Accepted to Journal of Visual Communication and Image Representation, Elsevier, 2019 - version of september 2020.

Steganography using a 3-player game

Mehdi YEDROUDJ
LIRMM, Univ. Montpellier, CNRS, Montpellier, France, Mehdi.Yedroudj@lirmm.fr
Frédéric COMBY
LIRMM, Univ. Montpellier, CNRS, Montpellier, France, Frederic.Comby@lirmm.fr
Marc CHAUMONT
LIRMM, Univ Montpellier, CNRS, Univ. Nîmes, Montpellier, France, Marc.Chaumont@lirmm.fr

Abstract—Image steganography aims to securely embed secret information into cover images. Until now, adaptive embedding algorithms such as S-UNIWARD or Mi-POD, were among the most secure and most often used methods for image steganography. With the arrival of deep learning and more specifically, Generative Adversarial Networks (GAN), new steganography techniques have appeared. Among them is the 3-player game approach, where three networks compete against each other. In this paper, we propose three different architectures based on the 3-player game. The first architecture is proposed as a rigorous alternative to two recent publications. The second takes into account stego noise power. Finally, our third architecture enriches the second one with a better interaction between embedding and extracting networks. Our method achieves better results compared to existing works [1], [2], and paves the way for future research on this topic.

Index Terms—Steganalysis, deep learning, CNN, GAN

1 Introduction

In his paper, Simmons [3] formalized the reasoning framework for the steganography/steganalysis domain. It is defined as a *3-player game*. The steganographs, usually named Alice and Bob, want to exchange a message without being suspected by a third-party. They need to create a secret communication channel in order to converse privately. So, they use a common medium, for example, an image, and dissimulate in this image a message. The steganalyst, usually named Eve, is observing the exchanges between Alice and Bob. If these exchanges are images, Eve has to check if they are natural (cover images) or if they hide a message (stego images).

In the passive scenario, Eve does not modify the images [3]; Eve's role is only to make a binary decision, i.e. a two-class classification. Usually, in laboratory conditions [4], Eve has to be clairvoyant, meaning that she knows or has a good estimation of all the public parameters used by Alice and Bob, but she does not know their private parameters. These hypotheses about Eve's knowledge are close to the Kerckhoffs' principles [5] used in cryptography and are interesting when one wants to evaluate or compare the empirical security of steganographic embedding algorithms.

Modern embedding algorithms are adaptive, meaning that they take into account the content of the hosting medium (the cover) in order to better hide the message [6], [7], [8].

Even if modern embedding approaches are the result of almost 20 years of research using codes and adaptivity, from a game theory point of view, these algorithms are qualified as *naive adaptive steganography* [9], [10]. Indeed, when creating an embedding algorithm, the evolution of Eve's steganalysis strategy is not taken into account.

It is more interesting to propose an *optimal adaptive steganography* [10], also called *strategic adaptive steganography*. With such a steganography algorithm, pixels that would not have been modified by a naive approach have a chance to be modified. In other words, in a *strategic adaptive steganography*, the pixels' modification probability is set to ensure the Nash equilibrium in the cat-and-mouse game between Alice/Bob and Eve.

Strategic adaptive steganography is a very nice concept, but trying to formalize it mathematically often requires simplifying assumptions which are far from modelling the practical reality. Another way to obtain a Nash equilibrium is to "simulate" the game. Alice can play the game alone (from her side and without interacting with Bob or Eve) by using *three algorithms*: the embedding algorithm, the extracting algorithm, and the steganalysis algorithm, which are competing against each other. We will name these algorithms *agents*; and more precisely, we will name *Agent-Alice* the embedding algorithm, *Agent-Bob* the extracting algorithm, and *Agent-Eve* the steganalysis algorithm, thus making a distinction with the *Human users* Alice (sender), Bob (receiver), and Eve (warden). Once, an equilibrium is achieved, Alice keeps her *strategic adaptive embedding* algorithm (*Agent-Alice*), and can send the extracting algorithm

(*Agent-Bob*), or any equivalent information to Bob [1].

In reference to Simmons' formalization [3] and the computer science point of view (algorithms notion), we decided to name the approaches relying on the three agents, the *3-player game* approaches. The reader should nevertheless be aware that from a game theory point of view, there are only two teams that are competing (Alice plus Bob from one side, and Eve from the other side) in a zero-sum game. We believe that the *"3-player game"* naming, better highlights the difference with the other families relying on adversarial approaches [11].

In the steganography domain, the pioneering approaches in order to find a *strategic equilibrium* date from 2011 and 2012, and were proposed in MOD [12] and in ASO [13] algorithms. Each of these two embedding-approaches iterates until a stopping criterion is reached between i) the embedding cost map update by Alice while requesting an Oracle (this is equivalent to an adversarial attack against a discriminant), and ii) the Oracle's update (update of the discriminant).

In 2016, the authors of [14] proposed a cryptographic toy example: an encryption algorithm using three Neural Networks. The use of Neural Networks facilitates a *strategic equilibrium* since the problem is expressed as a min-max problem. Moreover, its optimization could be completed through the well-known back-propagation optimization process. Naturally, this *3-player game* concept can be transposed in the steganography domain using deep learning.

