Vol. 4, No. 1 Jan 2013 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2009-2013 CIS Journal. All rights reserved.

http://www.cisjournal.org

Neural Network Synchronization Protocol

[1] **Jiri Giesl,** [2]**Radim Pisan**
[1] Software Developer, Monet+, A.S., Zlin, Czech Republic
[2] Software Developer, ADEON CZ, s.r.o., Zlin, Czech Republic
[1] giesl.jiri@email.cz, [2] pisanr@seznam.cz

ABSTRACT

This work deals with the design of secure communication protocol based on neural network synchronization. Protocol is built on the application-layer of the OSI upon TCP/IP. Server and each connected client must create special type of neural network called Tree Parity Machine then compute outputs of their neural networks and iteratively exchange those outputs through the public channel. After some time neural networks will become synchronized that way and will share secret information. Symmetric key which is then used for encrypting subsequent communication is derived from that shared secret by standard hash algorithm.

Keywords: *Neural network, synchronization, secure messaging, cryptographic protocol, communication protocol, shared secret, tree parity machines*

1. INTRODUCTION

Two totally different types of neural network learning process exist - batch and online. Batch learning process lies in presenting all learning samples in the same learning step. Those samples are then used for optimization of network's weights. On the other hand, online learning process is based on creating new sample in each learning step. Neural network then changes its weights according that sample and it is possible to train that network by dynamic rules changing in time.

Special case of online learning process, so called neural network synchronization, is described and deeply analyzed in [1,2]. The whole process lies in computing and exchanging outputs between two neural networks in each step. Synaptic weights are then modified according learning rule. This process is repeated iteratively until two networks have equal synaptic weights, i.e. neural networks are synchronized.

More complex neural networks called Tree Parity Machines (TPMs) show a new phenomenon – synchronization is faster than classic online learning. This solves classic cryptographic problem when two active participants of communication need to exchange secret message over insecure channel. Active participants are able to synchronize their synaptic weights much faster than passive eavesdropper. When active participants are synchronized the synaptic weights can be considered as a shared secret. Deeper analysis of synchronization time and its dependency on the structure of TPM can be found in [1,3]. Synchronization has been used in several applications such as remote authentication [4,5] and also advanced algorithm, such as logistic map, has been used to improve synchronization [6].

Synchronized weights, i.e. shared secret created on both sides of communication channel, can be used for generation of symmetric encryption key. Then, it is possible to establish secure communication channel by proper encryption algorithm. TCP/IP technology allows implementation of this type of protocol and this work deals with its design where TPM networks play their roles during synchronization, symmetric key is computed by hash algorithm SHA-256 and AES is used as a default encryption scheme.

2. TREE PARITY MACHINES

A. Structure

Tree Parity Machines (TPMs) are multi-layered neural networks which contain K hidden neurons and each neuron has N input units and one output unit. Common TPM structure is drawn on the Figure 1.

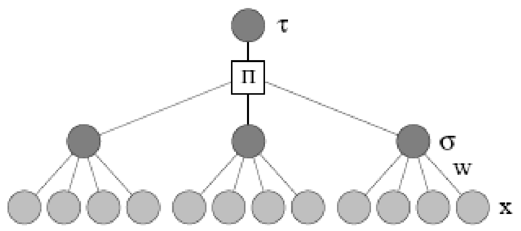

Fig 1: TPM with K=3, N=4

All input values x has binary form (1) and weights w are discrete numbers (2)

$$x_{i,j} \in \{-1,+1\} \qquad (1)$$

$$w_{i,j} \in \{-L,-L+1,...,+L\} \qquad (2)$$

Index i=1,…K represents i-th neuron in hidden layer and j=1,…N represents j-th item of input vector.

State h of each neuron in hidden layer is computed as multiplication of input value and appropriate weight (3).

$$h_i = \frac{1}{\sqrt{N}} \sum w_{i,j} \cdot x_{i,j} \qquad (3)$$

Output δ is defined according (4), thus it has binary form {-1,+1} according sign of state h. Special case of h=0 is mapped to δ=-1 to ensure binary form.

$$\delta_i = sign(h_i) \qquad (4)$$

Whole output τ of TPM is computed as multiplication of δ of all neurons (5). Value τ represents parity of TPM.

$$\tau = \prod \delta_i \qquad (5)$$

Output τ indicates if number of neurons with δ = -1 is even (τ = +1) or odd (τ = -1). This suggests that 2^{K-1} various representations of neurons in hidden layer lead to the same value τ.

