Evaluating Efficiency of Artificial Neural Networks for Solving Symmetric Cryptography Issues

Denis Roenko[a]

[a]ITMO University, 49 Kronverksky Pr, Saint Petersburg, 197101, Russian Federation

Abstract

The paper is devoted to explore efficiency of artificial neural networks applied to the field of symmetric cryptography. It describes several applications of neural networks to cryptography in general to define concepts allowing to use them in this sphere. After the literature analysis on the neural cryptography the encryption algorithm based on recurrent neural network was defined and implemented in order to conduct experiments on its performance and statistical security comparing with modern symmetric encryption standards such as AES and GOST R 34.12-2015 "Magma". Based on the results of these experiments, which were presented in graphs and tables, conclusions on the efficiency of the algorithm were drawn, prospects for its improvement were noted which will become a basis for further research in neural cryptography.

Keywords

Symmetric encryption algorithm, artificial neural network, neural cryptography, algorithmic efficiency, statistical security analysis

1. Introduction

Applying cryptographic algorithms has always been among of the most important approaches to information protection. Cryptography studies existing and develops new methods of transforming information, guaranteeing that third parties will not be able to obtain confidential data.

Active usage of artificial neural networks (ANN) - algorithms, simulating a work of human brain to solve applied tasks, where standard approaches does not show high effectiveness, has become a logical stage of computer systems development. Such features of neural networks as mutual learning, self-learning and stochastic behavior allows to solve issues of public key cryptography, symmetric encryption, key distribution and generation of pseudo-random numbers. These properties are also contributed to the emergence of neural cryptography - branch of cryptography that studies usage of neural networks for encryption and cryptanalysis.

Therefore, it is important to determine the efficiency of neural network algorithms applied to the sphere of cryptography, particularly to symmetric encryption. To achieve this goal the following tasks should be solved:

Proceedings of the 12th Majorov International Conference on Software Engineering and Computer Systems, December 10–11, 2020, Online & Saint Petersburg, Russia

roenko.dv@gmail.com (D. Roenko)

0000-0002-9216-4146 (D. Roenko)

1. analyze related works and evaluate the application of neural networks in cryptography;
2. determine neural network encryption algorithm under the study;
3. define methods of evaluating performance and security of proposed algorithm;
4. conduct experiments to compare various neural network topologies and modern symmetric encryption standards, such as AES and GOST R 34.12-2015 "Magma";
5. make conclusions and designate prospects for further research.

There are some limitations and assumptions in this work. Performance metrics of the proposed encryption algorithm, which were obtained during the tests are limited by a processing power of the test environment (the laptop with Intel Core i5 7300HQ CPU, Nvidia GeForce GTX 1050 GPU and 8 Gb of RAM was used to conduct experiments). Also it can be supposed that architecture of the neural network, which was selected to implement encryption algorithm, provides best results for the given task.

Paper's structure corresponds to the order of performing the research tasks. The usage of neural network algorithms in some areas of cryptography is stated in Section 2. "Description of the algorithm" section is dedicated to explaining working principles of the algorithm and its mathematical background. Section 4 stands for definition of evaluation metrics, which were used to compare certain neural network topologies with existing encryption standards and analysis of the experimental results. Last section focuses on conclusions of the research and prospects for future work.

2. Application of neural networks in cryptography

Artificial neural networks have already shown their effectiveness in implementation of hashing algorithm which was described by L. Shiguo et el. [1] and key exchange protocol introduced by A. Singh and A. Nandal [2]. In this paper a short explanation of these algorithms was provided in order to identify key features of neural networks allowing to use them for cryptographic purposes.

2.1. Hashing function based on ANN

In fact, the simplest unit of a neural network, such as a neuron, is a one-way function. Artificial neuron has several input parameters and one output parameter, therefore it is easy to obtain the output value with known input values, but the reverse process is almost impossible to perform. Such property of a neuron as the irreversibility of the output value makes it suitable for creating hash functions based on it.

The model of the neural network, shown in Fig. 1 and used to implement the hashing function, is a three-layer feed-forward neural network and consists of the input layer, the hidden layer and the output layer. Input layer of the ANN gets 1024 bit vector of information, which is divided by 32 parts called data-pixels, each data-pixel contains 32 bits of information. Then input layer produces 8 outputs by calculating weighted sum from 4 data-pixels in each neuron, adding corresponding biases and applying chaotic function to each value. Hidden layer and output layer performs the same operations with their inputs. As a result, ANN gets a 128-bit

hash block H that consists of 4 data-pixels [1]. The following equation defines the output of neural network:

$$H = f(W_2 f(W_1 f(W_0 P + B_0) + B_1) + B_2) \tag{1}$$

P - input vector;
W_0, W_1, W_2 - weight vectors of corresponding layers;
B_0, B_1, B_2 - biases of corresponding layers;
f - chaotic function.