In December 2017 [1] and September 2018 [2], two different teams from the machine learning community proposed, at NIPS 2017 and during ECCV 2018, to define *strategic embedding*, using 3 CNNs, iteratively updated, and playing the roles of the Agent-Alice, Agent-Bob, and Agent-Eve. These two papers provide an overview of the *3-player game* concept, but the security notions and their evaluation are not treated correctly. When Eve is clairvoyant, both approaches are, in reality, very detectable.

More generally, the *3-player game* approach belongs to one of the four GAN families, used in steganography [11]. These four families are the *no-modification/synthesis* SWE [15], the *probability map generation* ASDL-GAN [16], the *adversarial* ADV-EMB [17], and finally, the *3-player game*. In this paper, we only focus on the 3-player game approach. This approach requires the use of 3 CNNs and is totally different from the way the other families treat the problem. Therefore we will not compare our approach to the other families that are emerging. The philosophy of this paper is to clarify the *3-player game* concept and propose practical solutions.

In this paper, section 2 focuses on the steganography's main concept with the *3-player game*. In section 3, we recall the propositions given in [1] and [2]. In section 4, we present three architectures in order to resolve previous unsolved problems. In section 5, we give some experimental results and their analysis. Finally, we conclude in section 6.

2 THE 3-PLAYER GAME CONCEPT

Notations: for this document, lowercase letters in bold are for vectors and matrices, lowercase letters in italic represent scalars.

"$\times$" is used to separate the dimensions of multi-dimensional vectors and "$\cdot$" represents a multiplication.

Let $\mathbf{x} \in \{0,...,255\}^{w\times h}$ be a cover matrix composed of $w\times h$ pixels, and $\mathbf{y} \in \{0,...,255\}^{w\times h}$ be a stego matrix with a size of $w\times h$ pixels generated by **Agent-Alice**. Let us further note **m** a secret binary message vector of *m* bits that **Agent-Alice** wants to send to **Agent-Bob**, and **m′** the binary message extracted by **Agent-Bob**, where **m′** has the same length as **m**. Let **k** be the shared key between **Agent-Alice** and **Agent-Bob,** where **k** is a *k*-sized binary vector. We note $\mathbf{z} \in \{0,...,255\}^{w\times h}$ an image with an unknown label. We use the notation *l* for the image label where $l \in$ {0,1}, *l*=0 if **z** is a cover, and *l* =1 if **z** is a stego.

2.1 General concept

This part of the paper introduces the general concept of the *3-player game* and describes the role of each agent. The *3-player game*-based steganographic system illustrated in Fig. 1 is composed of three neural networks. These networks represent the three agents: Agent-Alice, Agent-Bob, and Agent-Eve.

The system's input consists of a cover image **x**, a secret message **m** and a key **k**. These inputs are first introduced to Agent-Alice's network that generates a non-discrete-stego $\tilde{\mathbf{y}}\in \mathbb{R}^{w\times h}$. Then the discretization module receives $\tilde{\mathbf{y}}$ and generates **y** a stego image with discrete values. **y** is then given to both Agent-Bob and Agent-Eve.

Agent-Bob tries to recover the secret message **m** from the stego **y** using the shared secret key **k**. Agent-Bob inputs (**y**, and **k**) and goes through a set of layers and mathematical operations; the extracted message **m′** is then generated.

Agent-Eve receives an image **z**, and returns a probability score of **z**'s membership to the cover or stego classes.

2.2 Three agent's losses

The objective of the steganographic system shown in Fig. 1 and described previously is to learn a model, so Agent-Alice can generate a stego **y** by embedding the secret message **m** within the cover **x**, and then secretly communicate it to Agent-Bob. Given this objective a loss function is given to each agent:

Agent-Eve's loss: Let an image $\mathbf{z} = (\mathrm{z}_{ij})^{w\times h}$ whose label *l* is unknown to Agent-Eve. Agent-Eve is modelled by a function $Agent\text{-}Eve$: $\mathbf{z} \rightarrow [0, 1]$, which takes the image **z** and returns a real score between 0 and 1, such that 0 corresponds to a cover and 1 corresponds to a stego.

Agent-Eve's general loss consists in minimizing the distance between the label *l* and Agent-Eve's prediction:

$$\mathcal{L}_{Eve} = dist(l - Agent\text{-}Eve(\mathbf{z})). \quad (1)$$

The distance used for Agent-Eve's loss is usually the cross-entropy distance; thus the loss of Eq. 1 is given as:

$$\begin{aligned}\mathcal{L}_{Eve} = &- l \cdot log(Agent\text{-}Eve(\mathbf{z})) \\ &- (1 \text{ - } l) \cdot log(1 - Agent\text{-}Eve(\mathbf{z})). \end{aligned} \quad (2)$$

[1] This initial transfer from Alice to Bob is equivalent to the key exchange problem and will not be discussed in this paper.

