Public Key Exchange by Neural Networks

Zahir Tezcan

Computer Engineering, Bilkent University, 06532 Ankara
zahir@cs.bilkent.edu.tr

Abstract. This work is a survey on the concept of neural cryptography, which is a communication of two tree parity machines for agreement on a common key over a public channel. Not only the communication but also the convergence analyses for both natural and feedback model are included. At last, some modifications on the communication protocol are proposed.

1 Introduction

Since January 2002, when the Physicists Kanter, Kinzel and Kanter [1] proposed a new key exchange protocol between two parties, there is a new notion of public key exchange, neural cryptography. They made it possible for two weakly interacting chaotic systems to converge, such that they agree on the same information.
Now, let us look at the concepts of neural networks and public key exchange, which will be basis for our observations.

1.1 Neural Networks

Throughout the foundation of learning rule for the synapses between neurons, some mathematical explanations of artificial neurons are given, which gave birth to the concept of neural networks [2].
To be more specific, a neural network is a weighted graph of artificial neurons, such that the learning process of a neural network is to adjust the weights of the graph. The accuracy of the learning neural network is changed by propagating the output error to the adjustment process of the weights for each given input. Therefore, by this back-propagation, precision of the output is increased.
For the purpose of neural cryptography, we will fix a learning rule (i.e. an adjustment process for the graph weights) for two neural networks of the same structure, such that they will be synchronized in the end. The synchronization will be in the sense that, from some point on they will have the same weights, even if they change their weights continuously in each step.

1.2 Public Key Exchange

Informally, a public key exchange protocol is for agreement on a secret over a public channel for which any eavesdropper of the channel could not be able to come up with the same secret.
For the key exchange, in the protocol of neural cryptography, each side of the synchronization process will use the weights as a secret key. Since the weights of the neural networks are random at the beginning of the communication, the synchronized weights could be recognized as an agreement on the same information.
As a consideration of the secrecy of the protocol, any eavesdropper of neural network should not be able to come up with the same weights, or in other words, should not be synchronized with the on going communication process.

As an outline of this work, first we describe the details of protocol in section 2. Details are given in the order such that what is the structure of a tree parity machine (2.1), how the two end points of communication synchronize (2.2), what any attacker can do (2.3) and some modifications proposed by the founders of the concept (2.4). In section 3, I will propose some modifications for decreasing the time complexity with respect to the increasing security of the protocol. And we will conclude in section 4.

2 Neural Cryptography

As I have described in previous section, neural cryptography is about the synchronization of two neural networks, hopefully without leaking information.
The two neural networks should be given the same inputs, such that each endpoint of the communication should have common information before to begin communication. For each communication step they should give common inputs to the neural networks, which can be done by a pseudorandom stream generator by giving common initial seed [3].
The neural network structure is a tree parity machine (TPM) such that it outputs a single bit (i.e. a parity bit) for some input. Now, we will examine the structure of a TPM, the communication and learning process of neural cryptography protocol.

2.1 Tree Parity Machine (TPM)

We know that a neural network is a graph of artificial neurons. A TPM is a tree structured graph of artificial neurons. Height of a TPM, in terms of graph concepts, is two. Additionally, leaves of a TPM are input units, intermediary nodes are hidden units, and root is the output unit of a TPM.
The edges between hidden units and root are not weighted. So, the weighted ones are the edges between hidden units and input units. Let us have a look on the structure of these hidden units, which are the backbones of both the TPM and neural cryptography.

Hidden Unit Structure The notation for the structure is as follows, K is the number of hidden units in TPM. N is the total number of inputs. X is the input vector of size N. W is the weight vector of size N. When we see a declaration of weights or inputs as w(i, j) or x(i, j), those are respectively j^{th} weight or j^{th} input for i^{th} hidden unit. At last τ stands for the output of the TPM.

As explained before, hidden unit is itself a tree. The function inside the hidden unit is the sign of weighted sum of inputs as follows:

$$\sigma(i) = sign(\Sigma_j[w(i, j)x(i, j)]) \qquad \textbf{(1)}$$

The *sign* is the signum function by a small difference, such that one of the endpoints counts zero as negative while the other side counts as positive[5]. Moreover the output of whole TPM is simply multiplication of the outputs of hidden units. Namely,

$$\tau = \Pi_i[\sigma(i)] \qquad \textbf{(2)}$$

Therefore the output of a TPM is a parity check on the weighted sum of its inputs. It is clear that for common input and weight vectors, the output of the TPMs should be same. Now a question arises, how could we synchronize randomly weighted TPMs?

