}$  \hspace{1cm} (8)

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

   $$w^+_{ij} = g\left(w_{ij} + x_{ij}^\tau \Theta(\sigma^\tau) \Theta(\tau^{A+B})\right)$$  \hspace{1cm} (9)

4. **Step 4.** Generate random input vector $X$. Inputs are generated by a third party or one of the communicating parties.

5. **Step 5.** Compute the values of the activated hidden neurons of activated hidden layer using (eq. 10)

   $$h_i = \frac{1}{\sqrt{N}}w_{ij}x_j = \frac{1}{\sqrt{N}}\sum_{j=1}^{N}w_{ij}x_{ij}$$  \hspace{1cm} (10)

6. **Step 6.** Compute the value of the output neuron using

   $$\tau = \prod_{i=1}^{K} \sigma_i$$  \hspace{1cm} (11)

   Compare the output values of both multilayer perceptron by exchanging the system outputs.
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 \( w_i^A / w_i^B \). For each time step \( K \), public input vectors are generated randomly and the corresponding output bits \( \tau_i^A / \tau_i^B \) are calculated. Afterwards A and B communicate their output bits to each other. If they disagree, \( \tau_i^A ≠ \tau_i^B \), the weights are not changed. Otherwise learning rules suitable for synchronization is applied. In the case of the Hebbian learning rule [10] both neural networks learn from each other.

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

\[
\dot{w}_{ij}^+ = g(w_{ij} + x_j \Theta(\sigma \tau^A \Theta(\tau^A \tau^B)))
\]

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, A’s and B’s multilayer perceptron learn the opposite of their own outputs. Consequently, the weights are pulled from the boundaries \(±L\). 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 (eq. 8), 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 \(\sigma_i\) is a function of all input values, \(x_{i,j}\) and \(\sigma_i\) 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 \(w_{i,j}\) [11, 13, 14, 15, 16].

\[
P(\sigma_i x_{i,j} = 1) = \frac{1}{2} \left[ 1 + \text{erf} \left( \frac{w_{i,j}}{\sqrt{NQ_i - w_{i,j}^2}} \right) \right]
\]  
(15)

According to this equation, \(\sigma_i x_{i,j} = \text{sgn}(w_{i,j})\) occurs more often than the opposite, \(\sigma_i x_{i,j} = -\text{sgn}(w_{i,j})\). 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 [11].

\[
P(\sigma_i x_{i,j} = w_i,j) = \prod_{m=1}^{1} \frac{1 + \text{erf} \left( \frac{m - 1}{\sqrt{NQ_i - (m - 1)^2}} \right)}{1 - \text{erf} \left( \frac{m}{\sqrt{NQ_i - m^2}} \right)}
\]  
(16)

Here the normalization constant \(P_0\) is given by

\[
P_0 = \left( \sum_{w = -L}^{L} \prod_{m=1}^{1} \frac{1 + \text{erf} \left( \frac{m - 1}{\sqrt{NQ_i - (m - 1)^2}} \right)}{1 - \text{erf} \left( \frac{m}{\sqrt{NQ_i - m^2}} \right)} \right)^{-1}
\]  
(17)

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.
\[
\sqrt{Q_i(t=0)} = \sqrt{\frac{L(L+1)}{3}}
\]  
(18)

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

\[
Q = \sum_{w=-L}^{L} w^2 P(w_j = w)
\]  
(19)

### 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.

\[
P_{ab} = \sum_{w=-L}^{L} w^2 P(w_j = a \land w_j = b)
\]  
(20)

which are defined as the probability to find a weight with \( w_j = a \) in A’s multilayer perceptron and \( w_j = b \) 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 \( P_{ab} \) [12].

\[
Q_i^A = \frac{1}{N} w_i^A w_i^A = \sum_{a=-L}^{L} \sum_{b=-L}^{L} a^2 P_{ab}
\]  
(21)

\[
Q_i^B = \frac{1}{N} w_i^B w_i^B = \sum_{a=-L}^{L} \sum_{b=-L}^{L} b^2 P_{ab}
\]  
(22)

\[
R_{i}^{AB} = \frac{1}{N} w_i^A w_i^B = \sum_{a=-L}^{L} \sum_{b=-L}^{L} ab P_{ab}
\]  
(23)

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

\[
\rho_{i}^{AB} = \frac{w_i^A w_i^B}{\sqrt{w_i^A w_i^A w_i^B w_i^B}} = \frac{R_i^{AB}}{\sqrt{Q_i^A Q_i^B}}
\]  
(24)