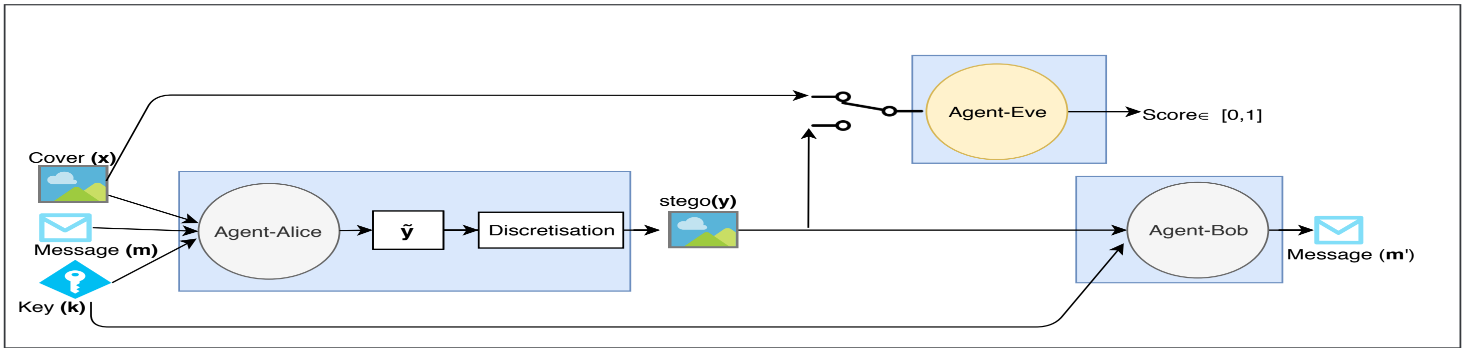

Fig. 1: The overall architecture of the *3-player game*.

Agent-Bob's loss: Agent-Bob attempts to reconstruct the secret message **m** from the received image **y** using the key **k**. The reconstructed message **m′** should be equal to **m** (**m** = **m′**). To this end, Agent-Bob's loss consists in minimizing a distance between **m′** and **m** (usually a L2 distance):

$$\mathcal{L}_{Bob} = dist(\mathbf{m}, \mathbf{m'}). \tag{3}$$

Agent-Alice's loss: Agent-Alice's objectives are multiple. The first is to generate a stego image **y** that is close enough to the cover **x**. The second is to allow Agent-Bob to reconstruct the secret message **m** correctly from the stego image. The third is that Agent-Eve's accuracy should not be better than a random guess whether a given image **z** is a cover or stego (50-50 chance of making the correct guess). The loss of Agent-Alice is then the weighted sum of three terms: $\mathcal{L}_{bob}$, $\mathcal{L}_{Eve}$, and $dist$(**x**,**y**) the distance calculated between **x** and **y**, where all coefficients λ_A, λ_B, λ_E belongs to [0,1] and sum to one in order to adjust the contribution of each term to the loss of Agent-Alice:

$$\mathcal{L}_{Alice} = \lambda_A \cdot dist(\mathbf{x},\mathbf{y}) + \lambda_B \cdot \mathcal{L}_{Bob} - \lambda_E \cdot \mathcal{L}_{Eve}. \tag{4}$$

Note that pixel values from **x** and **y** are all normalized by a division by 255. So, each of the three terms has similar value ranges, which is a practical requirement in an optimization process (see Fig. 8).

2.3 Training process

Now that we have presented the general concept of the *3-player game* and the loss for each agent, we present the algorithm used for the training process.

As shown in **Algorithm 1** on line 1, the global system is trained at the maximum for *max-iter* "*loop*". In each loop, the learning is completed sequentially by first, the team Agent-Bob and Agent-Alice (line 2) and then, the Agent-Eve (line 8). Note that there is a high number of loops in order to reach an equilibrium. Also, note that inside each loop there is also a certain number of back-propagation iterations for each agent.

Therefore, for the learning of Agent-Bob and Agent-Alice (lines 2 to 7), there are *it1* iterations (line 2). For an iteration, we load a mini-batch of *cover images*, *secret messages*, and *keys* (line 3), we forward-propagate all the cover images on Agent-Bob's and Agent-Alice's networks (line 4), and then we update Agent-Bob and Agent-Alice by minimizing the $\mathcal{L}_{Bob}$ and $\mathcal{L}_{Alice}$ thanks to the stochastic gradient descent (lines 5 and 6). During this learning phase, the weights of Agent-Eve are fixed.

Algorithm 1: *3-player game* training process

```
   Result: stegos, extracted_messages
   Data: covers-list, messages-list, keys-list
1  while not converge OR loop $\leq$ max-iter do
       // Alice and Bob learning
2      for $iter\_team1 \leq it1$ do
3          get_batch (covers_list,
             messages_list,keys_list,batch_size);
4          forward-propagation (covers, messages,keys);
5          update_Agent-Bob ($\mathcal{L}_{Bob}$);
6          update_Agent-Alice ($\mathcal{L}_{Alice}$);
7      end
       // Eve learning
8      for $iter\_team2 \leq it2$ do
9          get_batch (covers_list, stegos_list);
10         forward-propagation (covers, stegos);
11         update_Agent-Eve ($\mathcal{L}_{Eve}$);
12     end
13 end
```

The learning of Agent-Eve (lines 8 to 12) is similar to the learning of Agent-Bob and Agent-Alice. There are *it2* iterations (line 8). For an iteration, we load a mini-batch of *cover images* and *stego images* (line 9), we forward-propagate all the cover and stego images on Agent-Eve's network (line 10), and then we update Agent-Eve by minimizing the $\mathcal{L}_{Eve}$ thanks to the stochastic gradient descent (lines 11).