The hashing algorithm is built on the basis of ANN and key generator. The latter transforms user's key into a set of weights and biases for neural network and control parameters of chaotic functions for each layer (Fig. 2).

The scheme of hashing algorithm for all blocks of text is shown on Fig. 3. Function receives a hash block H from one block of message M than performs XOR operation between vector H and key vector K, which are both 128-bits long. The received value becomes a key for the next block hashing.

Analysis of this algorithm showed a high sensitivity of the output value to the input data and the user's key. In addition, the described algorithm is protected from "birthday" and "meet-in-the-middle" attacks. It is also worth noting that when carrying out parallel computations, the number of computational operations for the given algorithm becomes less than that of MD5 and SHA-1, as a result, the speed of calculating the function increases. However, to transform text using a neural network algorithm, the presence of a key is required, otherwise the unidirectionality of the algorithm is violated, while most hash functions work without a key [3].

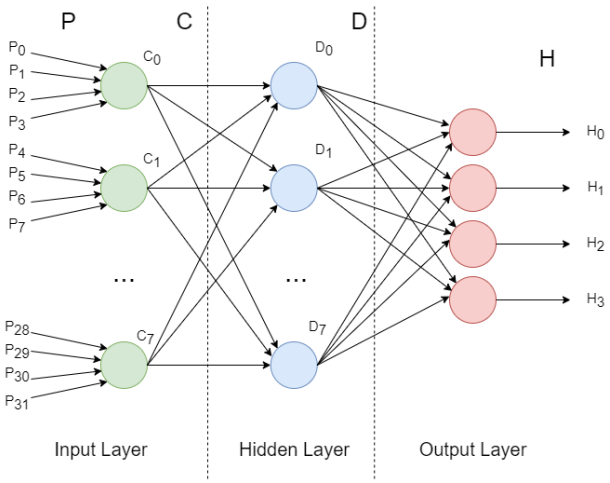

Figure 1: Neural network model used in the basis of the hashing algorithm

2.2. Neural key exchange protocol

The neural key exchange protocol is based on synchronization of two tree parity machines (TPM). It is a special type of multi-layer feed-forward neural network (Fig. 4). It consists of

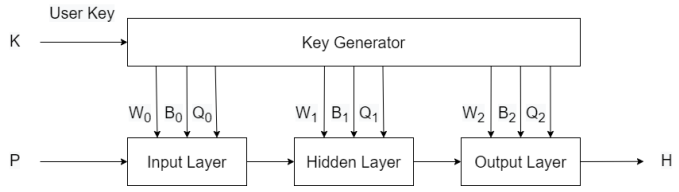

Figure 2: Scheme of hashing function for one block of message

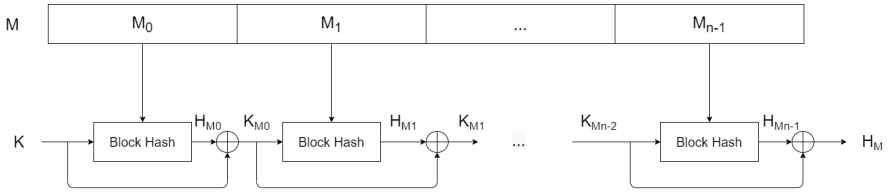

Figure 3: Scheme of hashing function for all message

one output neuron, K hidden neurons and $K \cdot N$ input neurons. Inputs of the network take 2 values: $x_{ij} \in \{-1, 1\}$. Weights between input and hidden layers takes following values: $w_{ij} \in \{-L, ..., 0..., L\}$. The output values from each hidden neuron are calculated as the sum of the products of input value and weight coefficient: $\sigma_{ij} = sgn(\sum_{j=1}^{N} x_{ij} w_{ij})$. The output value of the neural network is calculated as the product of all hidden neurons: $\tau = \prod_{i=1}^{K} \sigma_i$ [2].

