On Demand Ciphering Engine Using Artificial Neural Network

Qutaiba Ali and Shefa Dawwd
Department of Computer Engineering, Mosul University, Iraq

Abstract: *In this paper, a new light weight highly secured ciphering engine creation methodology we called On Demand Ciphering Engine (ODCE) was suggested. The main feature of this method is that both, the ciphering engine and the secret key, are kept secrets and created by the user or the system administrator when initiating his transmission, then the involved sides exchange these secrets. The design methodology and structure of this system was described and prototyped using Artificial Neural Network (ANN) as the main building block in the system. Software and hardware implementations of the suggested system were performed to prove its practicability. Also, different experimental tests were achieved to determine the impact of the suggested method on both network delay and system performance.*

Keywords: *Network security, cryptography, neural network parallel implementation, ODCE, normalized ciphering engine.*

Received November 22, 2013; accepted October 26, 2014; published online August 22, 2015

1. Introduction

Data cryptography generally is the scrambling of the content of data, such as text, image, audio, video to make the data imperceptible during transmission or storage [26]. The major aim of cryptography is keeping data secure form illegal attackers [5]. In the 19th century, a well-known theory about the security principle of any encryption system has been proposed by Kirchhoff. This theory has become the most significant principle in designing a cryptosystem for researchers and engineers. Kirchhoff observed that the encryption algorithms are supposed to be known to the opponents [5]. Thus, the security of an encryption system should rely on the secrecy of the encryption/decryption key instead of the encryption algorithm itself. This concept was implemented in all known encryption algorithms, i.e., the algorithm construction is public while the key is kept secret and known only by the concerned parts [5, 26].

2. Literature Review

Security applications of Artificial Neural Network (ANN) is not a new topic and can be categorized into many sub-fields such as cryptanalysis, key-exchange, various ciphering systems, hash function, watermarking and Steganography.

Neural cryptanalysis work was conducted by Pandey and Mishra [19], they proposed an algorithm that offers an approach to attack ciphering algorithm based on the principle that any function could be reproduced by ANN. Also, Alallayah *et al*. [1] suggested the adoption of ANN as an ideal tool for black box system identification. They developed a mathematical black box model to construct the Neuro-Identifier.

The work on neural key exchange and authentication is another research area. Kanter *et al*. [12] stated that the neural key-exchange protocol does not employ number theory but is based on a synchronization of Neural Networks (NN) by mutual learning [12]. The architecture used is a two-layered perceptron, exemplified by a parity machine with K hidden units. The secret information of each entity is the initial values for the weights which were secret. Each network is then trained with the output of its partner. The work was extended to multilayer networks, parity machines [13]. In the field of key authentication, Gomathi and Nasira [9] provided a survey of various biometric based authentication systems based on ANN.

Many papers deal with the adoption of ANN in cryptographic systems. Shihab [22] suggested a decryption scheme and a public key creation based on a multi-layer NN that is trained by back-propagation learning algorithm. Yu and Cao [30], proposed an approach of encryption based on chaotic Hopfield ANN with time varying delay. Prabakaran *et al.* [20] proposed the Tree Parity Machines (TPM) in order to generate common secret key over the public channel where both sender and reception sides use ANN for cryptography purposes. Ali *et al*. [2], examined Artificial Spiking Neural Network (ASNN) which interconnects group of artificial neurons that used a mathematical model with the old block cipher. Dalkiran and Danisman [7] made use of ANN to model the dynamics of Chua's circuit in order to overcome the disadvantages of chaotic systems. Mohamed [18] proposed a cryptosystem based on hybrid approaches to provide multi-security services while taking the advantage of ANN computation power to serve data

confidentiality. Long [16] proposed a stream cipher algorithm based on one-way function of ANN. Shukla and Tiwari [23] suggested the use of two ANNs in order to overcome public key computational demands, one of them is based on n-state sequential machine and the other is chaotic ANN.

ANN was suggested to build Hash functions. Kulkami *et al.* [14] proposed an algorithm for one-way hash function construction based on a two Feed-Forward Neural Networks (FFNN) along with the Piece-Wise Linear (PWL) chaotic map. Sumangala *et al.* [25] suggested an algorithm for one way hash function construction based on two layers feed forward ANN along with the PWL chaotic map.

Other promising research fields of using ANN in security are watermarking and Steganography. Tsai *et al.* [28] proposed an intelligent audio watermarking method based on the characteristics of the HAS and the techniques of ANN in the DCT domain. Anitha *et al.* [3] developed a hybrid approach which comprises of ANN and S-DES encryption scheme which is used to detect the stago content in corporate mails.

3. The Suggested Method

In this paper, we suggest that both the ciphering engine and the key are both secret and created by the user when initiating his transmission, then the involved sides exchange these secrets. These secrets can be used in future communication attempts between the two parties or a new secrets is created every time a new session is initiated. However, this suggestion suffers from many problems and challenges when implemented using traditional methods. First, a complete ciphering engine (encryption and decryption sides) must be created according to user demands. Secondly, huge amount of data (which represent the ciphering engine code and the keys) must be transmitted between the two sides. Also, this method needs great deal of flexibility and synchronization between the two sides.