When the equilibrium is reached, the last agent playing the game will not change its strategy. So, the fact to be the last player, i.e. the last learning agent, will not impact the performances of the other agents.

3 RELATED WORK

In this section, we recall the architectures of GSIVAT [1] and HiDDen [2]. These two architectures were basically proposed for steganography purposes. They belong to the *3-player game* family.

3.1 Generating Steganographic Images Via Adversarial Training (GSIVAT)

In [1], the authors propose a steganographic system (GSIVAT) composed of three neural networks, each one repre-

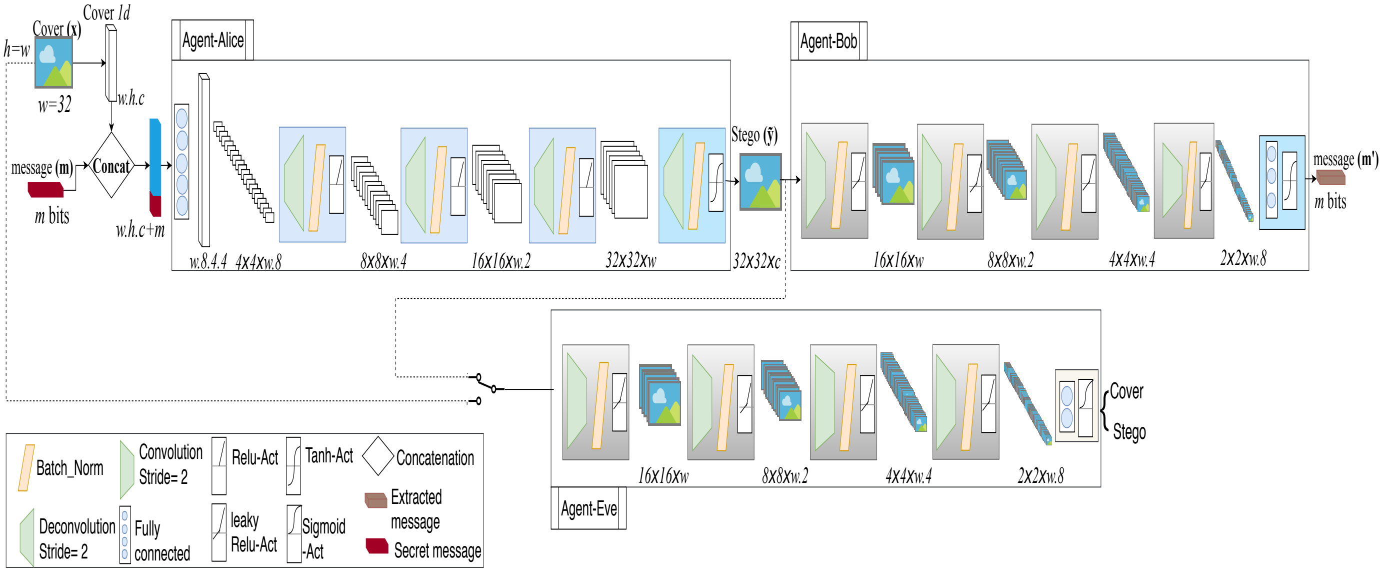

Fig. 2: GSIVAT [1] architecture.

senting one agent (Agent-Alice, Agent-Bob, Agent-Eve). We provide an overview of the GSIVAT architecture in Fig. 2.

Their system's input are a cover image **x**, a *3D* vector, whose size is $w \times h \times c$ (where c is the channel number) and a secret message **m** of m bits. **x** is flattened to a 1D vector and concatenated with **m**, the resulting vector size is $w{\cdot}h{\cdot}c{+}m$. This vector is then fed to Agent-Alice.

Agent-Alice is composed of one fully connected layer (FC) and four deConv-BN-Act blocks (see Fig. 2). Each block is the combination of a deconvolution (deConv), a batch normalization (BN) and one activation function (Act) layer (ReLU, leaky-ReLU, sigmoid, tanh).

The number of neurons in the fully connected (FC) layer is the multiplication result of $w{\cdot}8{\cdot}4{\cdot}4$. The output of the FC is reshaped to $w{\cdot}8$ of $4{\times}4$ sized feature-maps. Then, these feature-maps feed four *deConv-BN-Act blocks*. After each block, the size of the feature-maps is up-sampled by a factor of two, and the channel number is reduced by a factor of two. In the end, an image of the same size as **x** is obtained.

Agent-Alice outputs the stego image $\tilde{\mathbf{y}} \in \mathbb{R}^{w \times h}$ which is fed to both Agent-Bob and Agent-Eve. The distance between **x** and $\tilde{\mathbf{y}}$ (dist(**x**,**y**) in Eq. 4) is the L2 distance.

Agent-Bob's architecture is depicted in Fig. 2. It consists of four *Conv-BN-Act* blocks and one FC layer. Each block is made up of a fractionally-strided (stride=2) convolution layer (Conv), a batch normalization layer and an activation function "Leaky ReLU" [18]. The FC layer is made up of m neurons followed by a tanh activation function.

Agent-Bob outputs the extracted message **m′**. The L2 distance is used to compute the loss previously given in Eq. 3.

