Learning multi-party adversarial encryption and its application to secret sharing

Ishak Meraouche Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan (e-mail: ishak.meraouche@gmail.com)
Sabyasachi Dutta Department of Computer Science, University of Calgary, Canada (e-mail: saby.math@gmail.com)
Sraban Kumar Mohanty IIITDM Jabalpur, Jabalpur, India (e-mail: sraban@iiitdmj.ac.in) Isaac Agudo NICS Lab, University of Malaga, Spain (e-mail: isaac@lcc.uma.es) Kouichi Sakurai Department of Information Science and Electrical Engineering, Kyushu University, Japan (e-mail: sakurai@inf.kyushu-u.ac.jp)

***Abstract*—Neural networks based cryptography has seen a significant growth since the introduction of adversarial cryptography which makes use of Generative Adversarial Networks (GANs) to build neural networks that can learn encryption. The encryption has been proven weak at first but many follow up works have shown that the neural networks can be made to learn the One Time Pad (OTP) and produce perfectly secure ciphertexts. To the best of our knowledge, existing works only considered communications between two or three parties. In this paper, we show how multiple neural networks in an adversarial setup can remotely synchronize and establish a perfectly secure communication in the presence of different attackers eavesdropping their communication. As an application, we show how to build Secret Sharing Scheme based on this perfectly secure multi-party communication. The results show that it takes around $45,000$ training steps for 4 neural networks to synchronize and reach equilibria. When reaching equilibria, all the neural networks are able to communicate between each other and the attackers are not able to break the ciphertexts exchanged between them.**

Cryptography, Encryption, Generative Adversarial Networks, Neural Networks, Secret Sharing

I. Introduction

Artificial Intelligence and Cryptography have been separate disciplines in the past due to the difficulty to build models that can learn different cryptography primitives and techniques. However, with the advances in deep learning especially with the introduction of Generative Adversarial Networks (GANs) [1], things are starting to take another turn. GANs are neural networks (usually a generator and a discriminator) that compete against each other in order to learn a specific task or get better at it. Abadi and Andersen [2] have shown in 2016 that GANs can be used to learn symmetric encryption. In their model, two parties (namely Alice and Bob) compete against a third neural network (namely Eve) in order to protect their communication and prevent Eve from reading the messages exchanged between them. Alice and Bob had a common secret key and were able to use it to encrypt and decrypt messages. However, Zhou et al. [3] showed that the ciphertexts generated by the neural networks in this model are not secure as they failed multiple standard statistical tests such as the NIST Statistical test. Multiple follow up researches [4, 5] have shown that, with a modification to the neural network structure and the training process, the neural networks can learn perfectly secure encryption. More precisely, training results of the models proposed by Li et al. [4], Coutinho et al. [5] show that the ciphertexts are obtained as the exclusive or (XOR) of the plaintext and secret key i.e., an encryption of a bit of the plaintext is the result of the XOR operation with a unique bit of the key. The results in the work given by Li et al. [4] learns protection against more adversaries and has more chances of learning the OTP after a training session compared to the results of the work given by Coutinho et al. [5].

Our Contribution. In this paper, we use the model proposed by Li et al. [4] to build a perfectly secure multi-party adversarial encryption scheme. All the existing works except the contribution by Meraouche et al. [6] studied the problem in a two party setting. Our work is a continuation of the work given by Meraouche et al. [6] which establishes a communication between 3 parties and gives insights on how a multi-party communication can be established. As the model given by Meraouche et al. [6] has been implemented with the same neural network structure proposed in the model Abadi and Andersen [2], it is unable to provide strong security guarantees as shown by Zhou et al. [3]. By using the neural network structure and training model proposed by Li et al. [4], we can bypass the problem and claim that neural networks learn a secure scheme. We choose to use the model proposed by Li et al. [4] instead of the one proposed by Coutinho et al. [5] because the neural networks have more chances of learning the OTP at the end of training in the model given by Li et al. [4]. Additionally, the neural networks learn to secure the ciphertexts against stronger attack models which are detailed in Section III. Lastly, as an application of our new multi-party adversarial encryption model we show how to build an (information theoretic) implementation of secret sharing scheme for any general access structure.

This paper is structured as follows: In section II, we define

Ishak Meraouche is financially supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT) for his studies at Kyushu University.
Sabyasachi Dutta is financially supported by Mitacs, Canada.
This research is supported by the Telecommunications Advancement Foundation of Japan and also operating as a part of the India-Japan Cooperative Science Programme (IJSCP) by DST and JSPS.

the related works necessary to understand the paper. Namely: Adversarial Encryption [2], 3-Party Adversarial Encryption [6], Perfectly secure adversarial encryption [4, 5] and Secret Sharing. Then, we define our multi party adversarial encryption model in Section III and our Secret Sharing model in section IV. Lastly, we show and discuss results such as training time and decryption accuracy in section V. We conclude the paper in section VI.

II. Background

In this section we describe the relevant details on the background that are required for this paper: Adversarial encryption and key exchange. However, neural networks have also been applied to Steganography [7, 8, 9, 10] where Alice learns to hide a message in an image that Bob can extract and Eve cannot differentiate an image that has a hidden message and one that doesn't. A key exchange scheme has also been proposed by Kanter et al. [11], however, it has been proven to be weak against multiple attacks by Klimov et al. [12] and despite some improvements such as the ones given by Salguero et al. [13], this scheme is still considered as weak and vulnerable against many attacks.

A. Adversarial Encryption

Adversarial Encryption has been first introduced by Abadi and Andersen [2]. The authors have shown that two neural networks (Alice and Bob) with the same structure are able to remotely synchronize and learn to encrypt and decrypt messages in the presence of an eavesdropper Eve and prevent Eve from decrypting the messages. All three neural networks share the same neural network structure but Alice and Bob have a common secret key that Eve does not have access to. During training, Alice trains to generate ciphertexts that can be easily decrypted by Bob and cannot be decrypted by Eve without the key. Bob learns to reconstruct the plaintext from the ciphertext using the secret key and get a decryption accuracy as close to 100% as possible. Eve tries to decrypt the ciphertexts transiting between Alice and Bob without the secret key and get as close as possible to 100% accuracy. Alice is therefore competing against Eve and trying to generate ciphertexts that are difficult to decrypt without the secret key and Eve is competing against Alice by trying to decrypt these ciphertexts. If Eve is successful in its decryption, its high accuracy will push Alice to generate ciphertexts that are more complicated in order to prevent Eve from decrypting the ciphertexts. This "game" goes on until the three neural networks have reached equilibria and either Alice and Bob or Eve have won.

1) Neural Networks Structure used: The neural network structure used by Alice, Bob and Eve is described in Table I below. The neural networks need to have the same structure in order to be able to synchronize their weights and obtain the same weights matrix after training. If they used a different structure, they would have a weights matrix that is different in size and shape and therefore would not be able to synchronize. Synchronizing neural networks with different structures is a challenging issue that will be considered in a future contribution.