In order to respond to the above challenges, we suggest the use of ANN as an essential tool to build the suggested On Demand Ciphering Engine (ODCE). A proposed ciphering engine design tool is used to create a custom made ciphering engine together with its keys according to user demand. Our method can be described as shown in Figure 1:

1. The intended plaintext is prepared and entered into the ODCP. Then, the user takes his decision to design his own ciphering engine using either traditional ciphering methods or creates his new ciphering engine.
2. If the user chooses the traditional ciphering option, he can choose between two sub-options: A traditional single ciphering method OR (for better security) a serial/parallel combination of different methods. To clarify the second choice, Figure 2 shows an example of a new ciphering engine combined two ciphering methods (two 3DES and one AES with their keys) blocks connected together in a certain manner so that, the plaintext will go through more administrative and controlled scrambling process. In either case, the user must takes into accounts the ciphering parameters diversity among the different methods, such as block length, key length and its value.
3. On the other hand, if the user choose to create his own cryptographic system, he can follow the second option in which he will design his own ciphering engine according to fiestel type block ciphering systems [26]. Manual and automatic creation of the intended ciphering engine is available to the user. Automatic option permits the program to suggest a new ciphering engine structure, key value, no. of rounds, scrambling functions for each round, s-box, p-box, key scheduling, etc., on the other hand, the user may use manual design and makes use of a drag and drop Graphical User Interface (GUI) to determines the structure of the ciphering engine, key value, no. of rounds, scrambling functioning for each round, s-box, p-box, key scheduling, …, etc., it is worthwhile to mention that modifying traditional ciphering methods lead to produce different cipher text, e.g., changing the no. of rounds of any block ciphering method creates a new ciphering method. The user may benefit from this consequence to produce new ciphering engines in a short time.
4. The next step after any of the above steps is to generate the input/output relation table for both encryption and decryption sides. The generation process begins by giving a certain input plaintext value to the suggested ciphering engine then recording the output cipher text associated with this value. This procedure is repeated continuously so that, all the possible input/output pairs are included in the table. The time interval of this stage is determined mainly be the length of the input data block.
5. The next step is to convert the different forms of ciphering engines into a unified circuit. We suggest that ANN could play this role, i.e., be trained to emulate the behavior of the designed ciphering engine. As seen later in the next section, the ANN structure, i.e., the number of neurons for input, hidden and output layers depends mainly on the length of the input data block and hence, there is a certain structure for each possible block length. The next step is to train this NN using the previously mentioned input/output relation table (and output/input relation table in the decryption case). Learning procedure will continue until error value becomes less than a certain threshold then the weights and biases for both encryption and decryption ANNs are stored and prepared to be sent to the other side.

6. The final step is to send the ODCE, i.e., ANN structure, weights and biases, to the other side using a secure channel.

It is noted that although different ciphering engine structures could be obtained according to the user design, they are converted into a unified structured ANN. In this paper, we call this outcome as the Normalized Ciphering Engine (NCE).

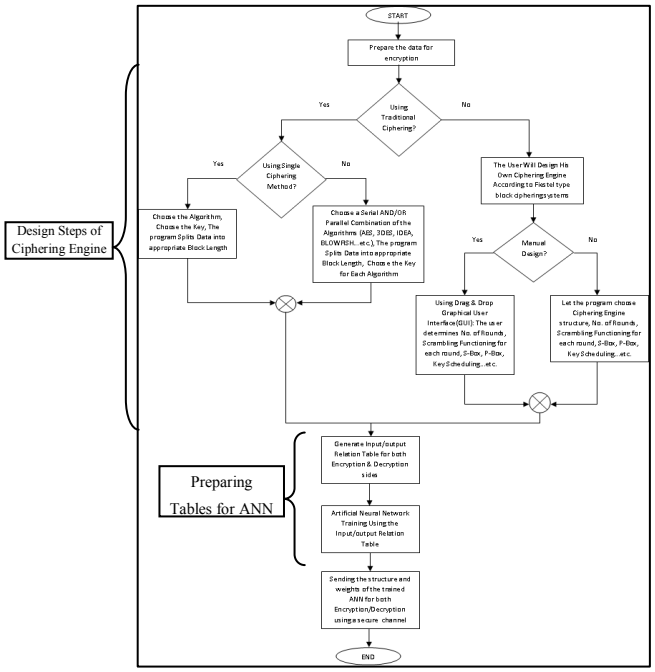


Figure 1. Flow chart of the suggested ODCE.

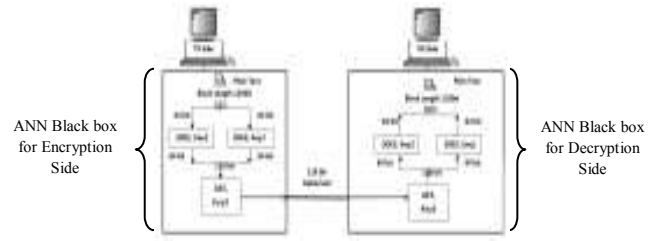


Figure 2. Example of a new ciphering engine.