Communication and Learning Process The TPMs in each endpoint will have same input vectors, where each input is either 1 or -1. The weight vectors will be randomized, where each weight will be an integer between L and –L, such that L is also a parameter for the communication process. For each step of communication the TPMs exchange their output bits in order to run their learning algorithms.

We will mark one of the endpoint as A and the other as B without loss of generality. Then the learning process is as follows:

```
Initially, set $W_0$ to random values
For each turn t
    Set $X_t$ to common values
    Run TPMs and exchange outputs
    $\tau_{t,A}$ is not equal to $\tau_{t,B}$ ➔ No change
    $\sigma_{A|B}(i) = \tau_A = \tau_B$ ➔ Train each such unit i
    $\tau_A = \tau_B$ for last $t_{min}$ turns ➔ Finish
```

Here t_{min} should be analyzed and considered to be a threshold of learning process, such that if machines give the same output for some number of times then we can

assume that they are synchronized[6]. The next thing for the communication is the training of hidden units.
There are three simple ways of training [7]:

$$W_{t+1}(i) = W_t(i) - \tau_t X(i)\Theta(\sigma_t(i)\tau_t)\Theta(\tau_{t,A}\tau_{t,B}) \qquad (3)$$

$$W_{t+1}(i) = W_t(i) + \tau_t X(i)\Theta(\sigma_t(i)\tau_t)\Theta(\tau_{t,A}\tau_{t,B}) \qquad (4)$$

$$W_{t+1}(i) = W_t(i) + X(i)\Theta(\sigma_t(i)\tau_t)\Theta(\tau_{t,A}\tau_{t,B}) \qquad (5)$$

Remark 1: if for any $w(i,j) \in W(i)$, $|w(i,j)|>L$ then set $w(i,j) = \mathit{sign}[w(i,j)]L$
Remark 2: Θ is the Heaviside function for which, if the input is positive, then output is 1 and if input is negative then the function evaluates to 0. Moreover, function is not defined on 0. Here, the usage of Heaviside function is to guarantee that the training process is executed when TPMs have common output and only the hidden units having the common output are trained.
It is easy to see that if two TPMs are synchronized such that they have the same weights, then for next turn their weights will be same too. So, after synchronization, they will remain synchronized.

2.2 Analysis of Convergence

The question of the synchronization is: why do TPMs converge to the same state of weights, even if they are distributed randomly? Well, as the founders of the system are physicists, they came up with the definition of attractive and repulsive forces.
Mutual Overlap is a metric for measuring the coherency of states of the weight vectors for each side of the communication [7]. For example, for any two vectors over a two dimensional space, the probability of having same projections on any axis increases while the angle between vectors converges to zero. And the probability decreases, while the angle approaches to $\pi/2$. So, we may approximate the cosine of the angle between vectors as probability of having same values for each attribute. Then define mutual overlap (ρ) for weight vectors $W_A(i)$ and $W_B(i)$ as

$$\rho(i) = W_A(i).\, W_B(i) /|\, W_A(i) \,||\, W_B(i)\, | \qquad (6)$$

Then the probability $\varepsilon(i)$ such that a common randomly chosen input vector X(i) leads to a different output for hidden unit $\sigma(i)$ is given by[7]

$$\varepsilon(i) = 1/\pi[\arccos \rho(i)] \qquad (7)$$

Therefore, the vectors should be adjusted in a way that, for the visualization of vectors, they should be attracted to each other and for an eavesdropper; attacker’s adjustment should result in repulsion of the weight vectors. So, for the learning rules (3), (4) and (5); the vectors are attracted if the outputs of the hidden units are equal to the common output. And they do repulsive moves if their outputs are not common.

2.3 Attacker

For a secure key exchange protocol, any attacker who knows all of the details of the protocol and all of the information exchanged should not have the computational power to calculate the secret key.
Under the assumption of the attacker E, who knows the algorithm, input vectors and output bits; could start from all of the $(2L+1)^{3N}$ initial weight vectors and calculate the ones which are consistent with the communication made. All of these initial states end up with the same final weight vector, so the key is unique. However, this task is computationally infeasible [7].
Although, the feasibility of the attack is in question, the attacks that make the attacker's probability of finding the key high are declared in [8].

Genetic Attack is made by a population of TPMs. As our communicating TPMs update their weights while the outputs are the same, the attacker's TPM population is increased by the ones, whose outputs are also the same with actually communicating TPMs. So, the population of TPMs evolves in the way that if a TPM's output is common with the communicating TPMs then it lives and reproduces; or dies if not same with the common output.

