3-Party Adversarial Cryptography

Ishak Meraouche[1(✉)], Sabyasachi Dutta[2], and Kouichi Sakurai[1]

[1] Kyushu University, Fukuoka, Japan
meraouche.ishak.768@s.kyushu-u.ac.jp, sakurai@inf.kyushu-u.ac.jp
[2] University of Calgary, Calgary, Canada
saby.math@gmail.com

Abstract. The domain of Artificial Intelligence (AI) has seen an outstanding growth during the last two decades. It has proven its efficiency in handling complex domains including speech recognition, image recognition and many more. One interesting and evolving branch that was put forward years ago but have seen a good growth only during the past few years is encryption using AI. After Google announced that it has succeeded teaching neural networks encryption in the presence of Eavesdroppers, research in this particular area has seen a rapid spread of interest among different researchers all over the world to develop new Neural Networks capable of operating different cryptographic tasks. In this paper, we take initial steps to achieve secure communication among more than two parties using neural network based encryption. We forward the idea of two party symmetric encryption scheme of Google to a multi party Encryption scheme. In this paper we will focus on a 3-Party case.

Keywords: Deep learning · Neural networks · Cryptography · Symmetric key encryption

1 Introduction

Digital communication world heavily relies on cryptographic techniques for resolving security threats. Provably secure cryptographic protocols have become basic building blocks for communication engineering where the channels are most often insecure. However, provable security incurs a huge overhead in the communication. Several faster cryptographic algorithms are proposed but in most cases they are not provably secure and often they are broken. Nowadays researchers are trying to develop cryptographic techniques based on artificial neural networks. There are quite a few works proposing basic encryption schemes based on neural networks (NN) [9,13,15] and also attacks [11].

One very interesting proposal was put forward by Google Brain Researchers [1] in 2016. In their proposal, two neural networks were able to learn (on their own) how to communicate securely in the presence of an eavesdropper. This work initiated a flow of research along this direction and even got improved in [3]. The idea of these works is a follow up of the work by Goodfellow et al. [5,6]. Recently,

L. Barolli et al. (Eds.): EIDWT 2020, LNDECT 47, pp. 247–258, 2020.
https://doi.org/10.1007/978-3-030-39746-3_27

Zhou et al. [16] initiated the study of security analysis of neural network based encryption schemes (in particular [1]) by using several statistical tests e.g. χ^2 test, Kolmogorov-Smirnov test.

The work done by Google Brain researchers is considered a big step in the world of cryptography as it ensures secure communication between two parties. The security is in the fact that the two neural networks are not taught any specific encryption method, all relies on the synchronisation of the two neural networks and their structure. It has been proved in [1] that even with a neural network with a similar structure, it is difficult to decrypt the cipher text by cryptanalysis.

Our aim in this paper is doing a similar work but among three parties instead of just two. As synchronising multiple parties might be a bit challenging, we study a single case and try to synchronize three neural networks so they communicate securely in different scenarios. In other words, we synchronize three neural networks to communicate securely as in [1] in the presence of an eavesdropper with the same neural network structure while trying to improve the accuracy and preserving the training time.

2 Related Works

2.1 Classical Symmetric-Key Cryptography

Symmetric cryptography is widely used in the world of cryptography. Usually, there are two parties Alice and Bob who are willing to communicate with eachother. These two parties share in advance, a key K and a pre-fixed encryption/decryption algorithm. Alice uses the plain text P and the key K as inputs to the encryption algorithm, process it and output the cipher text C. Once C is sent to Bob, he will process it with the decryption algorithm (which also depends on K) to output the original plain text P. Many existing encryption algorithms are widely used in networking security.A good and widely used algorithm is the Rijndael Algorithm (The advanced encryption standard) Approved by the NIST as the Advanced Encryption standard [8]. Other very efficient algorithms are Serpent [2] and Blowfish [12]. Symmetric key encryption schemes are very fast and incur a low communication overhead.

2.2 The Two Party Neural Network Based Encryption by Google Brain [1]

The Training Model

In this model, Alice and Bob are both neural networks sharing a secret key K willing to communicate securely. They have the same neural network structure but will be initialized with random parameters. In order to train the neural networks, a third neural network (called Eve) is added to the scenario and she plays the role of an eavesdropper. Eve has also the same neural network structure but does not know the key K. Figure 1 shows the schematic of the model.