TABLE I
Summary of the neural network structure proposed by Abadi and Andersen [2] for Alice, Bob and Eve

Layer	FC Layer Type	Activation	Filters	Kernel Size	Strides	Padding
1	FC Layer (Dense)	Relu	-	-	-	-
2	Conv1D	Sigmoid	2	4	1	same
3	Conv1D	Sigmoid	4	2	2	Valid for Alice and Bob. Same for Eve.
4	Conv1D	Sigmoid	4	1	1	same
5	Conv1D	Tanh	1	1	1	same

We can see that the neural network starts with a relu-activated fully connected layer to read and mix the input (Which is the plaintext and key for Alice, ciphertext and key for Bob and the ciphertext only for Eve). The plaintext, ciphertext and secret key all have the same size N. Alice and Bob receive an input of size $2N$ (plaintext and key) and Eve receives an input of size N (the ciphertext). The output of the fully connected layer is then followed by three sigmoid-activated convolutions with different number of filters, kernel sizes and strides. The second convolution has valid padding for Alice and Bob in order to reduce the data size from $2N$ to N. However as Eve already receives an N size input, there is no need for halving the data and therefore same padding is used. The last convolution is activated with the tanh function in order to produce an output in the interval $[-1, 1]$.

This structure is based on the mix and transform principle as explained by [2]. The fully connected layer at the beginning of the structure will read and mix the bits of the input together before forwarding to the next layers. The next layers, which are convolutions will transform their input.

The decryption accuracy of Bob and Eve is assessed through the following loss functions which are described by the following annotations:

- W_A denotes the weights of the neural network A.
- $A(W_{Alice}, P, K)$ denotes Alice's output on input P, K.
- $B(W_{Bob}, C, K)$ denotes Bob's output on input C, K.
- $E(W_{eve}, C)$ denotes Eve's output on input C.
- $d(x, y)$ denotes the L_1 distance between x and y.

The loss function for Bob is defined below in Equation 1.

$$L_{Bob}(W_{Bob}, P, K) = d(P, B(W_B, C, K)) \tag{1}$$

Intuitively, the loss function L_B determines how wrong Bob is in his decryption.

Similarly, we define the loss function for Eve in Equation 2 below.

$$L_{Eve}(W_{Eve}, P) = d(P, E(W_E, C)) \tag{2}$$

Intuitively, L_{Eve} determines how wrong Eve is when decrypting the ciphertext.

Alice's loss is related to Eve's loss and Bob's so that Alice is penalized when Bob's loss is too low or Eve's accuracy is too high. Alice's loss function is defined in Equation 3

$$L_{Alice} = L_{Bob} + (1 - L_{Eve}^2) \tag{3}$$

B. Perfectly Secure Adversarial Encryption

The model described before has been shown to produce ciphertexts that contain information about the plaintext and/or the secret key [3]. Therefore, Li et al. [4] modified the neural network structure and the training process with the aim to produce ciphertexts that are secure and do not leak information about the plaintext and/or the key. The key modifications shown by Li et al. [4] to improve the security are described below.

In addition to Eve, a neural network modeling the threat that an attacker could decrypt the ciphertexts without the secret key, two more neural networks have been introduced. The first one corresponds to an attacker that receives the ciphertext and the secret key and therefore can easily decrypt the ciphertexts generated by Alice. Alice's neural network in return will be forced to generate more complicated ciphertexts that do not entirely rely on the secret key but also the neural network structure of Alice and Bob. The authors conclude that adding an aggressive attacker that has access to leaked secret keys pushes Alice to learn a mapping that does not entirely rely on the secret key but also on the parameters of their neural networks and therefore produce stronger ciphertexts. We could then assume that the neural network parameters contribute to the mapping from plaintext to ciphertext and vice versa. The other additional threats refers to an attacker that tries to tell fake and real ciphertexts apart. This neural network receives a plaintext, its corresponding ciphertext and a randomly generated ciphertext and tries to tell which ciphertext corresponds to the plaintext. This pushes Alice to generate ciphertexts that are indistinguishable from randomly generated ones and therefore makes sure that no information can be extracted from these ciphertexts that are related to the plaintext and/or the secret key.

Apart from more aggressive attackers which seem to be pushing Alice to generate better ciphetexts as shown by Zhou et al. [3], Li et al. [4], Li et al. [4] also modified the structure of each neural network. The new neural network structure they used is shown in Table II

TABLE II
NEURAL NETWORK STRUCTURE PROPOSED BY LI ET AL. [4]. THE RESBLOCKS [14] IN THEIR MODEL CONTAIN TWO IDENTICAL CONVOLUTIONAL LAYERS.

Layer#	Layer Type	Activation	Filters	Kernel Size	Strides
1	FC Layer (Dense)	ReLU	-	-	-
2	Resblock*	Sigmoid	2	2	1
3	Conv1D	Sigmoid	4	4	2
4	Resblock*	Sigmoid	4	4	1
5	Conv1D	Tanh	1	1	1

The ciphertexts generated by the neural networks are information theoretic secure as they are shown by Li et al. [4] to be the result of the XOR operation between the plaintext and the secret key. Equation 4 taken from [4] shows a sample XOR operation between the plaintext P and the secret key K performed by the neural network (NN). Both the plaintext and ciphertext have a size of 42 bits. In their example, we can see that the first bit p_1 from the plaintext has been XOR-ed with the second bit of the secret key k_2, the second bit of the plaintext with the seventh bit of the key, etc.

$$NN\left(P,K\right) = \begin{bmatrix} p_1 \oplus k_2 \\ p_2 \oplus k_7 \\ p_3 \oplus k_9 \\ p_4 \oplus k_{14} \\ p_5 \oplus k_{25} \\ p_6 \oplus k_{21} \\ p_7 \oplus k_{19} \\ \dots \\ p_{41} \oplus k_{39} \\ p_{42} \oplus k_{11} \end{bmatrix} \tag{4}$$

Coutinho et al. [5] took a different approach by removing all the attackers and keeping only one Attacker that receives two plaintexts and one ciphertexts in order to differentiate between the two plaintexts and tell which one has been encrypted to the given ciphertexts. While their method has been shown to be effective at learning the OTP, the results of the model given by Li et al. [4] are better and therefore we will use this model to implement the multi-party adversarial encryption model in Section III.

C. 3-Party Adversarial Encryption

In the 3-party adversarial encryption scheme given by Meraouche et al. [6], a setup similar to the one given by Abadi and Andersen [2] is used in order to build a 3-party encrypted communication with neural networks. The model setup is organized as follows: in addition to Alice and Bob, a third neural network wants to join their communication and discuss with them. Naturally, Charlie needs to have access to the secret key and also have the same neural network structure as Alice and Bob.