Agent-Eve takes as input image **z**, and outputs a score determining the label of the input (cover/stego). The architectures of Agent-Eve and Agent-Bob are similar, except for the fully connected layer having only two neurons, and the activation function replaced by a sigmoid. The sigmoid cross-entropy loss is used for Eve's loss (see Eq. 2).

3.2 HiDDeN: Hiding Data With Deep Networks

In [2], the authors propose a model for image steganography, but also for watermarking (called HiDDen). We will only discuss the architecture of their steganographic system.

Their system takes as input a $w \times h \times c$-sized cover image **x** and a m-length secret message **m**. The system is composed of three neural networks as illustrated in Fig. 3. The network's structure is generally composed of blocks called *Conv-BN-ReLU* which stands for the combination of Convolution, Batch Normalization and a ReLU activation function.

Agent-Alice is composed of five *Conv-BN-ReLU* blocks and one convolution layer with a kernel size of 1×1. Firstly, the cover image **x** goes through four *Conv-BN-ReLU* blocks to obtain an intermediate representation image $\tilde{\mathbf{x}}$. Firstly, the message **m** is replicated so that the resulting size is $w \times h \times m$. Secondly, $\tilde{\mathbf{x}}$ is concatenated with the replicated message and fed to another *Conv-BN-ReLU* block with 64 output filters. A final convolution layer with a 1×1 kernel is used to generate the stego image $\tilde{\mathbf{y}} \in \mathbb{R}^{w \times h}$. The loss of Agent-Alice is calculated using the loss of Eq. 4.

Agent-Bob is composed of seven *Conv-BN-ReLU* blocks, followed by global spatial average pooling (to produce a vector with the same size as the message). Then, a single fully connected layer ends the architecture, as shown in Fig. 3. Agent-Bob receives the stego image $\tilde{\mathbf{y}}$, and produces the predicted message **m′**. The distance used in Agent-Bob's loss of Eq. 3 is the l2 distance.

Agent-Eve takes an image as input, and outputs a score indicating whether the given image is a cover or stego. Agent-Eve has an architecture similar to Agent-Bob, but only has three *Conv-BN-ReLU* blocks instead of seven. The last layer is a FC layer with an output size of two units (see Fig. 3). The authors adopt the use of the cross-entropy loss presented in Eq. 2 for Agent-Eve.

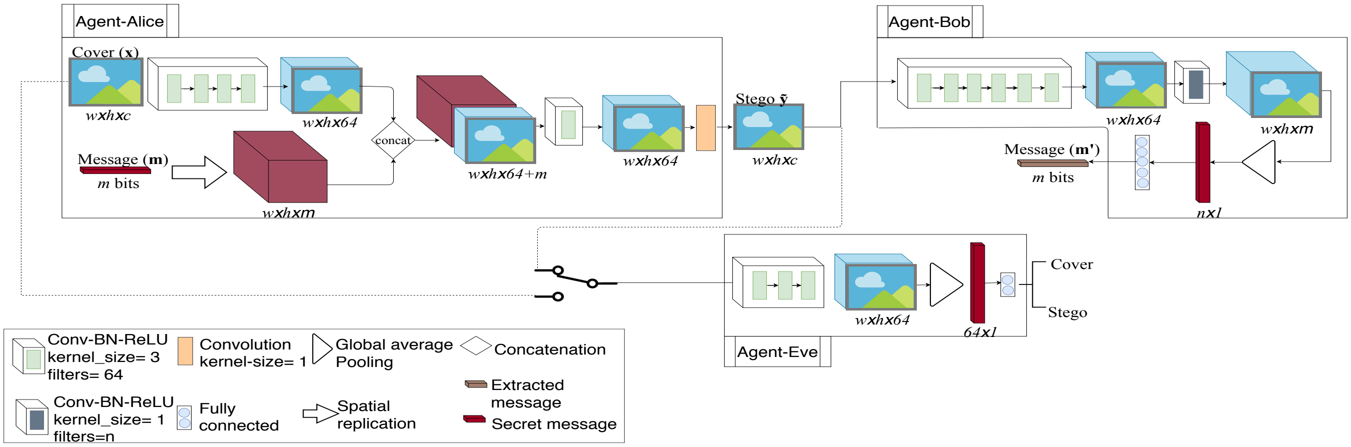

Fig. 3: HiDDeN architecture.

3.3 Discussion

The two papers presented previously [1] [2] offer some interesting ideas, but there are flaws in both.

Firstly, neither of the two approaches use a shared secret key during the embedding/extracting process. The authors of [1] and [2] suppose that the information about model weights, architecture, and the set of images used for training is shared between Agent-Alice and Agent-Bob. According to the authors, this shared information can be considered as the secret key. Besides the fact that such a hypothesis is heavy in size (almost 70 Mb sent to Agent-Bob [1]), it is also the equivalent of performing an embedding process with the same *secret-key*. Such an embedding process is highly discouraged in steganography as it leads to very easy detectability [19].