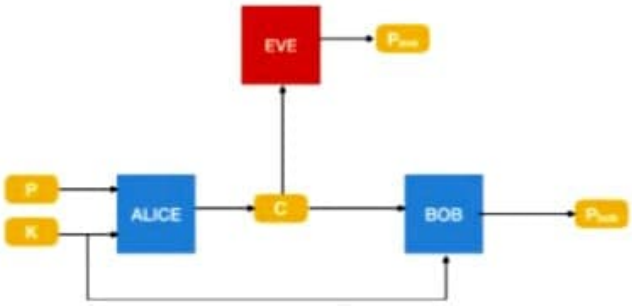

Fig. 1. Two party encryption using neural networks

Alice has as an input the plaintext P and the secret key K. They are used as inputs to Alice's neural network. After being processed, the neural network outputs the cipher text C.

C is sent to Bob who uses it as an input to his neural network along the secret key K and output $\mathrm{P_{Bob}}$.

Eve will intercept the ciphertext C and use it as an input to her neural network but without the secret key K.

In order to train the neural networks Alice, Bob and Eve are trained with the following parameters respectively: $\theta_\mathrm{A}, \theta_\mathrm{B}, \theta_\mathrm{E}$. The function $E_\mathbf{A}(\theta_\mathbf{A}, P, K)$ represents the encryption function for Alice with output as $\mathbf{C}$ and $D_\mathbf{B}(\theta_\mathbf{B}, C, K)$ represents the decryption algorithm of Bob with output as $\mathbf{P_{Bob}}$. Regarding Eve, she has the following decryption function: $D_\mathbf{E}(\theta_\mathbf{E}, C)$ with output as $\mathrm{P_E}$.

In order to calculate the distance between the original plaintext and each deciphered text the L_1 metric is used:

$$d(P, P') = \sum_{i=0}^{N} \mid P_i - P'_i \mid$$

The loss function for Eve can be defined as follow:

$$L_E(\theta_A, \theta_E, P, K) = d(P, D_E(\theta_E, E_A(\theta_A, P, K)))$$

As we can see, L_E measures the error in the decryption process of Eve. By taking an expected value, a loss function over the distribution of the plain texts and keys can be defined as follow:

$$L_E(\theta_A, \theta_E, P, K) = E_{P,K}(d(P, D_E(\theta_E, E_A(\theta_A, P, K))))$$

The optimal Eve is obtained by minimizing the loss:

$$O(E)(\theta_A) = argmin_{\theta_E}(L_E(\theta_A, \theta_E, P, K))$$

Similarly, the definition of the loss function for Bob is as follow:

$$L_B(\theta_A, \theta_B, P, K) = d(P, D_B(\theta_B, E_A(\theta_A, P, K), K))$$

The expected value over a distribution of plain texts and keys gives:

$$L_B(\theta_A, \theta_B, P, K) = E_{P,K}(d(P, D_E(\theta_E, E_A(\theta_A, P, K)), K))$$

With the above, the optimal values for Alice and Bob are as follow:

$$L_{AB}(\theta_A, \theta_B) = L_B(\theta_A, \theta_B) - L_E(\theta_A, O_E(\theta_A))$$

Training Process
Using the loss function defined above ($L_{AB}(\theta_A, \theta_B)$), first initialize neural networks with random parameters ($\theta_A, \theta_B, \theta_C$) and then update them after every iteration till minimization is done.

Once the minimization is done, Alice and Bob are said to be synchronized and can communicate securely even in the presence of eavesdroppers as their final parameters are optimal and can guarantee high decryption error for eavesdroppers and low decryption error for members of the same party.

How Is the Data Actually Being Encrypted?

The way data is being encrypted might not be so clear from the perspective of the calculation of the minimum distance.

Concretely, each neural network's input is the concatenation of the plain text with the secret key. After this, the bit stream will go through every layer of the neural network in the following order: First a fully connected layer that will output a bit stream of the same size as the input. Next, we will have a serie of Convolutional layers that will mix the bits of the input vector and output a cipher text. You will notice that the Mix & Transform method is being applied to the input vector in order to output a cipher text. What is happening inside the neural network when it receives the input vector is a kind of black-box and cannot concretely be analysed.

2.3 Steganography Based on Adversarial Neural Networks

Besides cryptography, Steganography is also an interesting field. It aims to hide an information inside another one. Usually we hide a text message or a small image inside another one using a secret key and make it impossible to extract the hidden information without the key.

Recent works show methods to do secure steganography based on adversarial neural networks. We have for example the work done in [10] that shows a way to train the cover image to generate the secret information without modifying the original image. Another method for steganography is also shown in [7].