This is the step of evolution of TPMs as given in [8](A and B are actual TPMs):

A and B have different outputs τ_A, τ_B; and thus do not change their weights. Then all the attacker's networks remain unchanged as well.

A and B have the same outputs $\tau_A = \tau_B$, and the total number of attacking networks is smaller than some limit M. Then the attacker replaces each network C from the population by variants of itself, $C_1, \ldots, C_4$ where in each variant the actual hidden outputs of the hidden units are replaced by one of the four possible combinations whose product is equal to the declared τ_A. The attacker then uses the standard learning rule with the new hidden outputs in order to update the weights of each network.

A and B have the same outputs $\tau_A = \tau_B$ but the total number of simulated networks machines is larger than M. In this case the attacker computes the outputs of all the networks, deletes the unsuccessful networks whose output is different from τ_A, and updates the weights in the successful networks by using the standard learning rule with the actual hidden outputs of the hidden units.

Geometric Attack relies on the similarities of the structure of TPM and a hyper plane definition. This, enables us to assume each X(i) as a hyper plane such that:

$$f(z(1),\ldots,z(n)) = \Sigma_j\, x(i,j).z(j) \qquad (\mathbf{8})$$

$$f(Z) \in U \text{ where } U = \{-L,\ldots,L\}^N \qquad (\mathbf{9})$$

So, we can think of K weight vectors W(1),...,W(K) as points in vector space U. Then the attack is as follows.
The attacking network C of the communicating TPMs A and B considers one of three hidden units (i.e. ith hidden unit). Then, if output of C is different from the common output, then most probably, only one hidden unit should have different output. So, actual output $W_A(i)$ will be separated from $W_C(i)$ by X(i) in U. Hence, we will deal with the distance from $W_C(i)$ to hyper plane X(i), which is: $|f(W_C(i))|$. Therefore, the attack will be nothing but minimizing this distance, when A and B have same output and C has different output

Probabilistic Attack goes under the assumption that distribution of w(i,j) are independent. So, we can find the probability of position of W(i) in the tth step from the probabilities for (t-1)st step. Such that, unconditional probability of being in any position over the field is 1/(2L+1). And conditionally, for any position *l*, probability will be the sum of probabilities of being in position *m*, where if $X_{t-1} = m$ then $X_t = l$. Thus, by dynamic programming it will be possible to compute distribution of w(i,j).x(i,j) because that τ is publicly known. Therefore, with the same probability of communicators A and B, attacker C will approach to the common vectors.

2.4 TPM with Feedback

As a consideration about the attacks on the scheme, the attacker's probability behaves similar to the synchronization probability function. Such that, the attacker approaches to synchronization as well as the communicating TPMs approach to the synchronization. Hence, there should be a way that increases the repulsion for the attacker, and the probability of the attack should drop drastically with respect to the decrease in synchronization probability.
In [7] Ruttor, Kinzel, Shacham and Kanter proposed TPM with feedback scheme. For feedback scheme, there are three new evaluations to consider:

After each step t the input is shifted, $x_{(t+1)}(i,j) = x_t(i,j-1)$ for j > 1.

If $\tau_{t,A} = \tau_{t,B}$ then $x_{(t+1)}(i,1) = \sigma_t(i)$, else $x_{(t+1)}(i,1)$ are set to common values

If $\tau_{t,A}$ is not equal to $\tau_{t,B}$ for R steps, then all X are reset to common values.

It is easy to see that, these evaluations give some privacy to inputs, and additionally system becomes sensitive about the learning rule. As described in [7] learning rule (3) will reveal less information then (4) and (5). Therefore, the anti-Hebbian learning will be more appropriate for the feedback scheme.

Attack Analysis of the TPM with feedback shows that basically the probability of the attacker drops down exponentially with L^2. Moreover, for large values of N probability of the attacker decreases with L. Not only the parameter L, but also R comes into place, for which increasing values of R make probability of the attacker to decrease as well [7].

As we have assumed it to be, security is increased with the confusion of the input values. However, making confusion on the input also results in the increase of the synchronization time. Therefore, for constant effort on the synchronization, increase in the security does not seem promising.

3 Modifications on the Structure

As we have inspected the analyses of the scheme, the main problem they have insisted on is the security of the protocol, which is based on the synchronization probability of the attacker. However, not only the security but also the usage of the scheme is a problem. The two TPMs exchange only single bits for the whole synchronization process, but we use headers for communication, where hundreds of bits are only overhead.

3.1 Multiple Machines

Since, the need for multi-bit output we can simply use multiple TPMs in each endpoint. But, some questions arise here. What will be the effect on the synchronization? How will the key be computed? What will be the effect on the probability of the attacker?