There are several benefits obtained when adopting the NCE:

1. The reduction in the amount of the transferred data (which represent the ciphering engine secrets). In other words, instead of sending variable file sizes (ciphering engine source code) resulted from designing different ciphering engines, the same circuit (the ANN) is used every time.
2. Deterministic reserved resources in terms of the required transmission bandwidth, the allocated memory and CPU execution time for these tasks.
3. Higher flexibility in designing different ciphering engines with great deal of ease.

However, the adoption of ANN as the ultimate black box needs a special attention and care should be paid to prepare the ANN to undertake the cryptography tasks efficiently in a practical and visible fashion.

4. NCE Design Issues

ANNs provide a general, practical method for learning real-valued, discrete-valued and vector-valued functions from examples. NNs are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the connections between elements largely determine the network function. The NN can be trained to perform a particular function by adjusting the values of the connections (weights) between elements. NNs are trained, so that a particular input leads to a specific target output. The network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Many such input/target pairs are needed to train a network. NNs have been trained to perform complex functions in various fields, including pattern recognition [8], identification [15], classification [4], speech [29] and control systems [10]. NNs can also be trained to solve problems that are difficult for conventional computers or human.

The chosen type of ANN in this paper is the layered FFNN which are theoretical machines historically based on Rosenblatt's perceptron model, in which there is a layer of input units whose only role is to feed input patterns into the rest of the network. Next, there are one or more intermediate layers of neurons evaluating the same kind of function of the weighted sum of inputs, which, in turn, send it forward to units in the following layer. One of the most general problems in multilayer NNs is to find the connection strengths and thresholds which transform several known input patterns into their corresponding output patterns according to a given interpretation of inputs and outputs. The typical approach is a progressive learning process based on the principle of back-propagation, which leads to a solution by a lengthy relaxation search after a number of iterations large enough. Training continues on the training set until the error function reaches a certain minimum. If the minimum is set too high, the network might not be able to correctly classify a pattern. But if the minimum is set too low, the network will have difficulties in classifying noisy patterns [4, 10, 15]. Since, the crypto-system proposed in this paper is considered as an encoder, therefore, the best choice to build such encoder it to use the FFNN. Other types of NNs such as recurrent NN or unsupervised networks are targeted to be used for different applications such as in control, data clustering and other domains that irrelevant to the field of our interests.

Next, we will discuss some issues related to the adoption of ANN as the intended NCE. First of all, a new optimized design of ANN to serve as an encoder is presented. Secondly, we discuss the different approaches to choose the appropriate ANN size as a function of the input block length. Finally, we suggest

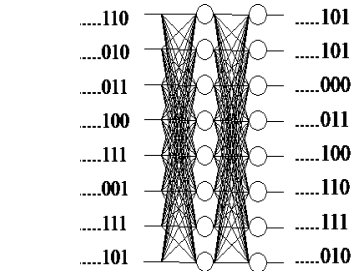
a suitable value to represent individual data units in both hardware and software implementations.

4.1. Encoder/Decoder Design Using ANN

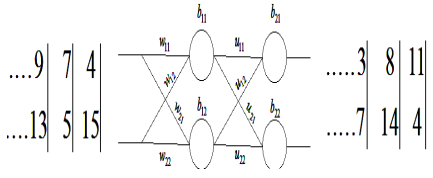
In order to implement our ODCE, NN must be used as an encoder, where a set of input pattern is to be associated to a set of desired patterns. Hence, each input pattern has only one associated output pattern. The binary encoder expects that either input or output patterns are binary values [17, 27]. The FFNN is used to model the encoder-decoder paradigm [1], where the input is first transformed into a typically lower-dimensional space (encoder) and then expanded to reproduce the initial data (decoder). The output of the hidden layer nodes represents the encoding pattern while the output layer nodes reproduce the input pattern. Our goal is to minimize and optimize the size and complexity of the ANN and hence, its performance. Unlike all previous works, we proposed that all inputs, weights and outputs of ANN are real values. The idea behind that is illustrated using the following example:

If it is required to map an 8bit input vector to an 8 bit output vector using FFNN of three (input-hidden-output) layers as shown in Figure 3-a.

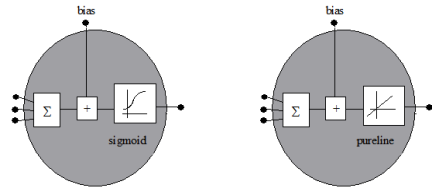
The ANN requires at least 8 hidden nodes (each has 8 connection weights) with 8 biases and 8 output nodes (each has 8 connection weights) with 8 biases, to map each of the 2^8 input patterns to its associated encoded output patterns. The weights and biases of either hidden or output layers are real values. The training error is reduced to an amount that makes all training samples have separable targets.



a) 8bit binary input.



b) Two digits input.



c) First and second layer node processing element for a and b.

Figure 3. The FFNN.

Our suggestion implies that the values of input that applied to ANN are represented by hexadecimal or decimal radix, rather than the binary radix see Figure 3-b. For example, in an 8bits word length number, each digit is converted to its associated radix as shown in Table 1.