A very recent paper [14] also features a method to do steganography using neural networks. In their work, there are two neural networks Alice and Bob sharing a key K and willing to communicate in the presence of the eavesdropper Eve. Alice trains to hide a message inside a cover image so that only Bob can extract it.

2.4 Over the Air Communication

Another promising work has been done in [4] where a full implementation of a communication over the air (radio frequencies) has been done based on deep learning.

Our Contribution
In traditional symmetric key encryption, if any number of parties hold the same key then they are able to communicate among themselves securely. The order in which the parties communicate does not matter. We ask the same question in case of NN based symmetric key encryption when all the parties share the common secret state/key. We consider the case of three neural networks who are to be trained to achieve a symmetric encryption rule. We consider three possible scenarios to train the neural networks. The challenging issue is to get a low error rate for decryption while and after training. We demonstrate that it is possible to include a third neural network to the system of Alice and Bob, called Charlie and that they will be able to synchronize and communicate securely in the presence of eavesdroppers. In the first scenario, there is no hindrance in the communication among the parties. In the last two scenarios, Alice has to act as an intermediary for secure communication to happen.

3 Results

Our aim is to synchronize three neural networks. Alice, Bob and Charlie are the parties who want to communicate securely even in the presence of eavesdroppers. Alice, Bob and Charlie share the same secret key K and have the same neural network structure. As before, Eve has access to any ciphertext communicated among the three parties but do not have access to the secret key K.

As in the two-party model and in order to synchronize our neural networks we need a loss function.

As we added Charlie to the group, we need to add his loss function too, which is defined as follows:

$$L_C(\theta_A, \theta_C, P, K) = d(P, D_C(\theta_C, E_A(\theta_A, P, K), K))$$

The expectation over a distribution on plain texts and keys is defined as follows:

$$L_C(\theta_A, \theta_C, P, K) = E_{P,K}(d(P, D_C(\theta_C, E_A(\theta_A, P, K)), K))$$

The final loss function is defined as follows:

$$L_{ABC}(\theta_A, \theta_B, \theta_C) = L_B(\theta_A, \theta_B, \theta_C) - L_E(\theta_A, O_E(\theta_A))$$

3.1 First Scenario

Training Process

In this scenario we suppose that one Neural network is sending messages to the two other neural networks in the party. We choose Alice as the sender and Bob, Charlie as receivers.

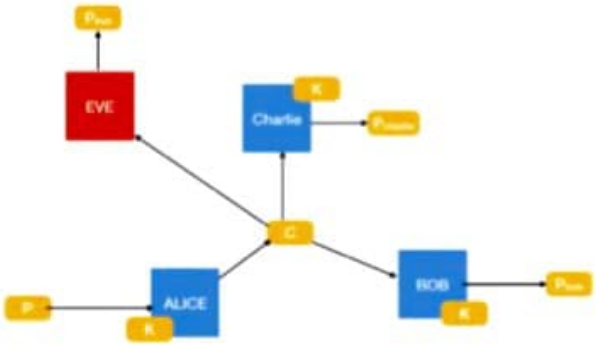

Fig. 2. Setup of experiment 1: In this setup Alice acts as a leader and sends messages to both Bob & Charlie so that the parameters of the NNs are optimized.

Experiment

We make Alice generate a plain text P of 32 bits, encrypting it and sending it to both Bob and Charlie.

We make Bob and Charlie decrypt the Cipher text and output $P_{\text{Bob}}, P_{\text{charlie}}$ respectively. Eve intercepts and tries to decrypt the cipher text C and outputs P_{eve}.

In every iteration, we take calculate the minimum distance between the original plaintext and decrypted ones respectively: $P_{\text{Bob}}, P_{\text{charlie}}$ and P_{eve}. Figure 2 shows the plot of the decryption error made during each iteration (We train by a set of epochs consisting of 2000 iterations each.) (Fig. 3).