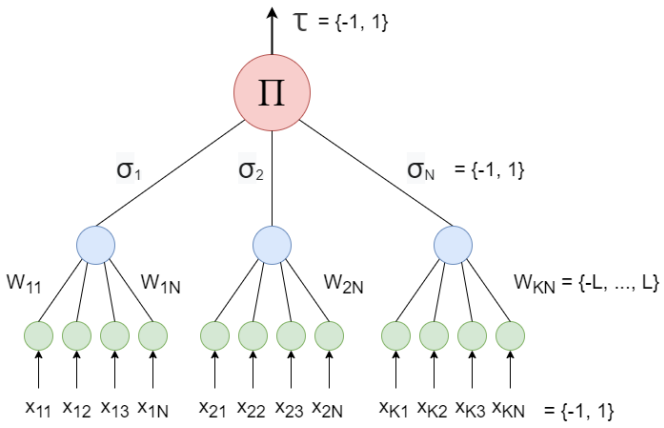

Figure 4: Scheme of tree parity machine

Each side of communication has its own TPM with the same configuration. Their synchronization occurs as follows:

1. random weight values are initialized for each party's tree parity machine;
2. the following steps should be done until synchronization is achieved:
   a) generation of random input vector X;

b) computation the values of hidden neurons;
c) computation of output neuron value;
d) compare the output values of two tree parity machines;

3. if outputs are different, return to the second step;
4. if outputs are the same, safe the weight values.

After the complete synchronization, both sides can use weights as a key. This method is known as bidirectional training [2].

3. Description of the algorithm

To test the effectiveness of applying ANN to implement symmetric block cipher, the algorithm proposed by M. Arvandi et el. [4] was chosen. It was selected due to the presence of a sufficient scientific base for its implementation, as well as vast prospects for its further improvement.

3.1. Neural network architecture and key sharing

Neural network architecture shown in Fig. 5 consists of one input, two hidden, one output layer and reverse relationship. The input of the neural network receives two-byte blocks of plaintext, combined with the two output values of the network from the previous step, then a procedure F_1 of calculating the values of neurons in the first hidden layer is performed. Next a procedure V_1 calculates the value of a single neuron in the second hidden layer. In this layer one neuron helps to get data mixing and applying nonlinear sigmoid function allows to achieve data replacement. At the end of iteration output values are calculated by F_2 procedure and sent to the next calculating iteration [4].

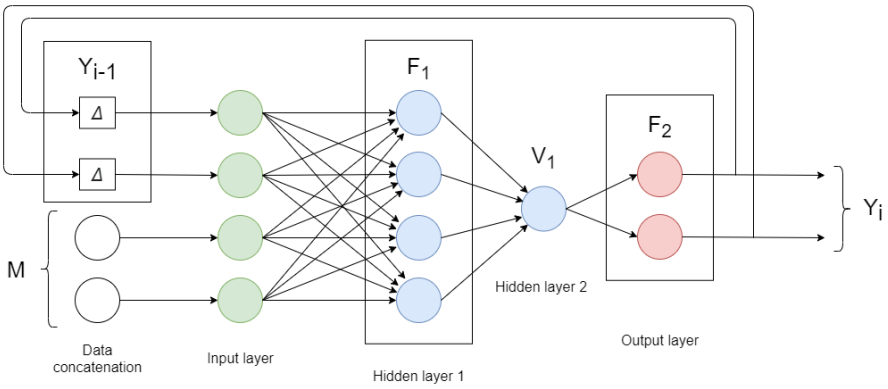

Figure 5: Basic architecture of recurrent neural network used in encryption algorithm

Before starting exchanging of encrypted messages users on both sides have to go through synchronization of weight coefficients using the same training dataset. This dataset is a set of vectors X and Y, which are input and expected output bytes respectively. It can be safely transmitted via open channel with asymmetric cryptography.

After receiving the dataset, each user performs learning process with their own copy of neural network to obtain weight coefficients, which will be a secret key for further encryption and decryption. Last output value M_0 from generation of a shared key becomes an initialization vector for encryption process [4].

It should be noted that learning process is performed with a predefined loss function (mean squared error) and optimizer (stochastic gradient decent). Another important moment is that initial weights and biases should be set by the algorithm and be the same for both parties.

3.2. Encryption process

Data encryption (Fig. 6) starts with concatenation of initialization vector M_0 with first plaintext block M_1. Resulting vector X_1 is used to obtain the value of intermediate neuron V_1 and neural network output Y_1. Then the difference E_1 between plaintext block and neural network output is calculated. Values V_1 and E_1 are the first block of ciphertext C_1 [4].