There are different communication scenarios, as shown in Figure 1, depending on the role of the third party, Charlie. In the first scenario, Charlie joins the existing communication channel and there is no secrecy between the 3 parties: any party can communicate with another party or intercept and read the messages sent to any another party. In the second scenario, Charlie joins the communication by creating a new channel with Alice, that way communication between Alice and Bob remain privy to Charlie but Charlie and Bob cannot communicate directly and need to use Alice as a bridge if they need to communicate. In the third scenario, Charlie creates a channel with Bob, and alice will have to use Charlie as a bridge to communicate with Bob. The parameters are unique per link and cannot be shared with other parties.

Assuming the second scenario, the training setup and procedure is the same as the one by Abadi and Andersen [2] with the difference that Alice now also needs to take into consideration Charlie's output when generating ciphertexts. This has been

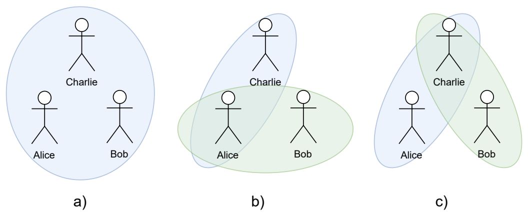

Fig. 1. Different communication scenarios

done by adding Charlie's loss to Alice's loss function so that it becomes as defined in Equation 5.

$$L_{Alice} = L_{Bob} + L_{Charlie} + (1 - L_{Eve}^2) \quad (5)$$

Charlie's Loss is the same as Bob's loss and is defined in Equation 6.

$$L_{Charlie}(W_{Charlie}, P, K) = d(P, Charlie(W_{Charlie}, C, K)) \quad (6)$$

$Charlie(W_{Charlie}, C, K)$ is Charlie's output on the ciphertext C and the secret key K and $W_{Charlie}$ are Charlie's neural network parameters.

The reason Meraouche et al. [6] considered experimenting on training with more than two parties was that because the 2-party designs [2] did not allow more than two parties to communicate. Abadi and Andersen [2] did not define any explicit way how more than two parties can synchronize and communicate together. Meraouche et al. [6] took initial steps in that direction by showing how more than two parties can train together and learn the same encryption/decryption pattern. In this paper, we follow these steps by showing how a communication can be established with any number of parties, not limited to 2 parties as shown by Abadi and Andersen [2] or 3 parties as shown by Meraouche et al. [6].

D. Secret Sharing

In a secret sharing system, a secret is distributed among a user set U such that authorized subsets of users can reconstruct the secret, and unauthorized set will not learn anything. Let Γ be a subset of the power set, 2^U , that specifies the subsets of users that form an authorized set; *i.e.*, the set of their shares can recover the secret. A subset $F \subset U$ which is not in Γ, *i.e.* $F \notin \Gamma$, is called an unauthorized set and the set of shares $(S_u)_{u \in F}$ will be independent of secret S. The collection of unauthorized sets is denoted by $\mathcal{F}$. Note that, in our model $\Gamma \cap \mathcal{F} = \emptyset$ and $\Gamma \cup \mathcal{F} = 2^U$. A formal definition of secret sharing [15] is as follows.

Definition 1 (Secret Sharing Scheme)**.** Let U be a set of n users labeled by $[n] = \{1, 2, \ldots, n\}$. Let $(\Gamma, \mathcal{F})$ denote an access structure on these n users with $\mathcal{F} = 2^U \backslash \Gamma$. A secret sharing scheme Π for an access structure $(\Gamma, \mathcal{F})$ consists of a pair of algorithms $(Share, Rec)$. $Share$ is a randomized algorithm that gets as input a secret S (from a domain of secrets $\mathcal{S}$ with at least two elements), Γ and the number of parties n, and generates n shares $(S_1, \ldots, S_n) \longleftarrow Share(S)$. Rec is a deterministic algorithm that gets as input the shares of a subset B of parties and outputs a string. The requirements for defining a secret sharing scheme are as follows:

- *Correctness:* If $\{S_u\}_u \leftarrow Share(S)$ for some secret $S \in \mathcal{S}$, then for any $B \in \Gamma$, we always have $Pr[Rec(\{S_u\}_{u \in B}) = S] = 1$.
- *Secrecy:* Let $\{S_u\}_u = Share(S)$. For $F \in \mathcal{F}$, let $S_F = \{S_u\}_{u \in F}$. Then, for any $s_0, s_1 \in \mathcal{S}$ and for any distinguisher D with output in $\{0, 1\}$, it must hold that $$|Pr[D(Share(s_0)_F) = 1] - Pr[D(Share(s_1)_F) = 1]| \leq \epsilon.$$

First information theoretic secret sharing for threshold access structures was proposed by Shamir [16] and Blakley [17]. Later, threshold secret sharing was extended to the case of general access structure by Ito et al. [18] and also to different types of important access structures like hierarchical [19, 20, 21], compartmented [19] etc.. It is well known that for information theoretic secret sharing the share size is at least the secret size. Krawczyk [22] proposed a computationally secure secret sharing to reduce the share size. Secret sharing for image data was introduced by Naor and Shamir [23].

Zheng et al. [24] modeled the secret sharing problem as a classification problem and built GANs based secret sharing scheme. Their model contains a Generator and a Discriminator that compete against each other. The Generator takes as input a secret S and outputs m shares. The discriminator is fed m real shares and m fake random shares and has to tell which ones are real and which ones are not. The training continues until the generator is producing shares that are indistinguishable from random ones and the discriminator is not able to differentiate between them.

A very recent work by Wang et al. [25] addresses the construction of progressive secret sharing. Their technique assigns multiple weights to model parameters for progressive recovery. Actually, they encode their model parameters using polynomial based threshold secret sharing such that a hierarchy is achieved among the set of shareholders. The sum of the weights needs to be higher than a threshold value to recover the secret.

It is worth pointing out that our secret sharing does not require anything else than training of NNs. In other words, our construction is not hybrid - it is purely dependent on synchronization of neural networks and does not use any external primitive(s) like Shamir secret sharing to achieve the goal.

The focus of this paper is on secret sharing. Based on the perfectly secure adversarial encryption model proposed by Li et al. [4], we extend the 3-party adversarial cryptography model [6] to an information theoretic secure multi-party encryption model and use it to obtain a secret sharing scheme.

III. Multi-Party Perfectly Secure Adversarial Encryption Model

With the outstanding progress that neural networks based encryption has seen especially with the possibility of learning the one time pad as shown by Li et al. [4], Coutinho et al. [5], a concrete definition of how to communicate and train among multiple parties is deemed necessary.

In a multi-party communication secured through GANs and adversarial training scenarios [2, 4, 5, 6], we have multiple neural networks all aiming to communicate together in a secure way that prevents attackers from decrypting the ciphertexts exchanged between them.