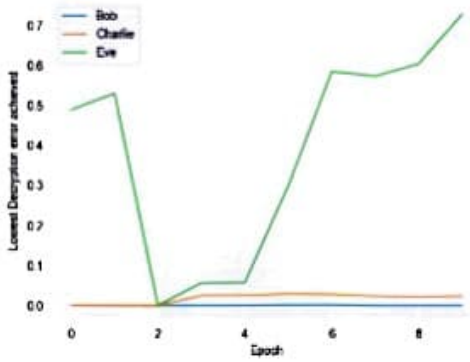

Fig. 3. Test results of experiment 1

Analysis

In this experiment, we can see that Bob and Charlie start with almost 100% accuracy and about 50% accuracy for Eve. It all goes good but Eve quickly learns how to decrypt the messages made between Alice and Bob. Alice starting from the second epoch starts making her encryption method harder and as a result, Eve's decryption error goes up to 70% by the end of the epoch. The change made by Alice in the encryption process results in decryption error of both Bob and Charlie. The accuracy of decryption is lowered to about 98%.

3.2 64-Bit Message Length

We noticed that rising the message and key length to 64 bits gives much more accuracy to Alice and Bob while training. However and most importantly, Eve's accuracy never goes beyond 50%. The training results are shown in Fig. 4.

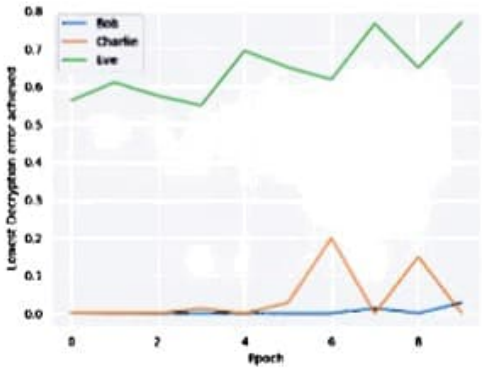

Fig. 4. Test results of experiment 1 with 64 bits key and message length

3.3 Second Scenario

Training Process

The Second Scenario consists into making two pairs of neural networks. Concretely, Alice will communicate with Charlie and Charlie will communicate with Bob. There is no direct communication between Alice and Bob.

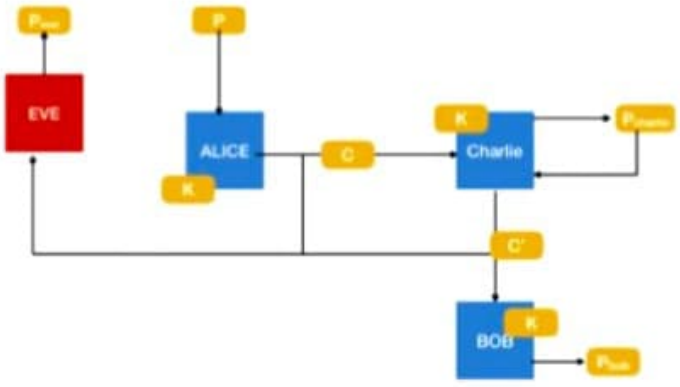

Fig. 5. Setup of the second scenario

Experiment

In Fig. 5, Alice generates a ciphertext and sends it to Charlie. Charlie deciphers the ciphertext, encrypts it again with the same method and send the new ciphertext to Bob. Bob decrypts the new ciphertext to output P_{Bob}. Eve has access to both of the communications. We train and calculate the error between the original ciphertext and the three outputs: P_{Bob}, $P_{charlie}$ and P_{eve}. We plot the decryption error made by every Charlie, Bob and Eve as shown in Fig. 6.

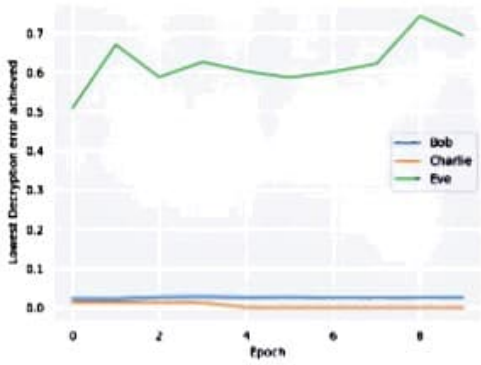

Fig. 6. Second scenario experimental results

Analysis

We see that Charlie has almost 100% accuracy in decryption after 4 epochs. Bob though has about 97.5% accuracy after the decryption process.

After synchronizing Alice with Charlie and Charlie with Bob, we tried to make Alice communicate directly with Bob with the synchronized parameters but unfortunately the communication was not accurate and therefore this is not possible. Figure 7 shows the decryption error achieved which is around 50%.