To encrypt the next block of plaintext, the concatenation of the values Y_1 and M_2 is sent to the network input. In a more general form, the encryption process can be represented by the following equations:

$$V_i = F_1(X_i) \tag{2}$$

$$Y_i = F_2(V_i) \tag{3}$$

$$E_i = M_i - Y_i \tag{4}$$

In addition, after encrypting each block of plaintext, it is possible to carry out one training iteration using M_i as the input vector and Y_i as the training goal. In this case, the weights will constantly change during the data encryption. This technique allows to improve the security of algorithm [4].

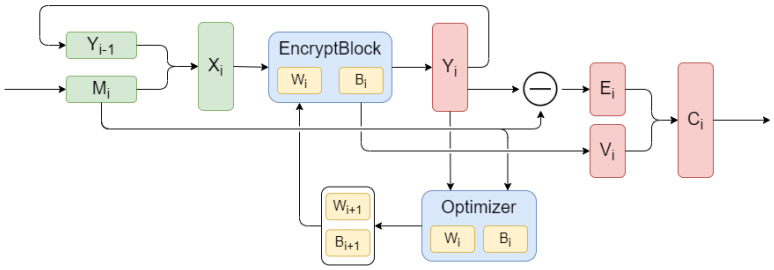

Figure 6: Scheme of encryption process

3.3. Decryption process

The decryption process (Fig. 7) starts when user on the other side of communication channel receives ciphertext blocks. Using value V_i from ciphertext it is possible to calculate neural network output Y_i:

$$Y_i = F_2(V_i). \tag{5}$$

With known Y_i and E_i the plaintext block can be calculated:

$$M_i = Y_i + E_i. \tag{6}$$

After recovering the plaintext block, one iteration of neural network training is performed in the same way as this operation was performed at the encryption stage.

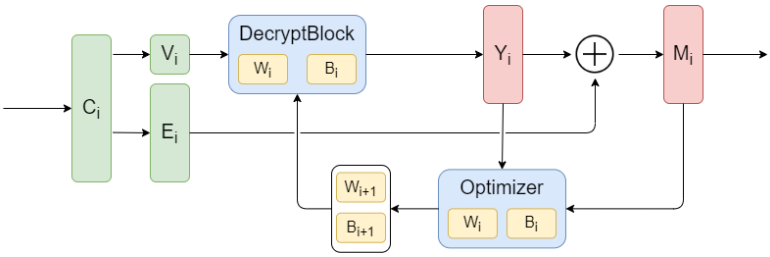

Figure 7: Scheme of decryption process

The security of the described encryption algorithm is based on the fact, that it will be difficult for an attacker to calculate the set synaptic weights and biases of neurons, without knowing the training dataset, which was used to synchronize the neural networks.

An important advantage of this cipher is the ability to easily regulate the level of security by increasing neurons in hidden layers or complicating their hierarchy. This advantage removes the limitation on the length of the secret key, which is present in all modern encryption algorithms. However, it is worth to avoid changes in the hidden layer with one neuron, otherwise there is a risk of losing the effect of mixing data [4].

4. Experiments and results

To conduct experiments, that will assess the performance and security of the neural network encryption algorithm, several neural network topologies were chosen:

- RNN-2 with layer configuration [4-4-1-2];
- RNN-4 with layer configuration [8-8-1-2-4];
- RNN-8 with layer configuration [16-16-1-2-4-8];
- RNN-12 with layer configuration [24-24-1-6-12];
- RNN-16 with layer configuration [32-32-1-8-16];
- RNN-32 with layer configuration [64-64-1-8-16-32].

The above topologies were also compared with the AES algorithm implemented in the `pycryptodome` library of Python programming language and GOST R 34.12-2015 "Magma" from the `pygost` library.

4.1. Execution time testing

Function `time` from Python standard library was used to estimate the time to perform encryption and decryption operations on a randomly generated 10,000-byte plaintext sequence. Due to the fact that measuring the execution time of programs always contains some anomalies, 5 measurements were made and average value of time consumption was found for each algorithm [5]. Experiment results are depicted in Table 1.

From the test results, it can be concluded, that deeper neural network architectures with a longer block length perform cryptographic transformations faster. Insofar as the main operation performed by neural network is matrix multiplication, so if network layers are small, it is necessary to perform much more operations, which greatly affects performance.