In the first scenario of the 3-party adversarial encryption model [6], Alice wants to communicate securely with Bob but also Charlie. As Alice, Bob and Charlie all share the same neural network structure, they were trained all together in the presence of Eve just like the model shown by Abadi and Andersen [2]. However, this training method with only one attacker has proven to be insecure [3] and training against more aggressive attackers has been shown by multiple authors [4, 5] to push Alice to generate ciphertexts that are perfectly secure and do not contain any information about the plaintext or the key.

Therefore, in our multi-party communication, we do not use the model and training process proposed by Abadi and Andersen [2] but the one proposed by Li et al. [4] as it provides neural networks that are able to generate perfectly secure ciphertexts.

Therefore, Alice trains with a total of N neural networks (or parties) that have the same structure as Alice. Each neural network's loss function L_{NN} is defined in Equation 7.

$$L_{NN}(W_{NN}, P, K) = d(P, NN(W_{NN}, C, K)) \tag{7}$$

Where P is the plaintext, K is the secret key, C is the ciphertext, W_{NN} are the neural network parameters of the neural network NN, d is the L_1 distance and $NN(W_{NN}, C, K)$ is the neural network's output on input C and K using the parameters W_{NN}.

Alice has to take into consideration every neural network in the setup and therefore Alice's loss function is going to be the sum of the losses of all the neural networks in the setup. Alice's loss function is shown in Equation 8.

$$L_{Alice} = \sum_{i=1}^{N} L_{NN_i} \tag{8}$$

Equation 8 contains the sum of the losses of every neural network in the setup and allows Alice to generate ciphertexts that can be decrypted by them at the end of training. However, without any attacker to compete against, the plaintext-ciphertext mapping is going to be weak as shown by Coutinho et al. [5] despite the communication being successfully established.

Adding only one attacker (Eve) has also been shown to produce ciphertexts that are weak against many attacks as shown by Zhou et al. [3]. For instance, Zhou et al. [3] show that the ciphertexts produced when using only one Attacker (Eve) did not pass many statistical tests such as the X^2 test and the NIST statistical test.

Therefore, we take the same approach done by Li et al. [4], Coutinho et al. [5] and add more attackers in the setup. These attackers are going to push Alice to generate perfectly secure ciphertexts.

Alice now has to train against a total of four attackers inspired from the models given by Li et al. [4] and Coutinho et al. [5] to learn protection against different types of attacks. The four attackers and the types of attacks that they are going to do are described as follows:

- Attacker 1: Has access to the ciphertext only and tries to decrypt without the key as proposed by Abadi and Andersen [2]. This is the most basic attacker which pushes Alice to generate ciphertexts that rely on the secret key and prevent Alice from learning a plaintext-to-ciphertext mapping that is too simple.
- Attacker 2: Has access to the ciphertext and the secret key and learns to decrypt with the key. This attacker pushes Alice to generate ciphertexts that rely not only the secret key but also the neural network parameters as shown by Li et al. [4].
- Attacker 3: Receives a plaintext P, its corresponding ciphertext C and a random ciphertext C' and has to determine which ciphertext belongs to P. This attacker outputs two probabilities: π_1, the probability that C is a ciphertext for P and π_2 the probability that C' is a ciphertext for P. This attacker pushes Alice to generate ciphertexts that are indistinguishable from randomly generated ones as shown by Li et al. [4].
- Attacker 4: This attacker receives two plaintexts P_1, P_2 and a ciphertext C and has to tell which plaintext has been encrypted to C as proposed by Coutinho et al. [5]. This attacker outputs two probabilities: π_1, the probability that P_1 is the plaintext that corresponds to C and π_2, the probability that P_2 corresponds to C. This Attacker pushes Alice to generate ciphertexts that are secure against chosen plaintext attacks as shown by Coutinho et al. [5].

As these attackers are neural networks, they are trained to perform different attacks. So even if one Attacker might look stronger than another one, it can only perform the attack it has been trained for. As a result, we use different attackers so that Alice can learn to produce ciphertexts that are resistant to multiple Attacks.

Figure 2 below shows the overall multi party model including the four attackers.

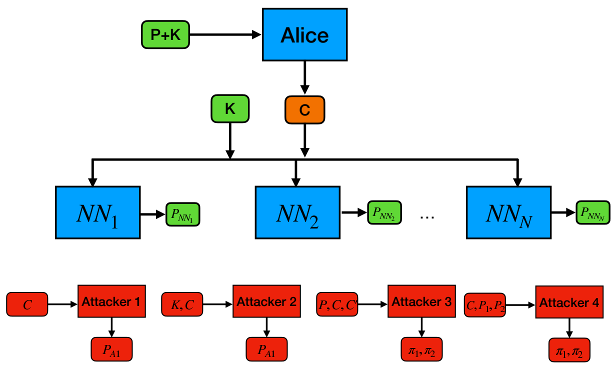

Fig. 2. Overall architecture of the multi-party adversarial encryption model.

We can see in Figure 2 that Alice receives as input a plaintext P and the secret key K and outputs C, the ciphertext.

TABLE III
NEURAL NETWORK STRUCTURE USED FOR $Attacker_3$ AND $Attacker_4$ IN ORDER TO PRODUCE PROBABILITIES.

Layer#	Layer Type	Activation	Filters	Kernel Size	Strides
1	FC Layer (Dense)	ReLU	-	-	-
2	Resblock	Sigmoid	2	2	1
3	Conv1D	Sigmoid	4	4	2
4	Resblock	Sigmoid	4	4	1
5	Conv1D	ReLU	1	1	1
6	FC Layer (Dense)	Softmax	-	-	-

All the neural networks that are communicating with Alice ($NN_1 \cdots NN_N$) receive as input the ciphertext C and the key K and produce their decrypted text $P_{NN_1} \cdots P_{NN_N}$. Attacker 1 receives the ciphertext and tries to decrypt it and output P_{A1}. Attacker 2 receives C, K and tries to decrypt with the key outputting P_{A2}. Attacker 3 receives a plaintext P, its corresponding ciphertext C and a random ciphertext C' and output two probabilities: π_1 the probability that C is a ciphertext for P and π_2 the probability that C' is a ciphertext for P. Lastly, Attacker 4 receives a ciphertext C its corresponding plaintext P_1 and a random plaintext P_2 and outputs two probabilities: π_1 the probability that P_1 is the plaintext that was encrypted to C and π_2 the probability that P_2 is the plaintext that was encrypted to C.