B. Synchronizing

Weights $w_i^{A/B}$ are created in random way at the beginning of synchronizing stage of two networks (A and B). Therefore, they are uncorrelated. Input vectors x are generated in each step and output bits $\tau^{A/B}$ are then computed. Exchange of output bits between both networks are also done in each step. Weights $w_i^{A/B}$ are not modified if output bits $\tau^{A/B}$ are different; otherwise Hebbian learning rule (6) is applied to weights. Also, only neurons where δ equals τ are modified by learning rule. Function Θ stands for Heaviside step function.

$$w_{i,j}^{A/B} = g(w_{i,j}^{A/B} + x_{i,j}^{A/B} \cdot \tau \cdot \Theta(\delta_i \tau^{A/B}) \cdot \Theta(\tau^A \tau^B)) \qquad (6)$$

Learning rule must ensure that weights stay in allowed range {-L,+L}. This is done by *g(ω)* (7) which is part of learning rule.

$$g(\omega(= \begin{cases} sign(\omega ign & |\omega| > L \\ \omega & others \end{cases} \qquad (7)$$

Learning process has to be repeated until weights $w_i^{A/B}$ in both networks are identical (i.e. A and B are synchronized). Further learning process will not break synchronization because modification of weights lies on inputs and weights only and they are identical in both neural networks from that time.

3. PROTOCOL DESIGN

Proposed Neural Network Synchronization Protocol (NNSP) is secure application-layer protocol implemented upon TCP/IP. Server must listen on its bounded socket for incoming connection until some client requests for communication. Protocol can be described by five following stages.

A. Stage 1 – Neural Networks Creation

Multi-threaded server creates TPM network of pre-defined structure for each connected client and settings of that network are send back to client as fixed-length message of structure (I,N,W), where N means number of neurons in network with I input vectors for each neuron and weight-range W.

After receiving that message client is able to create its own neural network with the same structure as server. Weights are generated randomly on both sides with respect to the weight-range.

B. Stage 2 – Synchronization Process

Learning process consists of iterating several steps. Description of the one iteration follows:

Step1: Client sends info that it is prepared for synchronizing its network. Packet payload is 1 byte length (consists of control byte).
Step2: Server generates seed value s for generating inputs of its network, computes output value τ^A
Step3: Server sends s and τ^A to the client. Packet payload is 6 bytes length (control byte, value byte, seed value takes 4 bytes).
Step4: Client gets s, generates its inputs by s and computes output value τ^B
Step5: Client sends τ^B and info about going to start learning process back to server, client starts to apply learning rule to its network using τ^A and τ^B. Packet payload is 2 bytes length (consists of control byte and value byte).
Step6: Server gets info about client's learning and starts its own learning process.
Step7: Server sends info about end of learning process to client. Packet payload is 1 byte length (consists of control byte).
Step8: Client gets info about server's end of learning process and sends command for applying the next iteration (back to Step1).

This stage is done iteratively for pre-defined number of iterations which should be big enough to make sure that both sides will be synchronized. As can be seen the amount of information per iteration exchanged between peers is not high. Client produces only 3 bytes each iteration and server responds by 7 bytes. Nevertheless, number of iterations must be taken into account when computing total amount of exchanged data. Number of iterations, i.e. synchronization time, can be exponentially prolonged by number of neurons, their inputs or weight range of TPM networks. According to [1] the weight range is the most affecting parameter of synchronization time.

C. Stage 3 – Key Derivation

Secret key derivation algorithm is based on hash computation of weights of synchronized networks. Values of all neuron's weights are serialized into the buffer which is then used as the input value of function H (8). Function H stands for SHA-256 hash algorithm in default.

$$k_{derived}^{A/B} = H\left(\left\{w_{0,0}^{A/B}, \ldots, w_{i,j}^{A/B}\right\}\right) \qquad (8)$$

Produced output is fixed-length hash value $k_{derived}$ which is considered as symmetric 256-bit AES key for securing communication. Key is derived by both participants this way. Server and client are allowed to destroy their TPM networks once secret key has been created.