Table 1. Binary to decimal conversion.

8Bits Binary Number	2Digits Decimal Number
0000_{bin} 0000_{bin}	0_{dec} 0_{dec}
0000_{bin} 0001_{bin}	0_{dec} 1_{dec}
0000_{bin} 0010_{bin}	0_{dec} 2_{dec}
.	.
1001_{bin} 0110_{bin}	9_{dec} 6_{dec}
.	.
.	.
1111_{bin} 1111_{bin}	15_{dec} 15_{dec}

4.2. Size of Encrypt ANN and Decrypt ANN

There is a number of theoretical results concerning the number of hidden layers in an ANN. Specifically, Hetcht-Nielsen [11] has shown that a network with two hidden layers can approximate any arbitrary nonlinear function and generate any complex decision region for classification problems. Later, Cybenko [6] showed that a single layer is enough to form a close approximation to any nonlinear decision boundary. (Furthermore, it was shown that one hidden layer is enough to approximate any continuous function with arbitrary accuracy-when the accuracy is determined by the number of nodes in the hidden layer; also, one hidden layer is enough to represent any Boolean function).

According to Cybenko's results, one hidden layer is used for either the encrypt ANN or the decrypt ANN. The number of nodes in the input, hidden, and output layers depends on two factors: The size of the input data block and the training performance that ensures classifying each plaintext code to its associated cipher text. The later factor can be realized by trial and error. Usually, one has to train different size networks and if they don't yield an acceptable solution, then they are discarded. This procedure is repeated until an appropriate network is found. Formal experience has shown that using the smallest network which can learn the task, is better for both practical and theoretical reasons. Smaller networks require less memory to store the connection weights and can be implemented in hardware more easily and economically. Training a smaller network usually requires less computation because each iteration is less computationally expensive. Smaller networks also, have short propagation delays from their inputs to their outputs. This is very important during the testing phase of the network, where fast responses are usually required.

As presented earlier, 8bits input data block requires 2 inputs, 2 hidden nodes and 2 output nodes. Table 2 below estimates the size of FFNN (either encrypt ANN or decrypt ANN) for different input data block sizes.

The encryption key K depends on the size of weights and biases. Thus, from the table above, one can see that the size of K increases with the increment

in both block and network sizes. Since, each weight or biases can be represented with *m* bytes, then *K* size (in bytes) can be formulated as:

$$K = [total\ \ no.of\ \ (weights + biases)] \times m] \quad (1)$$

Table 2. ANN size for different input data block sizes.

Input Block Size (Bit)	No. of Associated Digits	No. of Inputs (M)	No. of Hidden Nodes(N)	No. of Output Nodes (Q)	Total No. of Weights =N(M+Q)	Total No. of Biases =N+Q	K(Bytes), *M*=2
8	2	2	2	2	8	4	24
12	3	3	3	3	18	6	48
16	4	4	6	4	48	10	116
36	9	9	16	9	288	25	626
48	12	12	22	12	528	34	1124
64	16	16	30	16	960	46	2012
80	20	20	38	20	1520	58	1578
96	24	24	46	24	2208	70	4556
112	28	28	54	28	3024	82	6212
128	32	32	62	32	3968	94	8124

4.3. Data Representation

Data representation accuracy depends on the selection of *m* and either that weights or biases are represented in fixed point or floating point. In general, NNs have low-precision requirements, even though the exact specification is algorithmic and application dependent. Digital neuro-hardware can profit from this property by using fixed-point arithmetic with reduced precision which are less complex and less area consuming than floating-point arithmetic and helps to reduce system cost.

There are two parts in a fixed-point number, the integer part which is b_{ws-1} to b_4 and the other is the fractional part which is b_3 to b_0. If the base of this fixed-point number is β and it is a positive number, the decimal equivalent value can be calculated by:

$$v = b_{ws-1}\beta^{ws-5} + ,\ldots, + b_4 + b_3\beta^{-1} + b_2\beta^{-2} + b_1\beta^{-3} + b_0\beta^{-4} \quad (2)$$

Where *ws* is the precision.

If the base of fixed-point number is 2, the value is determined by what kind of representation is used (generally 2's complement is used).

For example, In an 8bit input data block NCE as shown in Figure 3-b, the network parameters are shown in Table 3. The precision *ws* and the radix point should be selected to attain the largest and lowest bias or weight values: 100.6401 and -44.7387.

Table 3. 8bit NCE network parameters.

