Asian Journal of Information Technology 7 (7): 300-306, 2008
ISSN: 1682-3915

A New Approach on Neural Cryptography with Dynamic and Spy Units Using Multiple Transfer Functions and Learning Rules

[1]N. Prabakaran, [2]P. Karuppuchamy and [1]P. Vivekanandan
[1]Department of Mathematics, Anna University, Chennai, India
[2]Department of Chemical Engineering, A.C. Technology, Anna University, Chennai, India

Abstract: There is a necessity to secure the message when the exchange of secret information is taken place among the intended users. We can generate a common secret key using neural networks and cryptography. Two neural networks which are trained on their mutual output bits are analyzed using methods of statistical physics. In the proposed TPMs, hidden layer of each output vectors are compared, then updates from hidden unit using Hebbian learning rule, left-dynamic unit using Random walk rule, right-dynamic unit using Anti-Hebbian learning rule, lower layer spy unit and upper layer spy unit with feedback mechanism. Also, we increase the effective number of keys using entropy of the weight distribution against brute-force attack. The genetic attack, geometric attack and majority attack are also explained in this study.

Key words: Neural cryptography, number of keys, feedback synchronization, genetic attack, geometric attack, majority attack

INTRODUCTION

In the field of cryptography, the two partners A and B can be transmitted secret message over the public channel. An attacker E who is able to listen to the communication should not be able to recover the secret message (Kinzel and Kanter, 2002; Kanter and Kinzel, 2002; Kinzel, 2002; Ruttor, 2006).

Before 1976, all cryptographic methods exchanged common secret keys for encryption which were transmitted between A and B over a secret channel not accessible to any attackers.

In 1976, Diffie and Hellman have shown how to generate a secret key over a public channel for exchange of secret message. Recently, it has been shown how to use synchronization of neural network by mutual learning to generate secret keys over public channel and this algorithm is called neural cryptography (Kinzel and Kanter, 2002).

The two neural networks of the same structure as well as same learning rule which is an adjustment process for the graph weights such that they will be synchronized at the end. The synchronization will be in the sense that, from some point, on which they will have the same weights, even if they change their weights continuously in each step.

NEURAL SYNCHRONIZATION

The weight vectors of the 2 neural networks begin with random numbers. The partners A and B receive a common input vector at each time, their outputs are calculated and then communicated. If they agree on the mapping between the current input and the output, their weights are updated according to the learning rule.

A structure of tree parity machine: The TPMs consist of K-hidden units, K-left dynamic units (Prabakaran *et al*., 2008) and K-right dynamic units each of them being a perceptron with an N-dimensional weight vector w (Godhavari *et al*., 2005). The lower layer and upper layer spy units are connected to the inputs, hidden units, left-dynamic units, right dynamic units and output unit.

The lower layer and upper layer spy units receive the input values from the N-input units, K-left dynamic units, K-right dynamic units, K-hidden units and output unit with feedback mechanism.

The structure of this TPM is shown in Fig. 1. The components of the input vectors x are binary,

$$x_{ij} \in \{ -1, +1\}, x_{im} \in \{-1, +1\}, x_{ik} \in \{ -1, +1\} \qquad (1)$$

and the weights are discrete numbers between -L and +L (Ruttor *et al*., 2006):

Corresponding Author: N. Prabakaran, Department of Mathematics, Anna University, Chennai, India

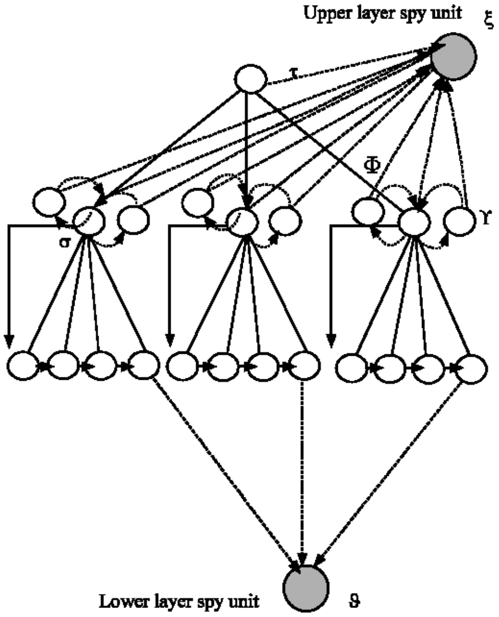

Fig. 1: A structure of tree parity machine with K = 3, δ = 3, Υ = 3, ξ = 1, ϑ = 1 and N = 4