It is clear that the output of one endpoint is simply concatenation of the output bits of each TPM. Therefore, the synchronization time will be $O(T.S)$ where T is number of TPMs in each endpoint, and S stands for the synchronization complexity of a TPM pair. Additionally, not to increase the probability of attacker, for each synchronized TPM P_k, the kth bit in the T-bit output is randomized (or simply set to some constant[1]) and P_k is stopped.

Now, let us think of the key. If we use concatenation of the weight vectors W_k for each TPM, then under the assumption that attacker has found some TPMs' weight vectors, parts of key would be compromised. Hence, simply exclusive-or of weight vectors will reveal no information unless the attacker finds W_k for each $k \in \{1,...,T\}$. So the key F as a function of weight vectors is:

$$F = \oplus_k W_k \tag{10}$$

As the computation of exclusive-or operation is simple, the computation time of the key will be $O(T)$.

There is no reveal of information, when the attacker synchronizes without all TPMs, and the synchronizations of the TPMs are independent. So, the probability of the attacker will be p_E^T, where p_E is the synchronization probability of the attacker for a single pair of TPMs. Hence, the probability of the attacker decreases exponentially with T.

Therefore, this multi-bit protocol increases the synchronization and processing time linearly by T, while the synchronization probability of the attacker decreases exponentially with T.

[1] Since the attacker eavesdrops the overall output, he can consider that kth bit is same for t_{min} rounds and kth TPMs are synchronized

3.2 Diffusion for the Attacker

For special uses of the protocol there are some improvements as well. Basically, the usage of the protocol is a session key agreement, where it is under the assumption that two endpoints have already agreed on a long term key. By using this long term key, they create random (actually deterministic randomization is achieved by using long term key as a seed) sequences for non-public but common input vectors.
However, the attacker can still use his assumptions on the input and can approach to the synchronization as well. Therefore, we can propose a random permutation, which relies on the common secret randomization process, on the output bits for multi-bit scheme. These will be nothing but diffusion for the attacker.
For the sake of completeness we can set the number of output bits to T+Z where T is the number of TPMs and Z is the number of extra random bits, for which randomization function takes secret as a seed too. But randomization function is different from the function for input randomization process. Additionally, we should randomize the output of synchronized TPMs (i.e. we set them constant in previous scheme) for no reveal of information on the permutation[2].

3.3 Possible Attackers

As I have described earlier, in the sense of geometric or probabilistic attacks, the probability of the attacker will decrease exponentially while the computing time increases linearly. However, for genetic attack the probability of the attacker does not diminish but the attacker's processing time increases as well as the communicator endpoints. Moreover, instead of the traditional scheme, if we use feedback system then inherently, genetic attack will be almost impossible to synchronize.

4 Conclusion

The neural cryptography schemes proposed so far, do not seem promising. But, this is a quite new approach to public key exchange and people in both physics and cryptography communities would find some way to make it reasonable.
Moreover, we have given two new approaches to the neural cryptography. Both of them use multi-bit communication, which decreases the attacker's probability with respect to the synchronization complexity. Additionally, one of them is for specific use (i.e. secret input vectors), and almost eliminates the attacker.
Furthermore, there are some more interesting titles for this scheme, such that they are investigated in [4] and much closer to the cryptography.

[2] Since, the synchronized TPMs output the same bit, while approaching to the synchronization, the permutation, which is a function of the common secret, will be clear.

References

1. Ido Kanter, Wolfgang Kinzel, Eran Kanter,”Secure exchange of information by synchronization of neural networks”, Europhys., Lett. 57, 141 (2002)
2. J. Hertz, A. Krogh, and R. G. Palmer: Introduction to the Theory of Neural Computation, Addison Wesley, Redwood City (1991)
3. Markus Volkmer and André Schaumburg: Authenticated tree parity machine key exchange (2004).
4. M. Volkmer and S. Wallner: “Tree parity machine rekeying architectures”, IEEE Transactions on Computers (2004).
5. Wolfgang Kinzel and Ido Kanter: “Neural Cryptography”, cond-mat/0208453 (2002).
6. E. Klein, R. Mislovaty, I. Kanter, A. Ruttor, W. Kinzel: “Synchronization of neural networks by mutual learning and its application to cryptography”, NIPS (2004).
7. A. Ruttor, W. Kinzel, L. Shacham and I. Kanter: “Neural Cryptography with Feedback”, Phys. Rev. E 69 046110 (2004).
8. A. Klimov, A. Mityagin and A. Shamir: “Analysis of Neural Cryptography”, ASIACRYPT (2002).