Parameter	Real Value	Fixed Point, Ws=32, Radix Point Between Bit_{23} and Bit_{24}
w_{11}	-0.0011	11111111111111111000000000000000
w_{12}	0.0399	00000000000100000000000000000000
w_{21}	-0.0395	11111111111100000000000000000000
w_{22}	-0.0027	11111111111111110000000000000000
b_{11}	0.3037	00000000100000000000000000000000
b_{12}	-0.2795	11111111100000000000000000000000
u_{11}	6.8383	00000111000000000000000000000000
u_{12}	101.690	01100110000000000000000000000000
u_{21}	100.6401	01100101000000000000000000000000
u_{22}	2.8103	00000011000000000000000000000000
b_{21}	54.3852	00110110100000000000000000000000
b_{22}	-44.7387	11010011000000000000000000000000

Using floating point representation requires at least 32bit (4bytes) word length if, for example, IEEE standard 754-1985 format is used. In other words, *m* is fixed to 4bytes and *K* is a series of floating point numbers. Using this choice is preferable when the NCE is proposed to be implemented in software paradigm. The reason behind this is that a high precision representation, very large or very small numbers can be represented using scientific notation as follows:

$$n = \pm s \times b^e \quad (3)$$

Where +/- the sign of the number, *s* the significant or mantissa, *e* the exponent and *b* the base.

Here, we can represent numbers between -0.999×10^{99} and 0.999×10^{99} with a magnitude that ranges from 0.100×10^{-99} to 0.999×10^{99} with only 5 digits and two signs.

We can conclude that (*m*=4Bytes) is adequate for most software and hardware implementations of the intended NCE.

5. Prototyping The Model

This section deals with building a simple prototype model to demonstrate the ODCE main concepts as shown earlier in Figure 3. In the beginning we must determine the block length, which was set to (8bits). Then, we suggest the ciphering engine shown in Figure 4, which is a combination of different keys Simplified AES (SAES) ciphering method.

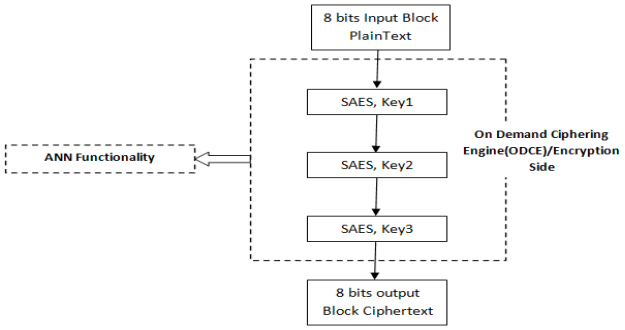


Figure 4. A proposed ciphering engine.

For software implementation (using MATLAB package), the (256 values) input/output relation table were generated and used to train the ANN shown earlier in Figure 3-b). The training time was (1.1s) on a (2.4 GHz Core i5, 4 GBytes RAM) PC. The resulted ANN composed the parameters presented in Table 3. The above procedure was repeated for the decryption side, hence, the total training time is about (2s) after which we obtain an 48bytes (to be sent to the other side) which were sufficient to represent the weights and biases for both encryption and decryption sides of the suggested NCE.

The tested NCE can also be implemented in hardware. In such a case, the floating point arithmetic computation of NN is more complex and area consuming than that achieved by fixed point computation. To achieve high precision weights and biases, the training is wholly implemented in software using floating point computations. After this and when weights and biases are fixed, neural hardware is used to model the NCE using fixed point computations. In

Figure 5, the weight matrix presentation of a simple NN (four nodes with four connections weight each) is shown in the middle, while the left side shows the conventional presentation of the same network. The rectangle *N* in the mid part of Figure 5 denotes the activation function of the neuron.

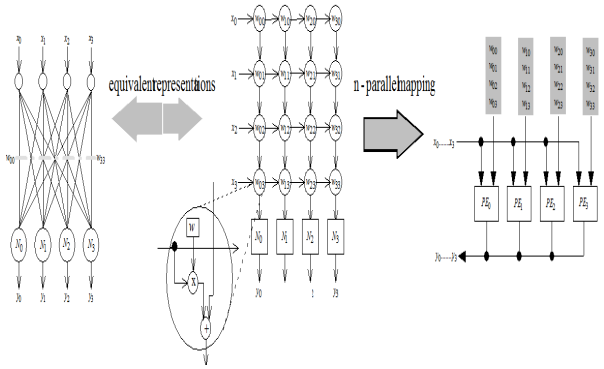


Figure 5. Hardware NN model: Conventional (left), weight matrix (middle), mapped neuron-parallel (right).

The right side of Figure 5 represents the mapping of the NN into a hardware model. The system architecture that is used to implement each of the two layers for Encrypt ANN or Decrypt ANN is shown in Figure 6. The circle w_{ij} represents the computation of the synapse: $y_i=w_{ij}\times x_j+y_{i-1}$ where y_{i-1} is the result from the proceeding synapse. Each processing element PE_i has an adder, a multiplier, an activation function and a local memory to store the node's weights vector and biases. The area utilization bottleneck is in the activation function of the 1st layer where sigmoid nonlinear pattern is to be achieved. To overcome this constrain, the sigmoid function was substituted by a piecewise linear function. Afterward, one multiplier is required for each sigmoid as an approximation to the nonlinear original function. While the output layer activation function is linear (pure line function), therefore the function weighted sum input will be the activation function output. Xilinx Spartan-3E FPGA of 500,000 gates is used as the implementation target. The intended 8bits NCE consumes 220 slices and 4 embedded multipliers for encryption side as shown in Figure 3-b and the same area utilization is consumed for decryption side. The designed network can operate on a 135MHz maximum frequency.