Table 1
Execution time tests for encryption algorithms

Algorithm	RNN-2	RNN-4	RNN-8	RNN-12	RNN-16	RNN-32	GOST "Magma"	AES
Test 1, sec.	17,212	8,964	4,596	3,135	2,438	1,350	0,518	0,001
Test 2, sec.	17,308	8,934	4,615	3,039	2,404	1,274	0,522	0,001
Test 3, sec.	17,309	8,987	4,545	3,037	2,453	1,298	0,524	0,001
Test 4, sec.	17,123	9,027	4,544	3,035	2,446	1,283	0,520	0,001
Test 5, sec.	17,359	8,989	4,558	3,041	2,429	1,291	0,523	0,001
Average time, sec.	17,26	8,98	4,57	3,06	2,43	1,30	0,52	0,001

Comparison with the GOST and AES algorithms showed, that there are prospects for creating a neural network cipher, which performance will be equal to or even faster than these algorithms. This can be achieved with optimization of the algorithm and applying hardware support, for example, performing calculations on graphics processing unit (GPU).

It should be noted that AES showed so high result because of hardware support of this algorithm at the level of special assembler instructions for Intel processor.

4.2. Memory consumption testing

To estimate the random access memory (RAM) consumption [5], the `tracemalloc` module from the standard Python library was used. The `get_traced_memory()` function shows the peak RAM consumption within a specific section of the program. The test results are shown in Table 2.

From the test results, it can be concluded that a deeper neural network topology consumes more system RAM, due to the fact that more values of weights and biases need to be stored.

The neural encryption algorithm shows significantly lower results by the criterion of system resources consumption comparing with AES and GOST standards, but its results is less than 3 MB, so it is not too critical for modern computers.

Table 2
RAM consumption tests for encryption algorithms

Algorithm	RAM consumption, MB
RNN-2	1,190
RNN-4	1,304
RNN-8	1,356
RNN-12	1,419
RNN-16	1,482
RNN-32	2,515
GOST "Magma"	0,010
AES	0,018

4.3. Statistical security assessment

To determine the security degree of a block cipher, a statistical assessment of ciphertexts was applied. It shows how close the ciphertext produced by the algorithm to a truly random sequence.

Statistical security test was performed by plotting graphs of ciphertexts and calculation of their entropy with Shannon's formula [6], where X - a set of non-repeating values x_i of ciphertext, $p(x_i)$ - the probability of specific value from ciphertext to occur, n - amount of these values:

$$H(X) = \sum_{i=1}^{n} p(x_i) \log_2 \frac{1}{p(x_i)} \tag{7}$$

The minimum value of entropy is equal to zero and characterized the sequence of identical symbols. The entropy takes maximum value when all symbols occurs with the same probability and equals to 8 for the alphabet of symbols from 0 to 255, that is all possible values for a byte of information.

The tests were performed on a plaintext of 600 characters, which is a cyclic repetition of 40 "A" symbols and 20 "z" characters. Entropy value of such text is low and approximately equal to 0,918 (Fig. 8).

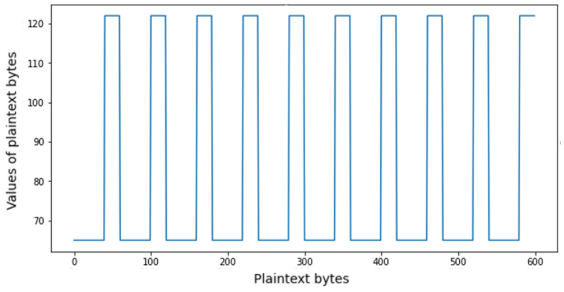

Figure 8: Statistical graph of plaintext

From the test results (Table 3), it can concluded that the topologies with a smaller block size

produces a ciphertext with a higher entropy value. It can also be seen on the graphs of the corresponding ciphertexts (Fig. 9, Fig. 10).

Table 3
Results of statistical security tests

Algorithm	Entropy
RNN-2	7,010
RNN-4	7,129
RNN-8	6,620
RNN-12	6,604
RNN-16, RNN-32	–
AES-256 (CBC)	7,611
GOST "Magma" (CBC)	7,640
Random number generator	7,696

For large block size configurations such as RNN-16 and RNN-32, entropy values were not obtained, as in these cases the problem of fading gradient was noticed. This means that the weights of initial layers of the network do not change up to the layer with one neuron. Obviously, this is a significant flaw in their security, as half of the network is not trained.