$Attacker_1$'s loss function is defined in Equation 9 and similarly to the other neural networks, it is the L_1 distance between the plaintext and $Attacker_1$'s output.

$$L_{A1}(W_{A1}, P) = d(P, A1(W_{A1}, C)) \tag{9}$$

$Attacker_2$'s loss function is defined in Equation 10 and similarly to the other neural networks, it is the L_1 distance between the plaintext and $Attacker_2$'s output. The difference from $Attacker_1$ is that $Attacker_2$ has also access to the secret key that we suppose that it was leaked to him.

$$L_{A2}(W_{A2}, P) = d(P, A2(W_{A2}, C, K)) \tag{10}$$

As for $Attacker_3$ and $Attacker_4$, these two neural networks are making classifications and therefore needs to have some changes in their neural network structure in order to output probabilities. Basically, we keep the same neural network structure as Alice but add an additional softmax-activated fully connected layer as a last layer in order to output probabilities instead of a bistream. The new neural network structure for $Attacker_3$ and $Attacker_4$ is shown in Table III.

We notice that the only change is an additional softmax-activated fully connected layer at the end of the neural network structure.

As for the loss function for $Attacker_3$, it is the binary cross-entropy. Given N plaintexts $\left[P_{(0)}, P_{(1)}, ...P_{(N-1)}\right]$, and two sets of N ciphertexts $\left[C^1_{(0)}, C^1_{(1)}, \cdots C^1_{(N-1)}\right]$, $\left[C^2_{(0)}, C^2_{(1)}, \cdots C^2_{(N-1)}\right]$ we define the loss function L_{A3} for $Attacker_2$ in Equation 11 below.

$$L_{A3} = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=1}^{2} y^j_{(i)} \log\left(\pi^j_{(i)}\right) \tag{11}$$

Where $y^j_{(i)} = 1$ if $P_{(i)}$ is the plaintext of $C^j_{(i)}$ and 0 otherwise. Intuitively, $\pi^j_{(i)}$ is the probability that $C^j_{(i)}$ is the ciphertext corresponding to the plaintext $P_{(i)}$. Therefore, $Attacker_3$ learns by minimizing L_{A3}.

The loss function of $Attacker_4$ is similar to the one of $Attacker_3$ i.e. the binary cross-entropy.

Given N ciphertexts $\left[C_{(0)}, C_{(1)}, \ldots, C_{(N-1)}\right]$, and two sets of N plaintexts $\left[P^1_{(0)}, P^1_{(1)}, \cdots P^1_{(N-1)}\right]$, $\left[P^2_{(0)}, P^2_{(1)}, \cdots P^2_{(N-1)}\right]$ we define the loss function L_{A4} for $Attacker_4$ in Equation 12 below.

$$L_{A4} = -\frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=1}^{2} y^j_{(i)} \log\left(\pi^j_{(i)}\right) \tag{12}$$

Where $y^j_{(i)} = 1$ if $C_{(i)}$ is the ciphertext of $P^j_{(i)}$ and 0 otherwise. Intuitively, $\pi^j_{(i)}$ is the probability that $P^j_{(i)}$ is the plaintext corresponding to the ciphertext $C_{(i)}$.

Therefore, $Attacker_4$ learns by minimizing L_{A4}.

Alice's loss function defined in 8 needs to be modified in order to take into consideration the four attackers that we added to the setup. Alice's new loss function is defined in Equation 13.

$$\begin{aligned} L_{Alice} = \sum_{i=1}^{N} L_{NN_i} + (1 - L^2_{A_1}) + (1 - L^2_{A_2}) \\ -min(L_{A3}, 0.5) - min(L_{A4}, 0.5) \end{aligned} \tag{13}$$

We use $min(L_{A3}, 0.5)$ and $min(L_{A4}, 0.5)$ in Alice's loss function in order to prevent Alice from maximising the loss of $Attacker_3$ and $Attacker_4$. Ideally, we want their loss to be equal to 0.5 which, in probabilities, corresponds to making assumptions that are random.

The results shown in Section V show that indeed all the neural networks are able to communicate with Alice while preventing the attackers from reaching their goals.

A. Communication scenarios

We consider the following two scenarios for our multi-party communication setup:

1) First Scenario: The first scenario is identical to the first scenario that was proposed by Meraouche et al. [6]: All the communicating parties are synchronized with Alice and therefore encrypted messages sent to/from any party can be decrypted and read by the others.

2) Second Scenario: In the second scenario, the encrypted messages communicated between any subset of parties and Alice remains hidden from the rest of the parties. In such a case, this subset of parties needs to train again with Alice with another set of parameters in order to be able to exchange messages with Alice while keeping privacy of the messages intact from the other parties in the set.

IV. Application: Secret Sharing based on Multi-Party Adversarial Encryption

Our proposed secret sharing scheme is based on the multi party adversarial encryption model proposed in Section III and uses the second scenario where it is possible to achieve secrecy of the messages exchanged between one or more parties with Alice.

In our secret sharing scheme, we have a Dealer D that has a master secret MS to be divided into N shares $st_1, \cdots, st_N$ and distributed among N shareholders $SH_1, \cdots, SH_N$ such that all the N shareholders are required to reconstruct the master secret MS. That is, we first propose an N-out-of-N secret sharing scheme. Using this construction, we later generalize to propose secret sharing schemes for any general access structure.

The Dealer and the shareholders are all neural networks with the same structure shown in Table I. The Dealer plays the same role of Alice in the proposed multi-party adversarial encryption model and synchronizes with the shareholders as described in the second scenario (*see* Section III). For N shareholders to synchronize with, the Dealer has N sets of parameters $W = \{W_1, \ldots, W_N\}$. Dealer uses one unique set of parameters W_i to synchronize with a unique shareholder SH_i. The Dealer can also be viewed as a server containing N neural networks each synchronizing with one unique shareholder.

Once the synchronization is complete, the Dealer generates N secret keys $K_1, \ldots, K_N$ that are going to be used to encrypt/decrypt the data with the N shareholders. We assume that the Dealer has a secure tunnel with every shareholder in order to deliver the secret key to them. The overall setup has the following variables:

- The master secret MS.
- $W_1, \ldots, W_N$ denote the parameters which the Dealer has used to synchronize with the N shareholders $SH_1, \ldots, SH_N$. Every W_i was used to synchronize with the shareholder SH_i ($i \in [1, N]$).
- We denote the parameters (of the neural network) of the i^{th} shareholder by W_{SH_i}. W_{SH_i} are the result of the synchronization process of the i^{th} shareholder with the Dealer. We note that this W_{SH_i} is equal to W_i after the training.
- W_{SH_i} will be stored by the i^{th} shareholder and W_i will be stored by the dealer.
- $K_1 \cdots K_N$, the secret keys that the Dealer has distributed to the N shareholders
  $SH_1, \ldots, SH_N$. Every K_i is sent to SH_i with $i \in [1, N]$.