### 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

$$\sigma_i = \text{sgn} \left( \sum_{j=1}^{N} w_{ij} x_j \right)$$

$$\text{sgn}(x) = \begin{cases} 
-1 & \text{if } x < 0, \\
0 & \text{if } x = 0, \\
1 & \text{if } x > 0.
\end{cases} \quad (25)$$

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. **THE KSOGMLPE TECHNIQUE**

Our proposed group key swap over technique offers two novel procedures for exchanging group key among different multilayer perceptron. Both procedures are based on the structure of complete binary tree. In the multilayer perceptron group key exchange algorithm, $N$ multilayer perceptrons need to synchronize together and they are represented by an $M$ number of leaves of a complete binary tree where $M$ is defined as $M = 2^{\log_2 N}$.

Complete binary tree based proposed procedures are

- a) Complete Binary Tree with Vote (CBTV)
- b) Complete Binary Tree with Exchange (CBTE)

Both procedures have the same end results but their implementations are different.

3.1. **Synchronization using Complete Binary Tree with Vote (CBTV)**

In the CBTV method, the $N$ multilayer perceptrons are represented by the $M$ leaves. For every step $j$ (starting at $j = 1$) of the algorithm the binary tree is divided into $\frac{M}{2^j}$ subtrees each with $2^j$ leaves. Then each pair of leaves sharing the same parent involved in mutual learning. Next, $j$ is incremented and in each subtree, a node is nominated and the mutual learning algorithm is executed by the nominated nodes and the rest if the nodes follow. When the algorithm reaches the root, then it terminates and hence, all the multilayer perceptrons are synchronized and share the same weight vectors.
3.2. Synchronization using Complete Binary Tree with Exchange (CBTE)

In the CBTE method, the mutual learning algorithm is take place between every two parties having the same parent in the binary tree structure. Let the max depth is the depth of the complete binary tree and cur depth is the current depth where the algorithm is functioning. Starting from a cur depth = max depth−1, apply the mutual learning algorithm between each pair of leaves having the same parent. Following the synchronization, one level up is marked (cur depth = cur depth − 1) and a exchange method is applied between the right leaves of both right and left branches for all subtrees in that cur depth. Once the cur depth becomes equal to zero, all leaves will be synchronized together. For sake of simplicity the group of parties will be represented as vector with indices {0, 1, . . . , M−1} Fig.2 shows the scenario of synchronization. Fig.2a shows the preliminary configuration of unsynchronized parties. In Fig.2b, pairs of parties are synchronized together, {(0, 1), (2, 3), (4, 5), (6, 7)}. Then, the exchange operation is performed, {(0, 2), (1, 3), (4, 6), (6, 7)}, and the mutual learning is applied again. This results in synchronization of two groups each with four parties, {(0, 1, 2, 3), (4, 5, 6, 7)}, as shown in Fig.2c. After that, the exchange operation is applied again and the vector takes the form {(0, 4), (1, 5), (2, 6), (3, 7)}. The algorithm terminates when pairs in the new vector apply mutual learning that produces full synchronization between all parties (Fig.2d). The CBTV method needs to transmit the data between the nominated nodes to other nodes in order to be followed. On the other hand, the CBTE algorithm applies the mutual learning algorithm between each pair of nodes separately.

At the same time as the proposed key exchange protocol is scalable; it remains susceptible to active attacks. An attacker can take part in the protocol and synchronize with the group and finally obtain the shared key which endangers the secret communication between the group later. As a result, it is compulsory to build up an authenticated key exchange protocol to permit only certified users to get hold of the mutual secret.
Figure 2. (a) Shows the preliminary configuration of unsynchronized parties. (b) Pairs of parties are synchronized together, \{(0, 1), (2, 3), (4, 5), (6, 7)\}. (c) Synchronization of two groups each with four parties, \{(0, 1, 2, 3), (4, 5, 6, 7)\}. (d) After that, the exchange operation is applied again and the vector takes the form \{(0, 4), (1, 5), (2, 6), (3, 7)\}.