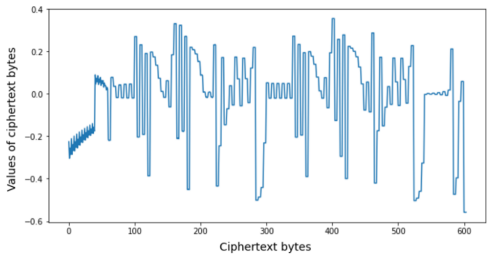

Figure 9: Graph of RNN-4 ciphertext

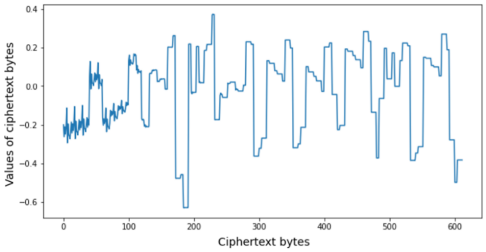

Figure 10: Graph of RNN-12 ciphertext

In comparison with existing encryption standards, it was found that more preferable configurations in terms of operating speed, such as RNN-8 and RNN-12 has lower entropy of ciphertexts than AES (Fig. 11) and GOST R 34.12-2015, while the ciphertext entropy of the latter is very close to a pseudo-random sequence.

5. Conclusions and prospects for further research

During analysis and testing of the encryption algorithm based on neural network, its main advantages and disadvantages were noted. The strengths of the neural network algorithm in comparison with the current cryptographic standards for block ciphers are:

- there is no restrictions on the length of the secret key;
- easy adjustment the security parameters of the algorithm.

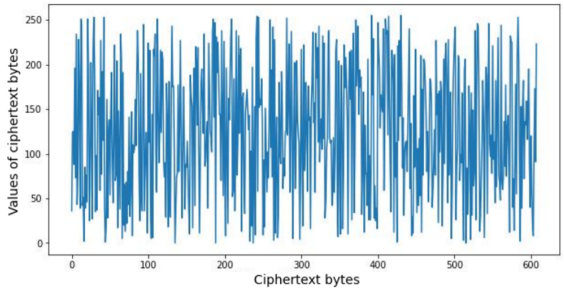

Figure 11: Graph of AES ciphertext

At the same time, there are a number of negative aspects:

- long execution time and the presence of additional time spent on the process of synchronizing the weight coefficients;
- the presence of a fading gradient problem, which makes it difficult to design an algorithm with deep neural network topologies;
- the security level is lower in comparison with current cryptographic standards.

The elimination of these drawbacks can be the basis for further research of the neural network algorithm for block encryption. Several potential directions for its improvement were identified.

Possible solution for the problem of low encryption and decryption speed can be optimization of the algorithm and applying hardware support to improve its performance. To solve the fading gradient problem, it is necessary to test the algorithm using other activation functions, such as Leaky ReLU. To increase degree of ciphertext randomness it is possible to add multiple rounds of data transformation and permutation operations, as it implemented in the AES and GOST R 34.12-2015 standards.

Conducted research showed that neural networks can be successfully used to implement block ciphers with good efficiency, but the results of experiments confirmed that neural network algorithms are not yet able to compete with the current standards in the field of symmetric cryptography. It is required to conduct more research in order to get better results.

References

[1] S. Lian, J. Sun, Z. Wang, One-way hash function based on neural network (2007). URL: https://arxiv.org/abs/0707.4032.

[2] A. Singh, A. Nandal, Neural cryptography for secret key exchange and encryption with AES, International Journal of Advanced Research in Computer Science and Software Engineering (2013) 367–381.

[3] Neural networks in cryptography, 2015. URL: http://cryptowiki.net/index.php?title=Neural_networks_in_cryptography.

[4] M. Arvandi, S. Wu, A. Sadeghian, W. W. Melek, I. Woungang, Symmetric cipher design using recurrent neural networks, in: The 2006 IEEE International Joint Conference on Neural Network Proceedings, 2006, pp. 2039–2046. doi:10.1109/IJCNN.2006.246972.

[5] Algorithm efficiency, 2017. URL: http://www.cs.kent.edu/~durand/CS2/Notes/03_Algs/ds_alg_efficiency.html.

[6] Y. Wu, Y. Zhou, G. Saveriades, S. Agaian, J. P. Noonan, P. Natarajan, Local shannon entropy measure with statistical tests for image randomness, Information Sciences (2013) 323–342. doi:10.1016/j.ins.2012.07.049.