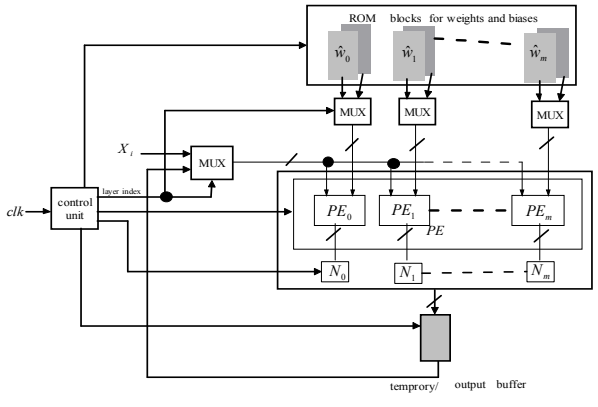


Figure 6. The complete architecture of the ciphering engine.

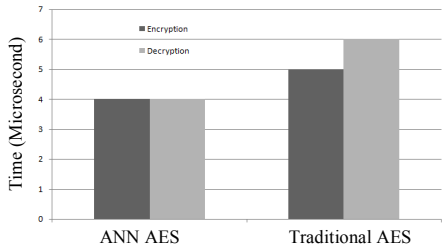
6. Security Analysis and Performance Investigation

In this section, we analyzed the security and the performance of the suggested ODCE. In this work, we did not focus on certain encryption algorithm, but instead, we suggest a new methodology to design, implement and distribute any ciphering engine. Traditional cryptanalysis techniques focus on the different methods taken by the attacker to discover the secret key used for encrypting the data [19] (as the encryption algorithm is known and public). However, the situation is different in our case in which both the encryption algorithm and its key are secret. Also, we suggest that encryption algorithm could be used for certain amount of time, then replaced by another secret algorithm according to user demands. Our primitive conclusion is that a traditional mathematical attack is not possible because the attacker does not have any referenced model (i.e., the ciphering engine) to be used for the calculation process (this claim will be proved in details in a forthcoming future paper). However, the attackers may take different ways focusing on discovering the internal structure on the ANN (e.g., number of layers) and the values of its weights and biases. This attack could be performed either by monitoring the traffic between the involved parties during ciphering engine distribution (handshaking) or measuring the electrical characteristics of the ANN hardware implementation.

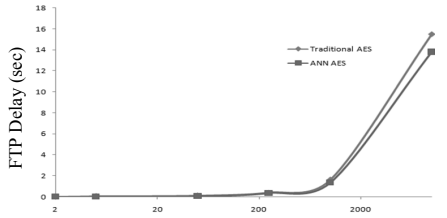
To solve the first problem, we can make use of the traditional secured key distribution methods, such as Diffie-Hellman protocol [26] or Key Distribution Center (KDC) [5], which can be modified to transfer the NCE data between the involved parties. Also, hardware implementation of the NCE must be resistible against different types on power analysis attacks using the known methods in this field [13].

In order to discover the practicability of the suggested method, we performed several experimental tests. We used two Corei5, 4GBytes RAM PCs connected together using a fast Ethernet switch. In the first experiment, we compared between traditional AES and an AES implemented in ANN (ANN AES) in order to measure their encryption and decryption delays to process a single 128bit block, as shown in Figure 7-a. It is obvious that ANN based AES proves its ability to outperform the speed of the traditional AES because less calculations are needed to perform its tasks. In the second experiment, we transferred encrypted files (with different lengths) between the two PCs using File Transfer Protocol (FTP), then measuring the total FTP delay. Again, we compared between the two AES versions as shown in Figure 7-b. In this experiment, ANN AES do better than traditional AES due to the less processing delay needed. In the last Experiment, we evaluate the efficiency of using ANN based ODCE (i.e., the NCE) to transfer the code

of the encryption algorithm itself. As mentioned before, in traditional ODCE we need to transfer the whole file containing the ciphering engine. However, the adoption of ANN ODCE eliminates the amount of transferred data to the weights and biases only which leads to the reduction of both file size and file transfer delay, as shown in Table 4.



a) Encryption and decryption delay.



b) FTP delay.

Figure 7. Experimental test results.

Table 4. ODCE code transfer delay.

Block Size (Bit)	Traditional ODCE		ANN ODCE	
	File Size of the C++ Code (Bytes)	FTP Delay (Sec.)	ANN Weights and Biases Size (Bytes)	FTP Delay (Sec.)
8(SAES)	15K	0.033	48	0.00024
64(3DES)	200K	0.22	4K	0.0045
128(AES)	100K	0.11	16K	0.018

From the above results, it is obvious that ANN proves its usefulness and practicability when used as the NCE in implementing the suggested ODCE.