$$w_{ij} \in \{ -L,\ -L+1,\ldots\ldots, L-1,\ L \},\quad w_{im} \in \{ -L, -L+1,\ldots\ldots, L-1, L \},\quad w_{ik} \in \{ -L, -L+1,\ldots\ldots, L-1, L \} \tag{2}$$

where, L is the depths of the weights of the networks (Mislovaty *et al*., 2003).

The index i = 1,….., K denotes the ith hidden unit of TPM (Mislovaty *et al*., 2004), m = 1,….., K left dynamic unit (δ) of the TPM (Prabakaran *et al*., 2008), k = 1,…..,K right dynamic unit (Υ) of the TPM and j =1,….,N denotes the N components (Kanter *et al*., 2002; Mislovaty *et al*., 2002; Kanter *et al*., 2004; Ruttor *et al*., 2004; Metzler *et al*., 2002). The transfer functions are given below

$$\sigma_i = \text{sign}\left(\sum_{j=1}^{N} w_{ij} \bullet x_{ij} \right) \tag{3}$$

$$\delta_i = \tanh\left(\sum_{m=1}^{N} w_{im} \bullet x_{im} \right) \tag{4}$$

$$\Upsilon_i = \tanh\left(\sum_{m=1}^{N} w_{im} \bullet x_{im} \right) \tag{5}$$

$$\vartheta = -\text{sign}\left(\sum_{i=1}^{N} \left[\sum_{j=1}^{N} x_{ij} \right] \right) \tag{6}$$

$$\xi = -\text{sign}\left(\sum_{i=1}^{K} \delta_i\ \sigma_i \Upsilon_i \right) \tag{7}$$

where, Eq. (3) is the transfer function of the hidden unit (Kinzel *et al*., 2000; Kinzer, 1997; Klein *et al*., 2004; Klimov *et al*., 2002), the Eq. (4) is the transfer function of the left dynamic unit (Rutter *et al*., 2002), the Eq. (5) is the transfer function of the right dynamic unit, the Eq. (6) is the transfer function of the lower layer spy unit and the Eq. (7) is the transfer function of the upper layer spy unit.

The K-hidden units of σ_i, left-dynamic units of δ_i and right-dynamic units of Υ_i define a common output bit τ of the total network and is given by:

$$\beta_a = \prod_{i=1}^{K} \sigma_i \tag{8}$$

$$\beta_b = \prod_{i=1}^{K} \delta_i \tag{9}$$

$$\beta_c = \prod_{i=1}^{K} \Upsilon_i \tag{10}$$

where, Eq. (8) is output for the hidden units (Rosen-Zvi *et al*., 2002; Klimov *et al*., 2002), Eq. (9) output for the left-dynamic units and Eq. (10) output for the right-dynamic units.

The 2 TPMs compare the output bits (Engel and Van Den Broeck, 2001) and then update the values between hidden units, left-dynamic units and right-dynamic units as well as 2 parties A and B that are trying to synchronize their weights.

$$\psi_i^{A,B} = \text{comp}(\beta_a, \beta_b, \beta_c) \tag{11}$$

$$\phi_i^{A} = w_{ij}^{A}\ x_{ij}^{A}\ \tau^{B} \psi_i^{A} \tag{12}$$

$$\phi_i^{B} = w_{ij}^{B}\ x_{ij}^{B}\ \tau^{A} \psi_i^{B} \tag{13}$$

where, Eq. (11) represents comparison of the output of hidden, left dynamic and right dynamic units of A and B. The Eq. (12) and (13) represent output of hidden, left and right dynamic units of A and B, respectively.