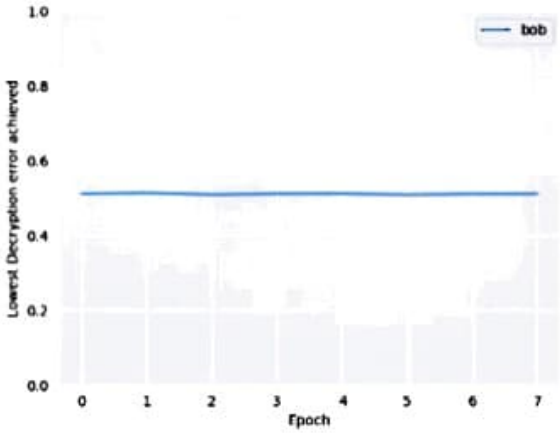

Fig. 7. Error made by Bob when Alice communicates directly with him without going through Charlie

3.4 Third Scenario

Training Process

The third and last scenario is to make Alice synchronize with Bob and Charlie separately. Concretely, Alice will synchronize with a set of parameters with Charlie and another one to communicate with Bob. In this scenario, if Bob wants to communicate with Charlie, he will need to use Alice as a bridge. Figure 8 shows the model of the third scenario.

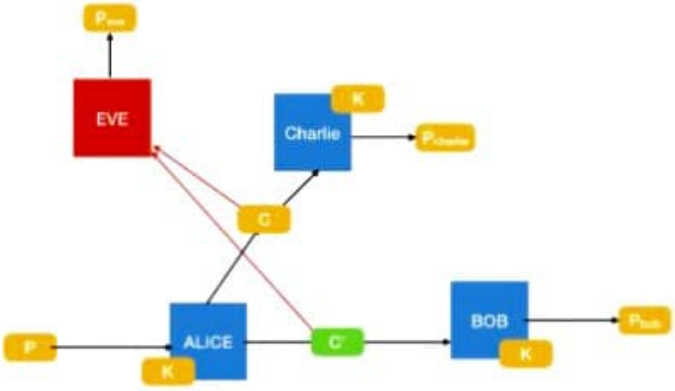

Fig. 8. Setup of the third scenario

Experiment

We make Alice generate two set of parameters for her neural network. When Alice generates a plain text P, when sending this plain text to Bob, she will use the first set of parameters. When sending a message to Charlie, she will use the second set of parameters. The decryption errors are recorded and plotted in Figs. 9 and 10.

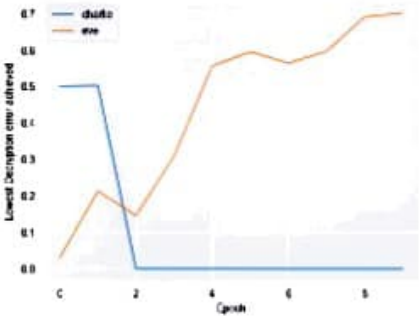

Fig. 9. Decryption error achieved when Alice communicates with Charlie

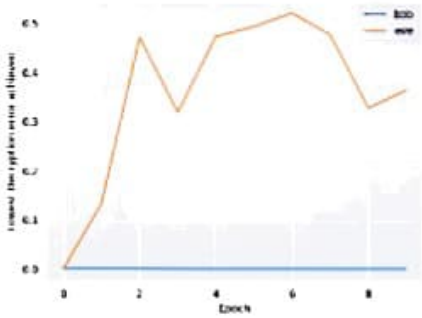

Fig. 10. Decryption error achieved when Alice communicates with Bob

Analysis

We can see that Charlie starts with a decryption error of 50% but quickly synchronizes with Alice (after 2000 iterations) and gets a nearly perfect accuracy.

Eve has at the start of the training a very good accuracy but it starts to go down quickly after about 2000 iteration.

In this training experiment, Bob had a perfect accuracy from the beginning. All Alice had to do while training is to rise the decryption error for Eve which is done after about 4 epochs.

As in the second scenario, Bob cannot talk properly to Charlie without a proper synchronization. As we can see in Fig. 11 Bob's error is around 45%.

3.5 Discussions

Despite having a different way to communicate, during our experiments all the three scenarios had pretty much the same training time and accuracy. The usage of cases depend on scenario. If one entity talks to two other entities at the same time, then the first scenario is appropriate. Second and Third scenario are best for talking separately. Third scenario differs from the second scenario by letting Alice use one single key per neural network which will allow to "hide" the messages sent by Alice to Bob from Charlie and vice-versa. In a multi-party perspective, The best usage cases will be the same, First Scenario for a group talk; Second and third scenario to talk separately.