7. ODCE Implementation Issues

As known, practicability is an essential condition to ensure the success of any suggested system. This section presents our vision to the future implementation of the suggested ODCE and discuses some concerns as follows:

1. The major challenge faces the current method is the long ANN training time. As shown earlier, 8bits input blocks takes about 2 seconds to achieve ANN training. When the input block and network sizes become larger, the training time overhead increases dramatically. For example, the training time for a 16 bit block (2^{16} values) input/output table consumes (187.85s) to achieve the same error accuracy which is equal to (0.0001). To speedup the training process of NNs, especially for very large training datasets, multithreaded and multicore CPUs with shared memory is used. The approach is to assign a part of the training dataset to each thread and process (train) them simultaneously. A training speedup of about 7x is achieved in [21] using 2 Quad-Core L5420 Intel CPUs (2.5GHz) with 8GByte RAM. Also, General Purpose Graphical Processing Unit (GPGPU) provided with massively multicore processors can be used for training. GPUs can be programmed using NVIDIAs CUDA C-language GPU programming environment. In [24], a 63x speedup is achieved using parallel training algorithm over sequential version when using a GPU type of the Tesla C1060 with 240 kernels and 4GB of memory that installed in a desktop Intel Core2 Duo E6750 @ 2.66 GHz and 4GB RAM.
2. In this paper, we suggest that the proposed ODCE could be realized by an online trusted center supplied with a high performance parallel processing servers (clusters) to perform the NN training. According to this suggestion, the user can (securely) access the web site of this ODCE center, making use of its resources to build his own ciphering engine (or making use of previously built ciphering engines by other users), performing a high speed NN training, then receiving the NN weights and biases which represent his new ciphering engine.
3. The other benefit of adopting the online ODCE center is to facilitate the ciphering engine distribution and management operations. Our proposal is to let the ODCE center to perform ciphering engine distribution tasks in a similar fashion to that of the well known KDC. Firstly, we assume that each user establishes a shared secret key with the ODCE center. Then, the user sends a request to the ODCE center, stating that he needs a certain ciphering engine to be used for communication with another user. The ODCE center informs the other user about this request and the intended ciphering engine is distributed between them. This secret ciphering engine that is established with the ODCE center is used to authenticate both of them to the ODCE center and to prevent the attackers from impersonating either of them.
4. Another important issue to be discussed is the handshaking procedure between the involved parties. In the conventional data exchanging operation, encryption method, key size and its value are distributed between the two sides. However, in the suggested ODCE, an additional data must be exchanged. These data includes NN structure (No. of input, hidden and output nodes) and the values of weights and biases. For example, for the ODCE shown later in Figure 3-b, these values are (2, 2, 2) for the NN structure and some values similar to those listed later in Table 3. For more security, these values can be sent after encrypting them using proper ciphering method.
5. One of the advantages of adopting ODCE is its backward compatibility with the known security

methods in the different TCP/IP protocol suite layers, especially network layer (IPSec protocol) and transport layer (SSL/TLS protocols).

8. Conclusions

This paper suggests the ODCE. Many benefits could be obtained from adopting this system, in front of them is the augmentation in privacy due to cryptographic algorithms secrecy. One of the consequences when adopting this suggestion is the infinite number of ciphering system could be created according to user demands and hence, adding his personal touch to secure his data. We believe that the suggested ODCE will make the attackers' task harder because new ciphering engines are created every time a secure transmission is initiated. We suggested ANN as the main building block in implementing ODCE in order create a unified-structure ciphering engine (we called normalized ciphering engine) for all encryption types and to reduce the amount of data to be transferred between the involved parties.

References

[1] Alallayah K., Amin M., Abd W. and Alhamami A., "Applying Neural Networks for Simplified Data Encryption Standard (SDES) Cipher System Cryptanalysis," *the International Arab Journal of Information Technology*, vol. 9, no. 2, pp. 163-169, 2012.

[2] Al-Omari K. and Sumari P., "Spiking Neurons with ASNN Based Methods for the Neural Block Cipher," *International Journal of Computer Science and Information Technology*, vol. 2, no. 4, pp. 138-148, 2010.

[3] Anitha P., Raj M., and Sivan S., "An Efficient Neural Network Based Algorithm for Detecting Content in Corporate Mails: A Web Based Steganalysis," *International Journal of Computer Science Issues*, vol. 9, no. 3, pp. 509-513, 2012.

[4] Bartlett P., "The Sample Complexity of Pattern Classification with Neural Networks: The Size of the Weights is More Important than the Size of the Network," *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 525-536, 1998.

[5] Cole E., Krutz R., and Conley J., *Network Security Bible*, Wiley Publishing Inc., 2005.

[6] Cybenko G., "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303-314, 1989.

[7] Dalkiran I. and Danisman K., "Artificial Neural Network Based Chaotic Generator for Cryptology," *SOURCE Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 18, no. 2, pp. 225, 2010.

[8] Fukushima K., "Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition," *Neural Networks*, vol. 1, no. 2, pp. 119-130, 1988.

[9] Gomathi P. and Nasira G., "A Survey on Biometrics Based Key Authentication Using Neural Network," *Global Journal of Computer Science and Technology*, vol. 11, no. 11, pp. 1-3, 2011.

[10] Gomi H. and Kawato M., "Neural Network Control For A Closed-Loop System Using Feedback-Error-Learning," *Neural Networks*, vol. 6, no. 7, pp. 933-946,1993.