Learning rules: The partners have their own networks with a same TPM architecture. Each party selects a random initial weight vectors $w_i(A)$ and $w_i(B)$ at t = 0.

The two TPMs are trained by their mutual output bits τ^A and τ^B as well as receive common input vectors x_i and corresponding output bit τ of its partner at each training steps.
The following are the learning rules:

- If the output bits are different, $\tau^A \neq \tau^B$, nothing is changed.
- If $\tau^A = \tau^B \equiv \tau$, the hidden, left and right dynamic units are trained which have an output bit identical to the common output $\phi_i^{A/B} = \tau^{A/B}$.
- To adjust the weights, we consider 3 different learning rules. They are:

(a) Hebbian Learning rule (Kanter *et al.*, 2004) for hidden units

$$\begin{aligned} w_i^A(t+1) &= w_i^A(t)+x_i\tau^A\Theta(\tau^A\phi_i^A)\Theta(\tau^A\tau^B) \\ w_i^B(t+1) &= w_i^B(t)+x_i\tau^B\Theta(\tau^B\phi_i^B)\Theta(\tau^A\tau^B) \end{aligned} \qquad (14)$$

where, Θ is the Heaviside step function (Engel and Van Den Broeck, 2001), if the input is positive then the output is 1 and if input is negative then the function evaluates to 0.

(b) Random walk learning for left dynamic units

$$\begin{aligned} w_i^A(t+1) &= w_i^A(t)+x_i\Theta(\tau^A\phi_i^A)\Theta(\tau^A\tau^B) \\ w_i^B(t+1) &= w_i^B(t)+x_i\Theta(\tau^B\phi_i^B)\Theta(\tau^A\tau^B) \end{aligned} \qquad (15)$$

(c) Anti-Hebbian learning for right dynamic units

$$\begin{aligned} w_i^A(t+1) &= w_i^A(t)-\phi_i x_i\Theta(\tau^A\phi_i^A)\Theta(\tau^A\tau^B) \\ w_i^B(t+1) &= w_i^B(t)-\phi_i x_i\Theta(\tau^B\phi_i^B)\Theta(\tau^A\tau^B) \end{aligned} \qquad (16)$$

Order parameters: The size of a matrix F is (2L+1)×(2L+1) in TPMs (Kanter and Kinzel, 2002). Their elements are $F^i(\mu)$, $F^j(\mu)$ and $F^k(\mu)$ where 'μ' is the state of the machines in the time step, 'i' is hidden units, where 'j' is the left-dynamic units and 'k' is the right-dynamic units. The element f^i_{qr} of matrix stands for the matching components in the ith weight-vector in which the A's components are equal to 'q' and the matching components of B are equal to 'r'. The element f^j_{st}, matching components of jth weight-vector in which the A's components are equal to 's' and the matching components of B are equal to 't'. The element f^k_{uv}, matching components of kth weight-vectors in which the A's components are equal to 'u' and the matching components of B are equal to 'v'. The values of q, r, s, t, u, v are equal to -L, ... -1, 0, 1,....., L. The overlap of the weights belonging to the ith hidden unit (Ruttor *et al.*, 2004), jth left-dynamic unit and kth right-dynamic unit in the 2 parties are given below:

$$R_i^{A,B} = \frac{W_i^A \cdot W_i^B}{N},\ R_j^{A,B} = \frac{W_j^A \cdot W_j^B}{N},\ R_k^{A,B} = \frac{W_k^A \cdot W_k^B}{N} \quad (17)$$

Also their norms

$$Q_i = \frac{W_i^A \cdot W_i^A}{N}, Q_j = \frac{W_j^A \cdot W_j^A}{N} \text{ and } Q_k = \frac{W_k^A \cdot W_k^A}{N}$$

are hidden, left and right-dynamic units of A's TPM, respectively.