Secondly, there is no discretization module for the generated images (Agent-Alice provides $\tilde{\mathbf{y}}$ instead of $\mathbf{y}$). In a real-world situation [4], Bob receives an image whose values are defined in $\{1,..., 255\}$, and has to extract the secret message. In [1] and [2], Agent-Alice generates real-valued images i.e. **not** discrete-value images, and these images are fed to Agent-Bob. This makes both Agent-Alice and Agent-Bob useless in practice. Alice needs to provide Bob with images that are not suspicious, meaning images with discrete values. Indeed, images have to be formatted in PGM, or any lossless compressed image format. Note that if Alice decides to round the real-value images (generated by Agent-Alice) in order to discretize them in $\{1,..., 255\}$, Bob, when using Agent-Bob algorithm, will not extract the message correctly, since Agent-Bob has been built for real-value images[2].

Thirdly, the computation load is a serious issue that we need to take into account when working with deep learning. GSIVAT authors [1] worked on 32×32-sized images while Hidden authors [2] used 16×16-sized patches. This limitation for the size of the images is due to the use of the FC layers, which introduce expensive memory and computation costs. The authors suggest that working on large images could be completed by treating bigger images with a separate treatment for each part of the image. This is a bad idea since statistical traces may be found at the block boundaries and would lead to an easily detectable embedding scheme (See for example, the discussion in section 4.2 of [20], or the dependencies preservation between blocks in JPEG steganography [21]).

Finally, note that in both of these papers, the experimental steganalysis is performed with the algorithm ATS [22] proposed in 2015. This algorithm is basically designed to handle the cover-source mismatch problem, which is definitely not the appropriate scenario to evaluate the empirical security of an embedding algorithm (especially when it is a *strategic embedding* algorithm). Indeed, ATS is based on the assumption of *constant noise direction* in the embedding space, which may not be true for *a strategic adaptive* algorithms. The empirical security is probably undervalued when compared to an Ensemble Classifier/Rich Model (EC+RM) [23], [24], Yedroudj- Net [25], ReST-Net [26], or SRNet [27]. In addition, these four steganalysis algorithms represent the current state-of-the-art in steganalysis, so their use makes more sense.

4 Our steganographic system's Architecture

In this paper, we propose a new *strategic adaptive steganography* system based on the *3-player game* concept. We are using an embedding algorithm (Agent-Alice) and an extracting algorithm (Agent-Bob) which functions in practice. We therefore:

1) Integrate a stego-key for the input of Agent-Alice and Agent-Bob. With two different stego-keys, Agent-Alice will generate two different stego images. Alice knows that she must change the stego-key very often if she doesn't want to be caught [19]. By extension, knowing that it is easier to break a system that always uses the same key, it is important to integrate a stego key in the input of Agent-Alice and Agent-Bob, in order to avoid the counterproductivity that a unique key could have on the convergence of Agent-Alice and Agent-Bob facing Agent-Eve. This argument is not considered at all in [1] and [2] and can be a major flaw in their performances.

[2] We observed this phenomenon during our experiments.

2) Handle the problem of discretization in order for Alice to be able to send to Bob, through e-mail, memory stick, cloud storage, an image in a non-suspicious standard format. ([1] and [2] do not deal with this fundamental issue).
3) Guarantee a scalable (in memory and in computation) solution thanks to an architecture that consists of only convolutions. This way, it can deal with image dimensions usually used in deep-learning and steganalysis by deep-learning in academic experiments (255×255 or 512×512) [11]. The convolutional architectures also allow deeper networks to deal with harder problems modelization. GSIVAT [1] works with 32×32 images and HiDDeN [2] works with 16×16 images and they both use very small networks.

Additionally, our approach offers two interesting properties. Firstly, it is "bit-rate adaptive". Indeed, there is no need to re-train the system each time we change the bit rate, i.e. each time the message size is different (this is not the case for [1] and [2]). Secondly, we adopt a strong steganalyst for Agent-Eve, which benefits from better security, but it is not the most up-to-date steganalysis.

We propose three different architectures for our steganographic system. These architectures illustrate three different solutions going from a basic one, to a more appropriate solution. On these three architectures, Agent-Eve's remains the same, while the design of Agent-Alice and Agent-Bob's changes. The first architecture is presented to illustrate the use of a secret shared key during embedding. The second architecture has been conceived to reduce the power of noise introduced by embedding the hidden message into the cover image. Finally, the third architecture tries to improve the performances of message extraction while linking Agent-Alice and Agent-Bob's behaviour.

4.1 The training process

Looking at the three proposed architecture, the training procedure of the system remains the same. We alternate the training between the three agents, Agent-Alice, Agent-Bob and Agent-Eve, where Agent-Alice and Agent-Bob are trained jointly as a single network, and Agent-Eve is trained separately.

First, Agent-Bob and Agent-Alice are trained on a fixed number of mini-batches using the two models of Agent-Alice and Agent-Bob saved previously. See Algorithm. 1. This training process is repeated for several loops, until all losses tend to be constant.

4.2 The proposed architecture of the Agent-Eve

Agent-Eve tries to guide both Agent-Alice and Agent-Bob through the process of learning a *strategic adaptive embedding* algorithm. If Agent-Eve is weak, the 3-agent system falls down. Indeed, Agent-Alice and Agent-Bob will no longer search for better solutions as Agent-Eve cannot cope with their evolution. To this end, it is essential to adopt a strong steganalyzer.