[11] Hetcht-Nielsen R., "Theory of the Backpropagation Neural Networks," *in proceeding of International Joint Conference on Neural Networks*, Washington, pp. 593-605, 1989.

[12] Kanter I., Kinzel W., Kanter E., "Secure Exchange of Information by Synchronization of Neural Networks," available at: http://arxiv.org/pdf/cond-mat/0202112.pdf, last visited 2002.

[13] Kinzel W. and Kanter I., "Interacting Neural Networks and Cryptography," *Advances in Solid State Physics 42*, pp. 383, 2002.

[14] Kulkami V., Shaheen S. and Apte S., "Hash Function Implementation Using Artificial Neural Network," *the International Journal on Soft Computing*, vol. 1, no. 1, pp. 1-8, 2010.

[15] Narendra K. and Parthasarathy K., "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4-27, 1990.

[16] Long H., "Stream Cipher Method Based on Neural Network," *in Proceedings of National Conference on Information Technology and Computer Science Conference*, pp. 1-4, 2012.

[17] Masci J., Meier U., Cireşan D., and Schmidhuber J., "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction," available at: http://people.idsia.ch/~ciresan/data/icann2011.pdf, last visited 2011.

[18] Mohamed M., "Multi-Service Cryptography Scheme for Secure Data Communication," *International Journal of Computer Science and Network Security*, vol. 11, no. 7, pp. 148-153, 2011.

[19] Pandey S. and Mishra M., "Neural Cryptanalysis of Block Cipher," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 2, no. 5, pp. 50-52, 2012.

[20] Prabakaran N., Loganathan P., and Vivekanandan P., "Neural Cryptography with Multiple Transfers Functions and Multiple

Learning Rule," *International Journal of Soft Computing*, vol. 3, no. 3, pp. 177-181, 2008.

[21] Schuessler O. and Loyola D., "Parallel Training of Artificial Neural Networks Using Multithreaded and Multicore CPUs," *in Proceedings of the 10th International Conference, ICANNGA 2011*, Ljubljana, Slovenia, pp. 70-79 2011.

[22] Shihab K., "A Backpropagation Neural Network for Computer Network Security," *Journal of Computer Science*, vol. 2, no. 9, pp. 7-10, 2006.

[23] Shukla N. and Tiwari A., "An Empirical Investigation of Using ANN Based N-State Sequential Machine and Chaotic Neural Network in the Field of Cryptography," *Global Journal of Computer Science and Technology*, vol. 12, no. 10, pp. 1-11, 2012.

[24] Sierra-Canto X., Madera-Ramırez F., and Uc-Cetina V., "Parallel Training of a Back-Propagation Neural Network using CUDA," *in Proceedings of the 9th International Conference on Machine Learning and Applications*, Washington, pp. 307-312 2010.

[25] Sumangala G., Kulkarni V., Sali S., and Apte S., "Performance Analysis of SHA2 Algorithm with and without Using Artificial Neural Networks," *World Journal of Science and Technology*, vol. 1, no. 12, pp. 1-12, 2011.

[26] Tanenbaum A., *Computer Networks*, Prentice-Hall Publishing, 2006.

[27] Tonkes B., Blair A., Wiles J., "A Paradox of Neural Encoders and Decoders or Why Don't We Talk Backwards?," *in Proceedings of the 2nd Asia-Pacific Conference on Simulated Evolution and Learning, SEAL'98 Canberra*, Australia, pp. 357-364, 1999.

[28] Tsai H., Cheng J., and Yu P., "Audio Watermarking Based on HAS and Neural Networks in DCT," available at: http://cse.hcmut.edu.vn/~minhnguyen/NET/Computer%20Networks%20-%20A%20Tanenbaum%20-%205th%20edition.pdf, last visited 2003.

[29] Waibel Álexander., Hanazawa A., Hinton T., Geoffery E., Kiyohiro S., and Kevin L., "Phoneme Recognition using Time-Delay Neural Networks," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 3, pp. 328-339, 1989.

[30] Yu W. and Cao J., "Cryptography based on Delayed Chaotic Neural Networks," *Physical Letters Journal*, vol. 356, no. 4-5, pp. 333-338 2006.

Qutaiba Ali received BSC and MSC in Electrical Engineering in 1996 and 1999. He obtained PHD in computer Engineering 1n 2006. Since 2000, he joined Mosul University/Iraq as a faculty member and still there. His research interests include: Network simulation and modeling, real time and embedded systems. He published 4 international books and more than 56 papers (some of them are in ISI indexed journals) in his fields of interest. He participated (as TPC) in more than 40 IEEE International conferences and joined the editorial board of more than 15 international journals (IEEE, IET and Elsevier Journals).

Shefa Dawwd received the BSc degree in Electronic and Communication Engineering, the MSc and the PhD degree in Computer Engineering in 1991, 2000, and 2006, respectively. He is presently a faculty member (Associated Professor) in the computer engineering department/University of Mosul. His main research interests include image and signal processing and their hardware models, parallel computer architecture, hardware implementation and GPU based systems. He has authored more than 27 research papers. He has been an editorial member of several national and international journals.