$$Q_i = \frac{W_i^B \cdot W_i^B}{N}, Q_j = \frac{W_j^B \cdot W_j^B}{N} \text{ and } Q_k = \frac{W_k^B \cdot W_k^B}{N}$$

are hidden, left and right-dynamic units of B's TPM respectively. They are given by the matrix elements

$$R_i^{A,B} = \sum_{q,r} qr\ f^i_{qr} \qquad (18)$$

$$R_j^{A,B} = \sum_{s,t} st\ f^j_{st} \qquad (19)$$

$$R_k^{A,B} = \sum_{u,v} uv\ f^k_{uv} \qquad (20)$$

The Eq. (18-20) represent overlap between two hidden units, 2 left-dynamic units and two right-dynamic units of A and B, respectively.

$$Q_i^A = \sum_{q,r} q^2 f^i_{qr}, \qquad Q_i^B = \sum_{q,r} r^2 f^i_{qr} \qquad (21)$$

$$Q_j^A = \sum_{s,t} s^2 f^j_{st}, \qquad Q_j^B = \sum_{s,t} s^2 f^j_{st} \qquad (22)$$

$$Q_k^A = \sum_{u,v} u^2 f^k_{uv}, \qquad Q_k^B = \sum_{u,v} v^2 f^k_{uv} \qquad (23)$$

The Eq. (21-23) represent weight distribution of hidden units, left-dynamic units and right-dynamic units of A and B, respectively.

These overlaps and norms fixed the probabilities of deriving the same internal representation via the normalized overlap,

$$\rho_i^{A,B} = \frac{R_i^{A,B}}{\sqrt{Q_i^A Q_i^B}},\ \rho_j^{A,B} = \frac{R_j^{A,B}}{\sqrt{Q_j^A Q_j^B}} \text{ and } \rho_k^{A,B} = \frac{R_k^{A,B}}{\sqrt{Q_k^A Q_k^B}}$$

then

$$\rho_{ijk}^{A,B} = \frac{R_i^{A,B}}{\sqrt{Q_i^A Q_i^B}} + \frac{R_j^{A,B}}{\sqrt{Q_j^A Q_j^B}} + \frac{R_k^{A,B}}{\sqrt{Q_k^A Q_k^B}} \qquad (24)$$

More precisely, the probability of having different results in the ith hidden unit, jth left-dynamic unit and kth right-dynamic unit of the 2 parties is given by the well-known generalization error for the perceptron by (Engel and Van Den Broeck, 2001):

$$\varepsilon_p^i = \frac{1}{\pi} \arccos\ (\rho_{ijk})$$

Synchronization with feedback: The TPMs A and B are beginning with different randomly chosen weight vectors 'w' and common randomly chosen input vectors 'x'. The feedback mechanism is defined as follows:

- The input is shifted at each step 't'. That is $x_{i,j}\ (t+1) = x_{i,\,j-1}\,(t)$ for $j > 1$.
- If $\tau^A(t) = \tau^B(t)$ then $x_{i,1}(t+1) = \phi_i(t)$, else $x_{i,1}(t+1)$ are reset to common values.
- If $\tau^A(t) \neq \tau^B(t)$ for R steps, then all input values are reset to common values.

These evaluations give some privacy to inputs and additionally system becomes sensitive about the learning rule. As described in Ruttor *et al.* (2004), learning rule of anti-hebbian will reveal less information than hebbian learning rule and random walk learning rule. Therefore, the anti-hebbian learning will be more appropriate for the feedback scheme.

NUMBER OF KEYS

To determine the number of keys, this can be produced by the neural key-exchange protocol using a given sequence of inputs, the following system consisting of two pairs of TPMs.

$$\begin{aligned} w_i^{A+} &= g(w_i^A + f(\phi_i^A, \tau^A, \tau^B)x_i), \\ w_i^{B+} &= g(w_i^B + f(\phi_i^B, \tau^A, \tau^B)x_i), \\ w_i^{C+} &= g(w_i^C + f(\phi_i^C, \tau^C, \tau^D)x_i), \\ w_i^{D+} &= g(w_i^D + f(\phi_i^D, \tau^C, \tau^D)x_i). \end{aligned} \qquad (25)$$