Additionally, we define the following functions that we use in the process of creating the shares and the reconstruction of the master secret:

- The function $Enc(W_i, M, K_i)$ denotes the encryption by the Dealer with plaintext input M, key K_i and using the parameters W_i.
- The encryption process consists of passing the message M, the key K_i through the Dealer's neural network and calculating the output of its neural network using the parameters W_i. The output is the encrypted result.
- $Dec(W_{SH_i}, C, K_i)$ denotes the decryption of the input C by the i^{th} shareholder using the key K_i and the parameters W_{SH_i}.
- The decryption process consists of passing the encrypted message C, the key K_i through the i^{th} shareholder's neural network and calculating the output of its neural network using the parameters W_{SH_i}. The output is the decrypted result.

Shares Construction.

In the Setup we have a Dealer (neural network) with parameters $W_1, \ldots, W_N$ in synchronization (as described in the second scenario of Section III) with N parties $SH_1, \ldots, SH_N$. The Dealer with input master secret MS constructs N shares $st_1, \ldots, st_N$ in the following manner.

1) The Dealer generates N random keys $K_1, \ldots, K_N$. Each key K_i is shared with the shareholder SH_i using a secure tunnel.
2) In the first step, the Dealer encrypts MS using W_1, K_1 and computes $S_1 = Enc(W_1, MS, K_1)$. Dealer now proceeds as follows:
   Computes $S_i = Enc(W_i, S_{i-1}, K_i)$ for all $i = 2, \ldots, N$.
3) The dealer sends S_N to SH_N through a secure tunnel and deletes all the information from its own storage.
4) The resulting shares are $st_i = (W_{SH_i}, K_i)$ for $1 \leq i \leq N-1$ and $st_N = (W_{SH_N}, K_N, S_N)$.

Master Secret reconstruction.

The reconstruction procedure is as follows. When all the N shareholders agree to recover the master secret they take the following steps.

The shares construction process is illustrated in Figure 3

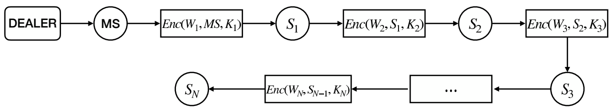

Fig. 3. The shares construction process

Recall, only the last encryption S_N has been distributed to the shareholder SH_N using the secure tunnel that we assume the Dealer has with all shareholders. The other shareholders have only kept their corresponding neural network parameters and secret keys. The reconstruction of the master secret is done by decrypting S_N in the reverse order:

1) SH_N calculates S_{N-1} by decrypting S_N with his parameters W_{SH_N} and key K_N i.e., $S_{N-1} = Dec(W_{SH_N}, S_N, K_N)$. Then, SH_N forwards S_{N-1} to SH_{N-1}.
2) The process is repeated $N-1$ times where in each step, shareholder SH_i calculates S_{i-1} and forwards it to SH_{i-1} until the first shareholder SH_1 receives S_1 and decrypts it to the master secret MS.

The master secret reconstruction process is illustrated in Figure 4

The *correctness* of the recovery of master secret follows immediately from the correctness of synchronization process (i.e., $W_{SH_i} = W_i$ for all i) and the correctness of the decryption algorithms $Dec(W_{SH_i}, S_i, K_i)$ for all i. The *security* property of the above (N, N) scheme follows from

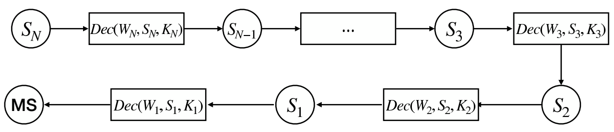

Fig. 4. The master secret reconstruction process

the security of the encryptions $Enc(W_{SH_1}, MS, K_1)$ and $Enc(W_{SH_i}, S_{i-1}, K_i)$ for all $i = 2, \ldots, N$. We emphasize that the encryption algorithms are in fact one time pads as described in Section III. Therefore, the N-out-of-N secret sharing scheme we achieve is perfectly secure.

A. *Secret Sharing schemes for General Access structures*

We have presented a construction to realize N-out-of-N secret sharing scheme for any value of N. Using this basic construction, we can achieve secret sharing schemes for any general access structure (GAS). Suppose $\Gamma = \{B_1, \ldots, B_r\}$ is a general access structure on a set U of users. The dealer runs a $|B_i|$-out-of-$|B_i|$ secret sharing as described above for every set B_i, $1 \leq i \leq r$. The master secret MS remains the same but the parameters W's and the keys K's are independently chosen for each B_i's. The correctness and secrecy of this construction is evident from the respective properties of the underlying $|B_i|$-out-of-$|B_i|$ schemes.

V. Results and Discussion

As a proof of concept, we implement our proposed multi-party perfectly secure encryption model described in Section III.

In the implementation, Alice wants to communicate with three neural networks NN_1, NN_2 and NN_3 in the presence of the four attackers shown in Figure 2. We implement the model using Tensorflow and Keras.

The following are the hyperparameters used to train our neural network.

- **Datasize**: 64 bits for the plaintexts, keys and ciphertexts.
- **Batch Size**: 256.
- **Number of epochs**: Up to 200 but the training might stop earlier if the receiver has reached 100% accuracy and the prediction accuracy of attackers is close to random guesses.
- **Training steps per epoch**: 300.
- **Learning Rate**: 0.0008.
- **Optimizer**: Adam's optimizer.

Alice and the three neural networks NN_1, NN_2 and NN_3 as well as $Attacker_1, Attacker_2$ have the neural network structure shown in Table I. $Attacker_3$ and $Attacker_4$ have the neural network structure shown in Table III.

We train the neural networks until NN_1, NN_2 and NN_3 are able to decrypt the ciphertexts sent by Alice and $Attacker_1, Attacker_2$ have a decryption accuracy of around 50% which is equivalent to a random decryption where the attacker does not know which bit is correct and which one is not. If we trained them to reach 100% accuracy, they would be able to become 100% right just by flipping all their bits. Therefore having 50% accuracy when decrypting is the worst case scenario from the point of the of an attacker making random guesses.

Figure 5 shows the decryption accuracy of the three neural networks NN_1, NN_2 and NN_3 as well as the decryption accuracy of the two attackers $Attacker_1, Attacker_2$ after 155 epochs.

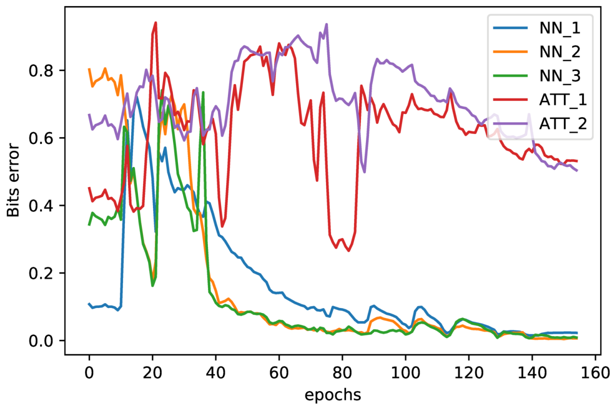

Fig. 5. Decryption accuracy of the three neural networks NN_1, NN_2 and NN_3 and the two attackers $Attacker_1, Attacker_2$ during training. 0% Bits error means that the neural network produced a plaintext with 100% of the bits correct and 1.0 Bits error means that the neural network produced a plaintext with 0% of the bits correct.