D. Stage 4 – Secret Key Validation

It is necessary to check if server and client have generated the same encryption key before its usage. This can be done in very simple way by using following steps of challenge verification:

Step1: Client generates 8-byte random challenge cc.
Step2: Client computes hash h_{cc} of cc using function H (9). Challenge cc is also encrypted by client's $k_{derived}$ key according (10). Resultant message M (11) is concatenation of first 2 bytes and last 2 bytes of hashed cc and encrypted challenge.

$$h_{cc} = H(cc) \qquad (9)$$

$$EC = Enc(k_{derived}^{B}, cc) \qquad (10)$$

$$M = EC \| h_{cc}[0] \| h_{cc}[1] \| h_{cc}[n-1] \| h_{cc}[n] \qquad (11)$$

Step3: Client sends message M to server.
Step4: Server gets EC part of received message M and decrypts it by its $k_{derived}$ key as stated at (12). Challenge cc will be then at disposal.

$$cc = Dec(k_{derived}^{A}, EC) \qquad (12)$$

Step5: Server applies (9) to compute hash of received challenge cc. Server gets last 4 bytes of received message M and verifies those values with recently computed hash. If values differ in one or more bytes, verification of secret key failed and client should be disconnected from the server.

E. Stage 5 – Securing Communication

Derived key $k_{derived}$ is used for encrypting every following data. AES (i.e. Rijndael) is used as a default encryption algorithm because of its high-level security and high performance.

4. CONCLUSION

Theory of neural network synchronization has been put into practice. Secure communication protocol has been proposed by using TPM networks and standard cryptographic algorithms. Protocol is able to work at classic client-server architecture on the application-layer of the OSI model. When both peers are connected together their neural networks are synchronized. Amount of data exchanged during synchronization depends on the settings of neural networks. Higher number of neurons, their inputs and weight range leads to longer synchronization time. Nevertheless, this synchronization process is much faster than synchronization of an eavesdropper; therefore mutual shared secret of client and server can be easily created this way. Subsequent communication between client and server is then encrypted by a key derived from that shared secret. Source codes of proposed protocol are available at [7].

REFERENCES

[1] A. Ruttor, Neural Synchronization and Cryptography, Dissertation, Bayerischen Julius-Maximilians-Universitat Wurzburg, 2006.

[2] W. Kinzel, I. Kanter, Interacting Neural Networks and Cryptography, Adv. in Solid State Phys. 42, vol. 42, pp. 383-391, 2002.

[3] R. M.Jogdand, S.Bisalapur, Design of an efficient neural key generation, International Journal of Artificial Intelligence & Applications (IJAIA), vol. 2, no. 1, pp. 60-69, 2011.

[4] T. Chen, J. Cai, A Novel Remote User Authentication Scheme Using Interacting Neural Network, ICNC 2005, LNCS 3610, pp. 1117-1120, 2005.

[5] E-J. Yoon, K-J. Yoo, A New Remote User Authentication Scheme Using Interacting Neural Network, SERC: Journal of Security Engineering, vol. 5, no. 2, pp. 107-114, 2008.

[6] E. Klein, R. Mislovaty, I. Kanter, et al., Synchronization of neural networks by mutual learning and its application to cryptography, Advances in Neural Information Processing Systems 17, Cambridge, MA, pp. 689-696, 2005.

[7] J. Giesl, R. Pisan, NNSP - Neural Network Synchronization Protocol, available at: http://nnsp.ucinne.cz, 2012.

AUTHORS

JIRI GIESL received Ph.D. degree in Engineering Informatics from Tomas Bata University in Zlin, Czech Republic in 2011. His areas of interests are mainly digital signal processing, compression, deterministic chaos, symmetric and asymmetric cryptography.

RADIM PISAN received master's degree in Automatic Control and Informatics from Tomas Bata Univerzity in Zlin, Czech Republic in 2008. Since 2008 he has been Ph.D. student. He is principally concerned with researches using neural network for process control and modeling technological process.