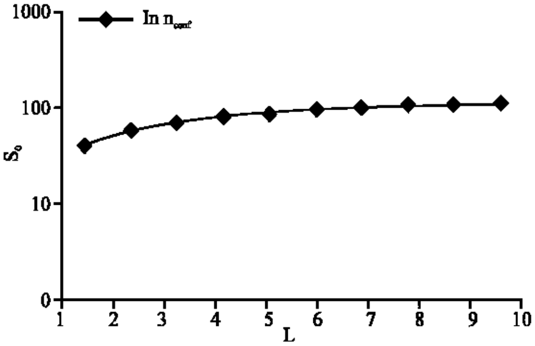

Fig. 2: The weight configuration of a TPM with K = 3, δ = 3, ϒ = 3 and N = 4

where, x_i is the common sequence of input vectors. Both pairs communicate their output bits only internally. Thus A and B as well as C and D synchronize using one of the available learning rules, while correlations caused by common inputs are visible in the overlap ρ_i^{AC}. Because of the symmetry in this system, ρ_i^{AD}, ρ_i^{BC} and ρ_i^{BD} have the same properties as this quantity, so that it is sufficient to look at ρ_i^{AC} only.

The synchronization of networks which do not interact with each other, e.g., A with C, is much more difficult and takes a longer time than performing the normal key-exchange protocol. That is why we assume $\rho^{AB} = 1$ and $\rho^{CD} = 1$ for the calculation of $(\Delta\rho^{AC}\ (\rho^{AC}))$. The number of keys is smaller than the number of weight configuration $n_{conf} = (2L+1)^{(K+\delta+\Upsilon)\cdot N}$ of a TPMs.

We further analyze these correlations by calculating the entropy of the weight distribution (Ruttor *et al.*, 2006)

$$S^{AC} = -(K + \delta + \Upsilon)\cdot N \sum_{a=-L}^{L} \sum_{c=-L}^{L} p_{a.c} \ln p_{a,c} \qquad (26)$$

where, S^{AC} is the entropy of a single hidden, left-dynamic and right-dynamic unit of TPM. Here $p_{a,c}$ is the probability to find $w^A_{ij} = a$ and $w^C_{ij} = c$ by selecting a random weight.

As the weights in each TPM alone stay uniformly distributed, the entropy of 2 fully synchronized networks is given by:

$$S_0 = \ln n_{conf} = (K + \delta + \Upsilon)\cdot N\ \ \ln(2L+1) \qquad (27)$$

From the Fig. 2, we are able to predict a large number of possible weight configuration, in which weights are chosen randomly for a pair of TPMs. The attacker E could use some clever algorithm, it is very difficult to determine weight configuration, which lead to identical results.

Consequently, S^{AS}-S_0 is the part of the total entropy, which describes the correlations caused by using a

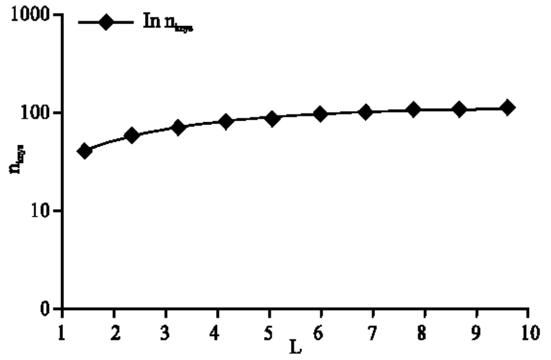

Fig. 3: Number of distinct keys for K = 3, δ = 3, ϒ = 3 and N = 4

common input sequence. It is proportional to the effective length of the generated cryptographic key,

$$l_{key} = \frac{S^{AC} - S_0}{\ln 2} \tag{28}$$

which would be the average number of bits needed to represent it using both an optimal encoding without redundancy and the input sequence as additional knowledge. If the possible results of the neural key-exchange protocol are uniformly distributed, each one can be represented by a number consisting of I_{key} bits. In the case,

$$n_{key} = 2^{l_{key}} = e^{S^{AC} - S_0} \tag{29}$$

However, that the real number can be larger, because not all possible weight configurations occur with equal probability as keys. Therefore n_{key}, in fact, a lower bound for the number of cryptographically usable configuration.