We can see that the neural networks start with random decryption accuracy at the beginning of training. After 50 epochs, the three neural networks communicating with Alice start getting better at their decryption with around 20% error in their decryption. The two other attackers have a decryption error ranging between 40 and 60%. It is only after 140 epochs that the neural networks finally reach a stable state where the parties communicating with Alice have perfect accuracy while the two attackers $Attacker_1, Attacker_2$ have approximately 50% accuracy which is the training goal for them so that their output is close to random and they cannot tell which bit is wrong and which one isn't.

As for $Attacker_3$ and $Attacker_4$, they were not able to produce correct probabilities from the beginning to the end of training. Figure 6 shows the probabilities produced by $Attacker_3$ on real and fake ciphertexts and Figure 7 shows the probabilities produced by $Attacker_4$ on real and fake plaintexts. Both of the neural networks are producing 50% probability on real and fake inputs meaning that they are not able to tell which one is real and which one is not.

The experimental results show that the ciphertexts generated by Alice are secure against all these attackers as cannot be decrypted without the key and the weights of the neural networks that trained with Alice. Additionally, Figures 6 and 7 show that the ciphertexts cannot be differentiated from randomly generated ones and contain no information about the plaintexts as $Attacker_4$ has not been able to link the real plaintext to the given ciphertext.

Therefore, we are achieving the same results as the results of the work proposed by Li et al. [4] while allowing more than one party to communicate with Alice. This means that the encryption done by Alice or the Dealer when performing

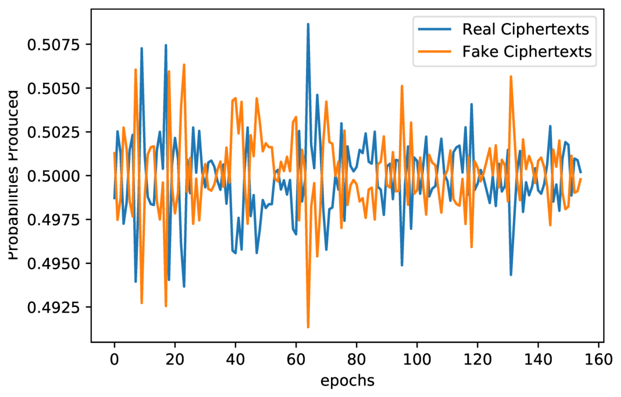

Fig. 6. Probabilities produced by $Attacker_3$ on real and fake ciphertexts. The attacker is producing probabilities that are close to 0.5 for each of the two inputs meaning that this attacker is not able to distinguish between real and fake ciphertexts to tell which one the original message P has been encrypted to.

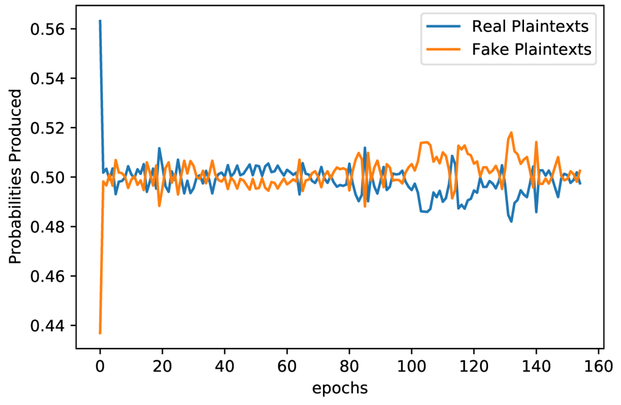

Fig. 7. Probabilities produced by $Attacker_4$ on real and fake plaintexts. The attacker is producing probabilities that are close to 0.5 for each of the two inputs meaning that this attacker is not able to distinguish between real and fake plaintexts to tell which one has been encrypted to the real ciphertext C.

secret sharing will produce outputs that are secure against the aforementioned cryptographic attacks and attackers.

The security and limitations of the proposed encryption method and by transition the secret sharing scheme are the same as the limitations of the models proposed by Li et al. [4], Coutinho et al. [5] and purely rely on the strength of the generated encrypted results. As long as the encrypted outputs are secure, the shares in secret sharing will also be secure.

A. *Robustness of the encryption in Secret Sharing*

The robustness of the encryption is the same as in the model proposed by Li et al. [4] (OTP encryption) because we use the same model as them. The results show that the encrypted outputs are resistant to different cryptographic attacks such as chosen plaintext attacks, chosen ciphertext attacks and cannot be decrypted without the key by a neural network that has the same neural network structure as Alice. As for the secret sharing scheme, all shares except for one consist of a random key and synchronized random parameters, which convey no information about the secret. The last share is the iterative encryption of the secret, using $N-1$ random keys and parameters, therefore the security of this construction is equivalent to the security of the encryption itself.

B. *Time Complexity*

The time complexity for encryption and decryption operations depends on the data size. We used a size of 64 bits for the plaintext, ciphertext and secret keys in our example. The Big-O time complexity for encryption/decryption can be calculated as follows: The neural network will receive a 128 bits input (64 bits key and 64 bits plaintext or ciphertext depending if it's an encryption or decryption). Let $n = 128$ in our example be the input size of the neural network.

- The first fully connected layer has an input size of n and its operations consist of multiplying the input by the weights matrix. Therefore it has a complexity of $O(n^2)$.
- The following resblock contains 2 convolutional layers with 2 filters, a kernel size of 2 and a stride of 1. Having a stride of 1, we will slide the kernel n times over the input. Every convolution with the $2x1$ kernel will result in 2 multiplications and 1 addition operations for a total of 3 operations. The 3 operations will repeat n times for each convolutional layer because we have a stride of 1. Therefore, the Resblock's overall complexity is $O(2*(3*n)) = O(n)$.
- The next convolutional layer has a kernel size of 4 and a stride of 2 which means this kernel will be slid over the input $n/2$ times over the input. Every convolution with the $4x1$ kernel will result in 4 multiplications and 3 additions for a total of 7 operations which will repeat $n/2$ times. Therefore, This convolutional layer's complexity is $O(7*n/2) = O(n)$.
- The following Resblock has a stride of 1 and a kernel size of 4. The kernel will be slide over the input n times. Every convolution with the kernel of size $4x1$ will result in 4 multiplications and 3 additions for a total of 7 operations which will repeat n times. Therefore, the complexity for this layer is $O(7*n) = O(n)$.
- The last layer, a convolutional layer with a kernel size of 1 and a stride of 1. The kernel of size $1x1$ will be slid over the input n times. Each time we slide the kernel over the input, we have only 1 multiplication operation and therefore this layer's complexity is $O(n)$.
- Therefore, the overall complexity for an encryption or decryption operation is $O(n^2+n+n+n+n) = O(n^2)$.
- The activation functions are composed of a single non-iterative instruction and therefore have a complexity of $O(n)$. The complexity will remain $O(n^2)$ even when taking them into consideration.