In 2018, the best spatial steganalyst was, Yedroudj-Net[25][28] (first published in January 2018), ResT-Net[26] (published in March 2018) and more recently SRNet [27] (published in September 2018). Among these networks, Yedroudj-Net is the shallowest network, with six convolution layers compared to 25 layers for SRNet, and 30 layers for ReST-Net (3 sub-networks each containing 10 layers). Besides its affordable size, training Yedroudj-Net does not require the use of any tricks that could increase the computational time. This network is therefore well adapted to Agent-Eve, especially knowing that *the 3-player approach* takes a lot of time before it converges to a good solution. Additionally, Yedroudj-Net can easily be improved in the future, if required [29].

Yedroudj-Net architecture [25] is presented in Fig. 4. It is composed of 7 blocks: a pre-processing block, five convolutional blocks, and a fully connected block made of three fully connected layers followed by a softmax [3].

Agent-Eve's network (Yeroudj-Net) is trained by minimizing the loss given in Eq. 2.

4.3 First-Architecture

The first architecture is similar in spirit to the previous approaches of existing literature except that it contains only convolutional layers, and integrates a stego-key.

Agent-Alice's network receives a *m*-length secret message $\mathbf{m}$, a key $\mathbf{k}$ of k bits, and a cover $\mathbf{x}$. in order to concatenate the cover $\mathbf{x}$ with the message $\mathbf{m}$, both should have the same size. So, we use the key $\mathbf{k}$ to spread out the secret message $\mathbf{m}$ in a matrix noted as $\mathbf{s}^{(\mathbf{m})} \in \{0,1\}^{w \times h}$ that is filled with zeros and has the same size as our cover image. The spreading of $\mathbf{m}$ in $\mathbf{s}^{(\mathbf{m})}$ is obtained by using a pseudo random number generator (PRNG) seeded by the key $\mathbf{k}$. The PRNG sequentially picks a bit of $\mathbf{m}$ and fills a *non-used* position in the $\mathbf{s}^{(\mathbf{m})}$ matrix. The filled positions define the binary mask $\mathbf{\Omega} \in \{0,1\}^{w \times h}$. $\mathbf{\Omega}$ therefore contains exactly m ones.

Note that with the knowledge of $\mathbf{k}$, and the index of a bit m_i in our message $\mathbf{m}$ with $i \in \{1, ..., \mathrm{m}\}$, we can deduce the position $(u,v) \in \{0,..w\} \times \{0,..h\}$ where this bit is stored in $\mathbf{s}^{(\mathbf{m})}$, and inversely from a position $(u,v) \in \{0,..w\} \times \{0,..h\}$, we can deduce the bit m_i with $i \in \{1, ..., \mathrm{m}\}$ of the message $\mathbf{m}$.

The cover image $\mathbf{x}$ is fed to a convolution layer called *SRM-F*. Its weights are initialized with the 30-basic high-pass filters of SRM [24], similarly to Yedroudj-Net [25]. The output is then concatenated with $\mathbf{s}^{(\mathbf{m})}$ and fed to *conv_Stack0* which is composed of a set of convolution layers. The output is a $w \times h$ image which represents the stego image $\mathbf{y}$ (see Fig. 5).

Agent-Bob uses the stego $\mathbf{y}$ and the key $\mathbf{k}$ to output the predicted message $\mathbf{m'}$, a vector of *m* bits. First, the stego image passes through a convolution layer similar to *SRM-F*. Next, the obtained feature-maps from *SRM-F* go through another set of convolutions in two *conv_Stack1*, *conv_Stack2* (Note that *conv_Stack2* shares the same construction as *conv_Stack0*, but not the same number of features-maps for each layer) . The extracted message $\mathbf{m'}$ is then obtained (see Fig. 5).

The back-propagation is done by minimizing both losses of Agent-Alice and Agent-Bob. For Agent-Bob's loss (see

[3] For more details on Yedroudj-Net, the reader can view the online code at www.lirmm.fr/∼chaumont/Yedroudj-Net.html

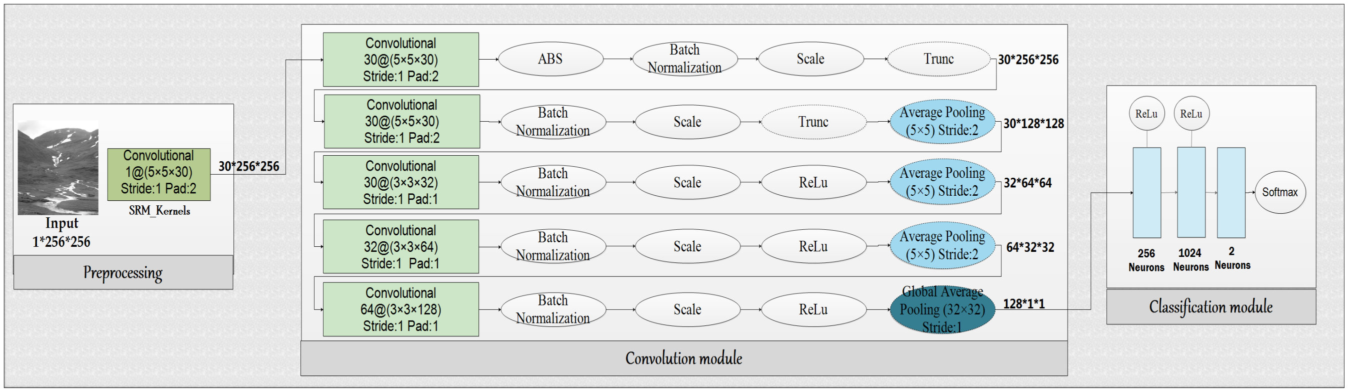

Fig. 4: The overall architecture of Agent-Eve [25].