First S^{AS}-S_0 shrinks linearly with increasing t, as the overlap ρ between A and C grows while it approaches the stationary state (Ruttor *et al.*, 2006).

This behavior is consistent with an exponential decreasing number of keys, which can be directly observed in very small systems in Fig. 3. We use this minimum value in order to determine n_{key}.

It is clearly visible that there are two scaling relation for S(t):

- Entropy is an extensive quantity. Thus S^{AC} and S_0 are proportional to the number of weights N. Consequently the number of keys, which can be generated by the neural key-exchange protocol for a given input sequence, grows exponentially with increasing N.
- The synchronization time $t_{sync} \propto L^2$ is the time scale of all process related to the synchronization of TPMs. It depends on the size of the learning steps $\langle \Delta\rho \rangle$. Therefore, the time needed to reach the fixed point of ρ_k^{AC} is proportional to L^2.

In order to determine the dependency between the synaptic depth L and n_{key}, we calculate the mutual information:

$$I^{AC} = 2\ S_0 - S^{AC} \tag{30}$$

between A and C, which is visible in the correlations of the weight vectors using the Eq. (27) and (28), we find:

$$I^{AC} = -\ln\left(\frac{n_{key}}{n_{conf}}\right) \tag{31}$$

Therefore, the number of distinct keys is given by:

$$n_{key} = n_{conf}\ e^{-I^{AC}} = (2L+1)^{(K+\delta+\Upsilon)\cdot N}\ e^{-I^{AC}} \tag{32}$$

Consequently, n_{key} increases exponentially with N,

$$n_{key} \approx [0.66(2L+1)^{(K+\delta+\gamma)}]^N \tag{33}$$

Then

$$\ln n_{key} \approx N \cdot \ln[0.66\ (2L+1)^{(K+\delta+\Upsilon)}] \tag{34}$$

$$\approx N \cdot \ln(0.66) + N \cdot (K + \delta + \Upsilon)\ln(2L+1) \tag{35}$$

From Fig. 3, we are able to estimate a large number of keys which are generated against a brute force attack. However, the attacker E could use some clever algorithm to determine which keys are generated with high probability for a given input sequence, because of large number of keys are generated, the attacker could not find the secret keys easily. A large number of keys are important for the security of neural cryptography against brute force attacks on the neural key-exchange protocol.

ATTACKS

The attacker E is trying to the internal representations of hidden units (σ_1, $\sigma_{2......}$, σ_K), left-dynamic units (δ_1, $\delta_{2......}$, δ_K), right-dynamic units (Υ_1, Υ_2,….., Υ_K) with lower layer spy unit ϑ and upper layer spy unit ξ of A's or B's TPMs.

The lower layer's spy unit vector ϑ and upper layer's spy unit vector ξ are transmitted, associated with output bits τ^A during synchronization process. When the output bits are reached, the destination of B's TPM, the lower layer spy unit's vector and upper layer spy unit's vector are automatically destroyed by the B's TPM over the public channel. The attacker can access in between the A's or B's TPM, the spy unit's vectors with output bits are mixed then pass to the attacker's TPM.

Genetic attack: The attacker E beginning with only one randomly initialized TPM, but E can use upto 'M' neural networks. Whenever, the partners update the weights because of $\tau^A = \tau^B$ in a time step, the following genetic algorithm is applied (Ruttor, 2006):

- The attacker E has at most $M/2^{K-1}$ TPMs and also determine all $2^{K-1} * 2^{\delta-1} * 2^{\Upsilon-1}$ internal representations

  $$\left(\sigma_1^E, \sigma_2^E, \ldots\ldots, \sigma_K^E, \delta_1^E, \delta_2^E, \ldots\ldots, \delta_K^E, \gamma_1^E, \gamma_2^E, \ldots\ldots\gamma_K^E\right)$$

  which reproduce the output τ^A. Then, these are used to update the weights in E's neural networks according to the learning rule, so that $2^{K-1} * 2^{\delta-1} * 2^{\Upsilon-1}$ variants of each TPM in this mutation step, are generated.
- If the attacker E has more than $M/2^{K-1}$ neural networks, only one fittest TPM should be kept. This is achieved by discarding all networks which predicted less than U outputs τ^A in the last V learning steps, with $\tau^A = \tau^B$ successfully, where U is the minimal fitness and V is the length of the output history.