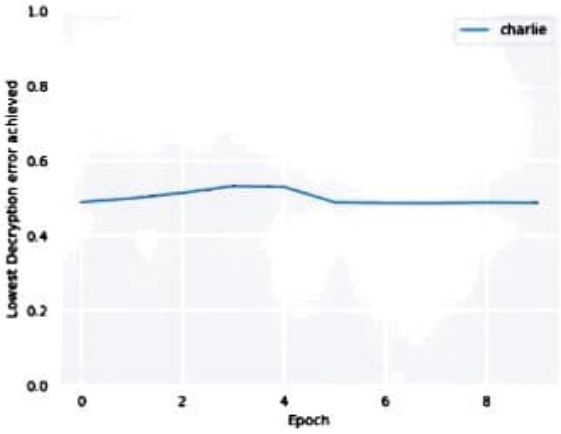

Fig. 11. Decryption error achieved when Bob communicates with Charlie

The extension to a multi-party scheme is theoretically possible, the only inconvenience is the training time – the more parties we have, the more time it takes to train them. One possible way to do it is to train the neural networks in advance and save the parameters in a trusted server or party; then, the trusted party will serve as much parameters as required for the entities willing to communicate.

4 Conclusion and Further Work

We presented our method to make a 3-party adversarial network based on Google's 2-party model. We proposed three different scenarios where we proved in each of them very good accuracy for the receiving parties.

As an interesting future work, one may consider extending three party neural networks towards achieving resistance against more concrete adversaries (e.g. chosen plaintext attack) and extending to multiple-party cases.

Acknowledgement

Ishak Meraouche is supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan for his studies at Kyushu University.

Sabyasachi Dutta was financially supported by the National Institute of Information and Communications Technology (NICT), Japan, under the NICT International Invitation Program during his stay at Kyushu University where the initial phase of the research work was carried out.

References

1. Abadi, M., Andersen, D.G.: Learning to protect communications with adversarial neural cryptography. CoRR **abs/1610.06918** (2016)
2. Anderson, R.J., Biham, E., Knudsen, L.R.: The case for serpent. In: The Third Advanced Encryption Standard Candidate Conference, 13-14 April 2000, New York, New York, USA, pp. 349–354 (2000)
3. Coutinho, M., de Oliveira Albuquerque, R., Borges, F., García-Villalba, L.J., Kim, T.: Learning perfectly secure cryptography to protect communications with adversarial neural cryptography. Sensors **18**(5), 1306 (2018)
4. Dörner, S., Cammerer, S., Hoydis, J., ten Brink, S.: Deep learning-based communication over the air. IEEE J. Sel. Top. Sign. Proces. **12**(1), 132–143 (2017)
5. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, 8-13 December 2014, Montreal, Quebec, Canada, pp. 2672–2680 (2014)
6. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015 (2015)
7. Hayes, J., Danezis, G.: Generating steganographic images via adversarial training. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pp. 1954–1963 (2017)
8. Jamil, T.: The rijndael algorithm. IEEE Potentials **23**(2), 36–38 (2004)
9. Kanter, I., Kinzel, W., Kanter, E.: Secure exchange of information by synchronization of neural networks. EPL (Europhys. Lett.) **57**(1), 141 (2002)
10. Ke, Y., Zhang, M., Liu, J., Su, T.: Generative steganography with kerckhoffs' principle based on generative adversarial networks. CoRR **abs/1711.04916** (2017)
11. Klimov, A., Mityagin, A., Shamir, A.: Analysis of neural cryptography. In: Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Queenstown, New Zealand, 1-5 December 2002, pp. 288–298 (2002)
12. Schneier, B.: Description of a new variable-length key, 64-bit block cipher (blowfish). In: Fast Software Encryption, Cambridge Security Workshop, Proceedings, Cambridge, UK, 9-11 December 1993, pp. 191–204 (1993)

13. Wang, X.Y., Yang, L., Liu, R., Kadir, A.: A chaotic image encryption algorithm based on perceptron model. Nonlinear Dyn. **62**(3), 615–621 (2010)
14. Yedroudj, M., Comby, F., Chaumont, M.: Steganography using a 3 player game. CoRR **abs/1907.06956** (2019)
15. Yu, W., Cao, J.: Cryptography based on delayed chaotic neural networks. Phys. Lett. A **356**(4), 333–338 (2006)
16. Zhou, L., Chen, J., Zhang, Y., Su, C., James, M.A.: Security analysis and new models on the intelligent symmetric key encryption. Comput. Secur. **80**, 14–24 (2019)