4. KSOGMLPE Certificate Generation

While proposed KSOGMLPE method offers rapid key exchange between groups of users, it is susceptible to malicious attacks where an challenger can participate in the protocol and hence obtains the group top secret key. So, the group requires an authentication certificate to safeguard it against such types of attacks. In order to construct an authentication certificate, KSOGMLPE assumes that the group obtains a secret password which can be used to authenticate the exchange protocol. This password can be mapped to multilayer perceptron guided cryptographic public parameter which can be used as an initial seed for a random number generator which encrypts the output bits $\tau$ in a fashion similar to that was proposed in [12, 13]. Assume a random number generator (RNG), $R_i, R_i^p = \text{Rem}((a^p R_{i-1}^p - c), m)$ with the set $\lambda = \{a, c, m\}$ being the RNG parameters.

![Algorithm 3 - KSOGMLPE Password Authentication Scheme](image)

Require: $n$ is the number of iterations needed to synchronize between two parties and a nonlinear function $F$.

$\text{loop} \quad \text{for each pair}$

Generate random number $R_0$ and publicly exchange between each pair.

Compute $R_i = R_i \oplus \text{password}$

$\text{loop} \quad \text{for } i$

Generate random number $R_i$.

if $F(R_i) > 0$ then

$\tau_{\text{sent}} = -\tau_{\text{computed}}$

end if

if $F(R_i, R_i^p) > 0$ then

$\tau_{\text{used}} = -\tau_{\text{sent}}$

end if

$\text{end loop}$

$\text{end loop}$
5. **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 (log N).

**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 $S = s_0 \ s_1 \ s_2 \ s_3 \ s_4 \ldots \ s_{L-1}$, which is the source block takes O(L) as this step also takes constant amount of time for merging $s_0 \ s_1 \ s_2 \ s_3 \ s_4 \ldots \ s_{L-1}$.

Key exchange algorithm has complexity of logarithmic proportional to the number of the parties need to synchronize together. Because key exchange protocols works on a structure of a complete binary tree. Algorithm works form leaf level to the root i.e. the height of a complete binary tree which is O (log N).

6. **EXPERIMENT RESULTS**

In this section, CBTV is applied between group of parties and some simulation results are presented. For simplicity, the number of communicating parties is taken to be four. i.e., ($M = 4$). Assuming four parties A, B, C and D need to share a common key so they apply the CBTV algorithm. As shown in Fig.3 curve 1, A and B apply the ordinary mutual learning algorithm till they synchronize. At the same time both C and D do the same as shown in curve 2. Then the swapping mechanism is applied and hence, A and C apply the ordinary mutual learning algorithm and the same scenario repeats for B and D. It is evident that curves 3 and 4 are identical which indicates that the four parties have synchronized at common weight vectors. If another party requires to share a key with previously N synchronized parties, it does not need to repeat the entire algorithm again. Instead, the N synchronized parties are dealt with as a single partner and the mutual learning algorithm is applied between an elected party of the N partners and the new party. Then the other ($N-1$) parties apply the learning rules without sending their output bits over the public channel.

![Figure 3. Synchronization between 4 Multilayer Perceptrons.](image)
In this section the results of implementation of the proposed KSOGMLPE 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 [1] technique, existing ANNRLBC [8] and NNSKECC [9].

Table 1. Encryption / decryption time vs. File size

<table>
<thead>
<tr>
<th>Source Size (bytes)</th>
<th>KSOGMLPE</th>
<th>NNSKE CC [9]</th>
<th>Encrypted Size (bytes)</th>
<th>KSOGMLPE</th>
<th>NNSKE CC [9]</th>
</tr>
</thead>
<tbody>
<tr>
<td>18432</td>
<td>6.42</td>
<td>7.85</td>
<td>18432</td>
<td>6.99</td>
<td>7.81</td>
</tr>
<tr>
<td>23044</td>
<td>9.23</td>
<td>10.32</td>
<td>23040</td>
<td>9.27</td>
<td>9.92</td>
</tr>
<tr>
<td>35425</td>
<td>14.62</td>
<td>15.21</td>
<td>35425</td>
<td>14.47</td>
<td>14.93</td>
</tr>
<tr>
<td>36242</td>
<td>14.72</td>
<td>15.34</td>
<td>36242</td>
<td>15.19</td>
<td>15.24</td>
</tr>
<tr>
<td>59398</td>
<td>25.11</td>
<td>25.49</td>
<td>59398</td>
<td>24.34</td>
<td>24.95</td>
</tr>
</tbody>
</table>

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 4 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 KSOGMLPE key. It is observed that the decryption time is almost linear, because there is no KSOGMLPE key generation process during decryption.