Geometric attack: In the geometric attack $\tau^A = \tau^B - \vartheta\xi$, attacker E tries to remove the lower layer spy unit vector and upper layer spy unit vector, otherwise this cannot be done by just applying the same learning rule. The attacker tries to correct the internal representation of own TPM using the local fields,

$$\left(\sigma_1^E, \sigma_2^E, \ldots\ldots, \sigma_K^E\right)$$

for hidden units,

$$\left(\delta_1^E,\ \delta_2^E, \ldots\ldots, \delta_K^E\right)$$

for left-dynamic units,

$$\left(\Upsilon_1^E, \Upsilon_2^E, \ldots\ldots, \Upsilon_K^E\right)$$

for right-dynamic units as additional information. These quantities can be used to find out the level of confidence associated with the output of each hidden, left-dynamic and right-dynamic units. The low absolute values,

$$\left|\sigma_i^{E,m}\right|, \left|\delta_i^{E,m}\right| \text{and} \left|\Upsilon_i^{E,m}\right|$$

indicate a high probability of $\phi_i^A = \phi_i^E$, the attacker E changes the output ϕ_i^E of the hidden, left-dynamic and right-dynamic unit with minimal absolute values,

$$\left|\sigma_i^E\right|, \left|\delta_i^E\right| \text{ and } \left|\Upsilon_i^E\right|$$

and the total output $\tau^E - \vartheta\xi$ before applying the learning rule.

The geometric attack does not always succeed in estimating the internal representation of A's TPM correctly due to the hidden, left-dynamic and right-dynamic units with $\phi_i^A = \phi_i^E$.

Majority attack: The majority attack E can improve the ability to predict the internal representation of A's neural network. The attacker E uses an ensemble of 'M' TPMs instead of a single neural network. At the beginning of the synchronization process, the weight vectors of all attacking networks are chosen randomly.

The attacker E does not change the weights in time steps with $\tau^A \neq \tau^B$, because the partners skip these input vectors. But for $\tau^A = \tau^B$, an update is necessary then the attacker calculates the output bits $\tau^{E,m}$ with lower layer spy unit and upper layer spy unit. It is very hard to estimate output bit $\tau^{E,m}$ by an attacker because of spy units vectors are mixed with output bits when entering into attacker's TPM. If the output bit $\tau^{E,m}$ of the mth attacking network disagrees with τ^A, then attacker E searches the hidden, left-dynamic and right-dynamic units of 'i' with minimal absolute values $|\sigma_i^{E,m}|$, $|\delta_i^{E,m}|$ and $|\gamma_i^{E,m}|$. Then, the output bits $\phi_i^{E,m}$ and $\tau^{E,m}$ are inverted similarly to the geometric attack. Afterwards the attacker counts the internal representations $(\phi_i^{E,m}, \ldots\ldots, \phi_i^{E,m})$ of own TPM and selects the most common one. The majority votes are adopted by all attacking networks for the application of the learning rule.

CONCLUSION

In the proposed TPMs, the synchronize time of the attacker is increased by the three transfer functions of hidden unit using Hebbian learning rule, left-dynamic unit using Random walk learning rule, right-dynamic unit using Anti-Hebbian learning rule with lower layer spy unit and upper layer spy unit. A feedback mechanism with hidden, dynamic and spy units decrease the probability of the successful attack. The synchronization time and feedback yield an improvement of the security of the system. We generate the effective number of keys, both a configuration space $n_{conf} = (2L+1)^{(K+\delta+\Upsilon)\cdot N}$ and n_{key} grow exponentially with increasing number of weights per hidden unit, left-dynamic unit and right dynamic unit. Therefore, a large value of N guarantees the security of neural cryptography against brute-force attacks.

