UQAC

ENSEEIHT

Neural cryptography key exchange

authors:
THOMAS - BARRAS
MAXIME - ARBELOT

Abstract

Neural cryptography is a key exchange protocol based on Diffie-Hellman built to avoid Man in the middle attack. Both Alice and Bob build simplified neural networks called Tree Parity Machine and synchronize them. Eve also build a network, but slower than the originals.

1 Diffie-Hellman key exchange

Alice				**Bob**		
Secret	**Public**	**Calculates**	**Sends**	**Calculates**	**Public**	**Secret**
a	p, g		p, g $\rightarrow$			b
a	p, g, A	g^a mod p = A	A $\rightarrow$		p, g	b
a	p, g, A		$\leftarrow$ B	g^b mod p = B	p, g, A, B	b
a, s	p, g, A, B	B^a mod p = s		A^b mod p = s	p, g, A, B	b, s

1.1 Man-in-the-middle

The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack. Here is a scenario of an attack :

- Eve intercepts Alice's public value and sends her own public value to Bob.
- When Bob transmits his public value, Eve substitutes it with his own and sends it to Alice.
- Eve and Alice thus agree on one shared key and Eve and Bob agree on another shared key.
- After this exchange, Eve simply decrypts any messages sent out by Alice or Bob, and then reads and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the other party.

1.2 Neural cryptography

Neural key exchange, which is based on the synchronization of two tree parity machines, should be a secure replacement for this method.

1.3 Tree Parity Machines

The tree parity machine is a special type of multi-layer feed-forward neural network.
It consists of one output neuron, K hidden neurons and K*N input neurons. Inputs to the network take 3 values:

$$x_{i,j} \in \{-1, 0, +1\}$$

The weights between input and hidden neurons take the values:

$$w_{i,j} \in \{-L, .., 0, .., +L\}$$

Output value of each hidden neuron is calculated as a sum of all multiplications of input neurons and these weights:

$$\sigma_i = sgn(\sum_{j=1}^{N} w_{i,j} x_{i,j})$$

sgn is a simple function, which returns -1,0 or 1:

$$sgn(x) = \begin{cases} -1 \text{ if x} < 0 \\ 0 \text{ if x} = 0 \\ +1 \text{ if x} > 0 \end{cases}$$

If the scalar product is 0, the output of the hidden neuron is mapped to -1 in order to ensure a binary output value. The output of neural network is then computed as the multiplication of all values produced by hidden elements:

$$\tau = \prod_{i=1}^{K} \sigma_i$$

Output of the tree parity machine is binary.