![Figure 4. Encryption decryption time against stream size](image)

Table 2 shows Chi-Square value for different source stream size after applying different encryption algorithms. It is seen that the Chi-Square value of KSOGMLPE is better compared to the algorithm ANNRLBC [8] and comparable to the Chi-Square value of the RSA algorithm.

Table 2. Source size vs. Chi-Square value

<table>
<thead>
<tr>
<th>Stream Size (bytes)</th>
<th>Chi-Square value (TDES) [1]</th>
<th>Chi-Square value in (KSOGMLPE)</th>
<th>Chi-Square value (ANNRLBC) [8]</th>
<th>Chi-Square value (RSA) [1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>1228.3803</td>
<td>2856.2673</td>
<td>2471.0724</td>
<td>5623.14</td>
</tr>
<tr>
<td>2500</td>
<td>2948.2255</td>
<td>6582.7259</td>
<td>5645.3462</td>
<td>22638.99</td>
</tr>
<tr>
<td>3000</td>
<td>3679.6432</td>
<td>7125.2364</td>
<td>6757.8211</td>
<td>12800.355</td>
</tr>
<tr>
<td>3250</td>
<td>4228.2119</td>
<td>7091.1931</td>
<td>6994.6198</td>
<td>15097.77</td>
</tr>
<tr>
<td>3500</td>
<td>4242.9165</td>
<td>12731.7231</td>
<td>10572.4673</td>
<td>15284.728</td>
</tr>
</tbody>
</table>

96
Figure 5 shows graphical representation of table 2.

Figure 5. Chi-Square value against stream size

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

Table 3. Data Exchanged and No. of Iterations For Different Parameters Value

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

Following figure 6. Shows the snapshot of KSOGMLPE key simulation process.

Figure 6. KSOGMLPE Key Simulation Snapshot with N=12, K=10 and L=6

7. 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 KSOGMLPE 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 KSOGMLPE based key exchange protocol focuses mostly on two key attributes of KSOGMLPE. Firstly, two nodes coupled over a public channel will synchronize even though each individual network exhibits disorganized behaviour. 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) [10]. For usual cryptographic systems, we can improve the safety of the protocol by increasing of the key length. In the case of KSOGMLPE, we improved it by increasing the synaptic depth L of the neural networks. For a brute force attack using K hidden neurons, K*N input neurons and boundary of weights L, gives \((2L+1)KN\) possibilities. For example, the configuration \(K = 3, L = 3\) and \(N = 100\) gives us \(3*10^{253}\) key possibilities, making the attack unfeasible with today’s computer power. E could start from all of the \((2L+1)\) 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 [10]-[12]. So increasing synaptic depth \(\text{L}\) of the KSOGMLPE we can make our KSOGMLPE safe.

8. Security Issue

The main difference between the partners and the attacker in KSOGMLPE is that A and B are able to influence each other by communicating their output bits \(\tau^A\) & \(\tau^B\) 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 KSOGMLPE protocol more difficult than simple learning of examples. These confirm that the security of KSOGMLPE key generation is based on the bidirectional interaction of the partners. Each partnet 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 Synchronization 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 synchronization. Only one weight get changed where, \(\tau = t\). So, difficult to find weight for attacker to know the actual weight without knowing internal representation it has to guess.

9. Future Scope & Conclusion

This paper presented a novel approach for group key exchange. KSOGMLPE algorithms are proposed as extensions to the ordinary mutual learning algorithm. Also it has been shown that the complexity of the algorithms is logarithmic proportional to the number of the parties need to
synchronize together. This algorithm can be used in many applications such as video and voice conferences. This technique enhances the security features of the key exchange algorithm by increasing of the synaptic depth $L$ of the KSOGMLPE. 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 KSOGMLPE model can be used in wireless communication and also in key distribution mechanism.

ACKNOWLEDGEMENTS

The author expresses deep sense of gratitude to the Department of Science & Technology (DST), Govt. of India, for financial assistance through IN SPIRE 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

2010, Department of Computer Science, The University of Burdwan, Golapbag North, Burdwan – 713104, West Bengal, India.ISBN 93-80813-01-5, 2010


Arindam Sarkar

INSPIRE Fellow (DST, Govt. of India), MCA (VISVA BхАRATI, Santiniketan, University First Class First Rank Holder), M.Tech (CSE, K.U, University First Class First Rank Holder). Total number of publications 13.

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. one submitted and 8 are pursuing. Total number of publications 230.