We note that this is the time complexity of the model when using simple matrix multiplications without any acceleration library. This only applies to simple forward pass propagation in order to perform an encryption or decryption operation.

C. *Space Complexity*

As for space complexity, our model built using Keras and Tensorlfow has a total of 16705 trainable parameters.

These parameters are stored as 4 bytes floating numbers. This means that our model parameters will occupy a total of $16705 \cdot 4 = 66820$ bytes of memory. This is approximately 65.25 megabytes of memory.

VI. Conclusion

We proposed a multi-party adversarial encryption model based on the works of Li et al. [4], Coutinho et al. [5] and forwarding the work of Meraouche et al. [6] from 3-party to multi-party. This work differs from [6] in the sense that a new neural network structure and training model is used to generate ciphertexts that are more secure against different types of attacks. Also, the works proposed by Li et al. [4], Coutinho et al. [5] were for two party communication while ours is for multi-party communications. We obtain an encryption technique which learns the *One Time Pad* among several neural networks and against stronger adversaries than the one Meraouche et al. [6] has considered earlier. Our methodology can be a candidate for providing post-quantum security when multiple servers/NNs learn to communicate among themselves. As an application, we show how to build an information theoretic secure secret sharing scheme for General Access Structures. In the proposed secret sharing scheme, the Dealer trains and synchronizes with multiple shareholders and then splits a secret into N shares and distributes it among them.

As future work, it would be interesting to experiment on synchronizing neural networks that do not have the same neural network structure.

Acknowledgment

Ishak Meraouche is financially supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT) for his studies at Kyushu University.
Sabyasachi Dutta is financially supported by MITACS Accelerate fellowship, Canada vide Ref. No. IT25625, FR66861.
The authors express their gratitude to Dr. Rodrigo Roman for his comments to improve the paper.
This research is supported by the Telecommunications Advancement Foundation (TAF) of Japan and also operating as a part of the India-Japan Cooperative Science Programme (IJSCP) by DST and JSPS.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[2] M. Abadi and D. G. Andersen, "Learning to protect communications with adversarial neural cryptography," *CoRR*, vol. abs/1610.06918, 2016.

[3] L. Zhou, J. Chen, Y. Zhang, C. Su, and M. A. James, "Security analysis and new models on the intelligent symmetric key encryption," *Computers & Security*, vol. 80, pp. 14 – 24, 2019.

[4] Z. Li, X. Yang, K. Shen, R. Zhu, and J. Jiang, "Information encryption communication system based on the adversarial networks foundation," *Neurocomputing*, vol. 415, pp. 347–357, 2020.

[5] M. Coutinho, R. de Oliveira Albuquerque, F. Borges, L. J. García-Villalba, and T. Kim, "Learning perfectly secure cryptography to protect communications with adversarial neural cryptography," *Sensors*, vol. 18, no. 5, p. 1306, 2018.

[6] I. Meraouche, S. Dutta, and K. Sakurai, "3-party adversarial cryptography," in *Advances in Internet, Data and Web Technologies*, L. Barolli, Y. Okada, and F. Amato, Eds. Cham: Springer International Publishing, 2020, pp. 247–258.

[7] J. Hayes and G. Danezis, "Generating steganographic images via adversarial training," *Advances in Neural Information Processing Systems*, vol. 30, pp. 1954–1963, 2017.

[8] M. Yedroudj, F. Comby, and M. Chaumont, "Steganography using a 3-player game," *Journal of Visual Communication and Image Representation*, vol. 72, p. 102910, 2020.

[9] Y. Ke, M. Zhang, J. Liu, and T. Su, "Generative steganography with kerckhoffs' principle based on generative adversarial networks," *CoRR*, vol. abs/1711.04916, 2017.

[10] I. Meraouche, S. Dutta, and K. Sakurai, "3-party adversarial steganography," in *International Conference on Information Security Applications*. Springer, 2020, pp. 89–100.

[11] I. Kanter, W. Kinzel, and E. Kanter, "Secure exchange of information by synchronization of neural networks," *EPL (Europhysics Letters)*, vol. 57, 02 2002.

[12] A. Klimov, A. Mityagin, and A. Shamir, "Analysis of neural cryptography," in *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, ser. Lecture Notes in Computer Science, Y. Zheng, Ed., vol. 2501. Springer, 2002, pp. 288–298. [Online]. Available: https://doi.org/10.1007/3-540-36178-2_18

[13] E. Salguero, W. Fuertes, and J. Lascano, "On the development of an optimal structure of tree parity machine for the establishment of a cryptographic key," *Security and Communication Networks*, vol. 2019, pp. 1–10, 03 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] A. Beimel, "Secret-sharing schemes: A survey," in *Coding and Cryptology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 11–46.

[16] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979. [Online]. Available: http://doi.acm.org/10.1145/359168.359176

[17] G. R. Blakley, "Safeguarding cryptographic keys," in *AFIPS 1979*, 1997, pp. 313–317.

[18] M. Ito, A. Saito, and T. Nishizeki, "Multiple assignment scheme for sharing secret," *J. Cryptology*, vol. 6, no. 1, pp. 15–20, 1993. [Online]. Available:

https://doi.org/10.1007/BF02620229

[19] E. F. Brickell, "Some ideal secret sharing schemes," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1989, pp. 468–475.

[20] M. Nojoumian and D. R. Stinson, "Sequential secret sharing as a new hierarchical access structure," *J. Internet Serv. Inf. Secur.*, vol. 5, no. 2, pp. 24–32, 2015. [Online]. Available: http://isyou.info/jisis/vol5/no2/jisis-2015-vol5-no2-02.pdf

[21] T. Tassa, "Hierarchical threshold secret sharing," *Journal of cryptology*, vol. 20, no. 2, pp. 237–264, 2007.

[22] H. Krawczyk, "Secret sharing made short," in *Annual international cryptology conference*. Springer, 1993, pp. 136–146.

[23] M. Naor and A. Shamir, "Visual cryptography," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1994, pp. 1–12.

[24] W. Zheng, K. Wang, and F.-Y. Wang, "Gan-based key secret-sharing scheme in blockchain," *IEEE Transactions on Cybernetics*, vol. 51, no. 1, pp. 393–404, 2021.

[25] X. Wang, H. Shan, X. Yan, L. Yu, and Y. Yu, "A neural network model secret-sharing scheme with multiple weights for progressive recovery," *Mathematics*, vol. 10, no. 13, p. 2231, 2022.