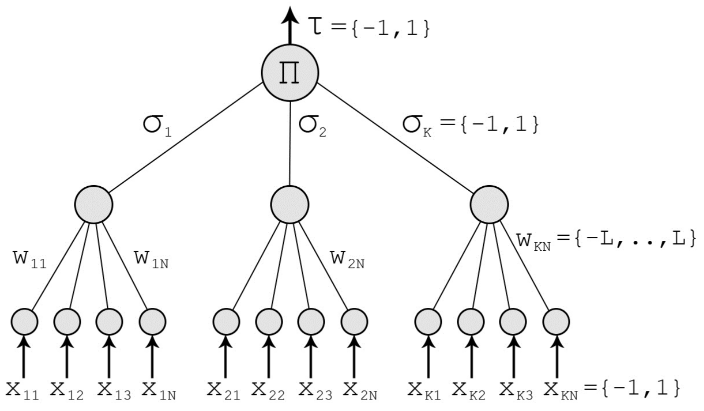

Figure 1: Tree Parity Machine

1.4 Algorithm

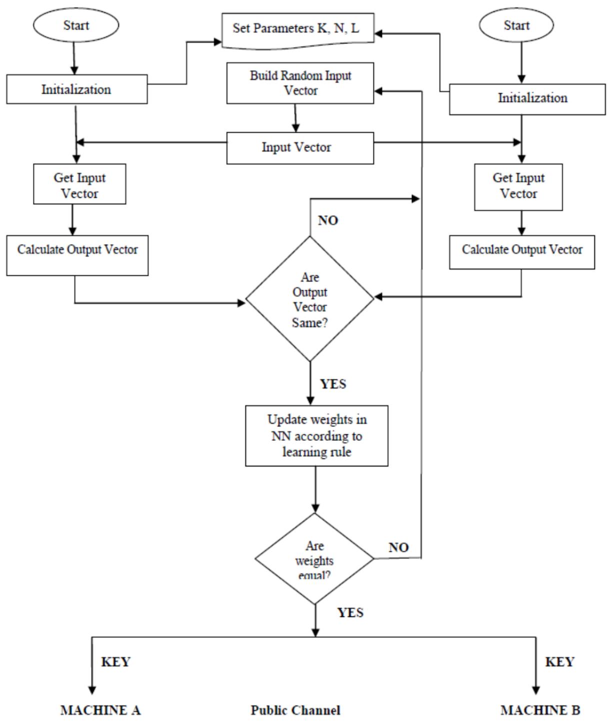

Figure 2: Neural key exchange algorithm

Each party (A and B) uses its own tree parity machine. Synchronization of the tree parity machines is achieved in these steps:

How do we update the weights? We update the weights only if the output values of the neural machines are equal. There are three different rules :

$w_{i,j}^{+} = g(w_{i,j} + x_{i,j}\tau\Theta(\sigma_i\tau)\Theta(\tau^A\tau^B))$	Hebbian learning rule
$w_{i,j}^{+} = g(w_{i,j} - x_{i,j}\tau\Theta(\sigma_i\tau)\Theta(\tau^A\tau^B))$	Anti-Hebbian learning rule
$w_{i,j}^{+} = g(w_{i,j} + x_{i,j}\tau\Theta(\sigma_i\tau)\Theta(\tau^A\tau^B))$	Random-walk learning rule

Here, Θ is a special function. $\Theta(a, b) = 0$ if a<>b; else $\Theta = 1$.
The g(...) function keeps the weight in the range - L..+L. x is the input vector and w is the weights vector.
After the machines are synchronized, their weights are equal: we can use them for constructing a shared key.

1.5 Pseudocode

We follow the following steps for neural key generation which is based on neural networks :

1. First of all determine the neural network parameters i.e.: k, the number of hidden layer units n, the input layer units for each hidden layer unit l, the range of synaptic weight values is done by the two machines A and B.

2. The network weights to be initialized randomly.

3. Repeat 4 to 7 until synchronization occurs..

4. The inputs of the hidden units are calculated.

5. The output bit is generated and exchanged between the two machines A and B.

6. If the output vectors of both the machines are same i.e. $\tau_A = \tau_B$ then the corresponding weights are modified using the Hebbian learning rule, Anti-Hebbian learning rule and Random-walk learning rule.

7. After complete synchronization, the synaptic weights are same for both the networks. And these weights are used as secret key.

1.6 Attacks

1.6.1 Brute force

By K hidden neurons, K*N input neurons and boundary of weights L, this gives $(2L + 1)^{KN}$ possibilities. These attacks are currently impossible.

1.6.2 Build his own Tree Parity Machine

TPM synchronization :

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

2. Output(A) = Output(B) = Output(E): All the three parties update weights in their tree parity machines.

3. Output(A) = Output(B) $\neq$ Output(E): Parties A and B update their tree parity machines, but the attacker can not do that. Because of this situation his learning is slower than the synchronization of parties A and B.

Eve cannot synchronize as fast as Alice and Bob and takes not recoverable delay. Depth of TPM increase this delay.

1.6.3 Quantum computers

"Neural key exchange protocol is not based on any number theory. It is based on the difference between unidirectional and bidirectional synchronization of neural networks. Therefore, something like the neural key exchange protocol could give rise to potentially faster key exchange schemes." [1]

References

[1] Neural cryptography. `https://en.wikipedia.org/wiki/Neural_cryptography`. Accessed: 2016-11-11.