REFERENCES

Engel, A. and C. Van Den Broeck, 2001. Statistical mechanics of learning. Cambridge University Press, Cambridge.

Godhavari, T., N.R. Alamelu and R. Soundarajan, 2005. Cryptography using neural network. IEEE Indicon Conference, pp: 258-261.

Kanter, I. and W. Kinzel, 2002. Neural cryptography. In: Proceeding of the 9th International Conference on Neural Information Processing Vol. 3, Singapore, 18-22 Nov. 2002, pp: 1351-1354.

Kanter, I., W. Kinzel and E. Kanter, 2002. Secure exchange of information by synchronization of neural network. Euro Phys. Lett., 57 (1): 141-147.

Kanter, I., W. Kinzel, L. Shancham, E. Klein and R. Mislovaty, 2004. Cooperating attackers in neural cryptography. Phys. Rev. E., 69 (6): 066137-3.

Kinzel, W. and I. Kanter, 2002. Interacting neural networks and cryptography. Adv. Solid State Phys. (Springer, Berlin), 42: 383-391.

Kinzel, W., 2002. Theory of Interacting Neural Network. Preprint cond-mat/0204054.

Kinzel, W., R. Metzler and I. Kanter, 2000. Dynamics of Interacting neural networks. J. Phys. A: Math Gen. 33: L141-L149.

Kinzer, W., 1997. Phase transition of neural networks plenary talk for the MINERVA workshop on mesoscopics fractals and neural networks. Eilat, [cond-mat/9704098 v1].

Klein, E., R. Mislovaty, I. Kanter, A. Ruttor and W. Kinzel, 2004. Synchronization of neural networks by mutual learning and its application to cryptography. In: Proc. of Neural Information Processing Systems (NIPS), Whilster, British Columbia, Canada.

Klimov, A., A. Mityagin and A. Shamir, 2002. Analysis of neural cryptography. In: Proceeding of Asia Crypt 2002. LNCS, Queenstown, New Zealand, Springer Verlag, pp: 288-298.

Metzler, R., W. Kinzel and I. Kanter, 2000. Interacting neural networks. Phys. Rev. E., 62 (2): 2555-2565.

Mislovaty, R., E. Klein, I. Kanter and W. Kinzel, 2004. Security of neural cryptography. IEEE. Trans., 5 (4): 219-221.

Mislovaty, R., E. Klein, I.Kanter and W. Kinzel, 2003. Public channel cryptography by synchronization of neural networks and chaotic maps. Phys. Rev. Lett., 91, No. 118701.

Mislovaty, R., Y. Perchenok, I. Kanter and W. Kinzel, 2002. Secure key-exchange protocol with an absence of injective functions. Phys. Rev. E., 66: 1-4.

Prabakaran, N., P. Loganathan and P. Vivekanandan, 2008. Neural cryptography with multiple transfer function and multiple learning rule, 3 (3): 177-181.

Rosen-Zvi, M., E. Kleign, I. Kanter and W. Kinzel, 2002. Mutual learning in a tree parity machine and its application to cryptography. Phys. Rev. E., 66 (13): 066135.

Ruttor, A., I. Kanter, R. Naeh and W. Kinzel, 2006. Genetic attack on neural cryptography [cond-mat/0512022v2/1].

Ruttor, A., 2006. Neural Synchronization and cryptography, Ph. D. Thesis.

Ruttor, A., I. Kanter and W. Kinzel, 2006. Dynamics of neural cryptography [cont-mat/061257].

Ruttor, A., W. Kinzel and I. Kanter, 2002. Neural cryptography with queries. Phys. Rev. E., 25 (1): 1-12.

Ruttor, A., W. Kinzel, L. Shacham and I. Kanter, 2004. Neural cryptography with feedback. Phys. Rev. E., 69: 1-8.

Volkmer, M. and S. Wallner, 2005. Tree parity machine rekeying architectures. IEEE. Trans. Comput., 54 (4): 421-427.