Eq. 3), the Mean Square Error (MSE) distance between $\mathbf{m}$ and $\mathbf{m'}$ is used. It is written as:

$$\mathcal{L}_{Bob} = \left(\sum_{i=1}^{m} (m_i - m_i')^2\right) / m \,, \tag{5}$$

which is equivalent to:

$$\mathcal{L}_{Bob} = ((\mathbf{s}^{(\mathbf{m})} - \mathbf{s}^{(\mathbf{m}')}) \odot \Omega)^2 / m \,, \tag{6}$$

with $\odot$ the point-wise operation.

Agent-Alice uses the loss presented in Eq. 4 with *dist*(**x**,**y**) calculated as:

$$dist(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{w} \sum_{j=1}^{h} (x_{ij} - y_{ij})^2\right) / w.h \,, \tag{7}$$

and so Agent-Alice's loss is:

$$\mathcal{L}_{Alice} = \lambda_A \cdot dist(\mathbf{x}, \mathbf{y}) + \lambda_B \cdot \mathcal{L}_{Bob} - \lambda_E \cdot \mathcal{L}_{Eve} \tag{8}$$

where λ_A, λ_B, λ_E $\in$ belongs to [0,1] and sum to one in order to adjust the contribution of each term to the loss. In this architecture, the magnitude of modifications is not restricted and therefore the stego noise power is too strong. To impose stego noise power restriction, we propose a second architecture.

4.4 Second-Architecture (noise power reduction)

The second architecture improves the first one by imposing a stronger restriction on the magnitude of modification of stego noise. We force Agent-Alice to make the least amount of changes to the cover whilst still allowing Agent-Bob to retrieve the secret message correctly. Within this architecture, see Fig. 6, Agent-Bob's design remains the same as in the previous architecture. However, the architecture of Agent-Alice has changed. Instead of letting the network decide the intensity of modification for each pixel in the cover image, it is restricted to a ternary modification; the stego noise values are {-1, 0, 1}. More precisely, during the first iterations, values are in the range of $[-1, 1]$ and belong to $\mathbb{R}$. But at the end of iterations, a discretization is completed in order to have only three discrete values {-1, 0, 1}.

Said differently, Agent-Alice generates a modification map $\mathbf{n}\in${-1, 0, 1} which is then added to the cover image $\mathbf{x}$ to generate the stego $\mathbf{y}$ directly (see Fig. 6). The generation of $\mathbf{n}$ is performed thanks to the resulting feature maps of *SRM-F*, and the spread message $\mathbf{s}^{(\mathbf{m})}$. These are both concatenated and fed to *conv_Stack3*; the output of the latter is $\mathbf{n}$ in the range of $[-1, 1]$ thanks to a TanH activation function, and is one of the three discrete values {-1, 0, 1} once the discretization is activated.

Agent-Bob's network loss remains the same as in Eq. 5. For Agent-Alice's loss, we calculate the mean of the absolute values of the modification maps $\mathbf{n}$ (which is equivalent to the MSE) for the distance between the cover image $\mathbf{x}$ and the stego $\mathbf{y}$.

$$dist(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{w} \sum_{j=1}^{h} (|n_{ij}|)\right) / w.h \,, \tag{9}$$

One can notice that minimizing the loss using this distance, forces Agent-Alice to output only **zeros** over the map of modifications. Agent-Bob is then no longer capable of retrieving the secret message. To reduce this constraint, we introduce a constant β in Agent-Alice's loss. This β value is related to the change rate notion. So the loss becomes:

$$\mathcal{L}_{Alice} = \lambda_A \cdot (dist(\mathbf{x{,}y}) - \beta) + \lambda_B \cdot \mathcal{L}_{Bob} - \lambda_E \cdot \mathcal{L}_{Eve} \,, \tag{10}$$

where $\lambda_A, \lambda_B, \lambda_E \in [0, 1]$. Note that β controls the discretion of the embedding network, i.e. how many pixels Agent-Alice is allowed to alter from the cover image $\mathbf{x}$.

4.5 Third-Architecture (source separation)

The architecture shown in Fig. 7 is proposed as an improvement to the second architecture. The embedding part of the second architecture was changed to make as few adjustments as possible. The extracting part, on the other hand, remains the same from the first architecture. Nevertheless, constraining the amplitude of modifications directly impacts the extracting part. When we limit the number of pixels that can be modified, more errors occur during message extraction. In other words, altering fewer pixels means less detectability, but more errors during *message-extraction*, while changing more pixels means fewer errors when retrieving the message, but more detectability. How can Agent-Bob extract the secret message correctly when Agent-Alice carries out the minimal required modification?

In embedding algorithms such as S-UNIWARD [6], WOW [7], etc, the message coding requires, in practice, the use of a Syndrome-Trellis-Codes STC [30]. The extractor (Bob) has access to the parity-check matrix $\mathbf{h}\in\{0, 1\}^{w \times h}$ used by Alice during the embedding process. This matrix $\mathbf{h}$ is then shared between

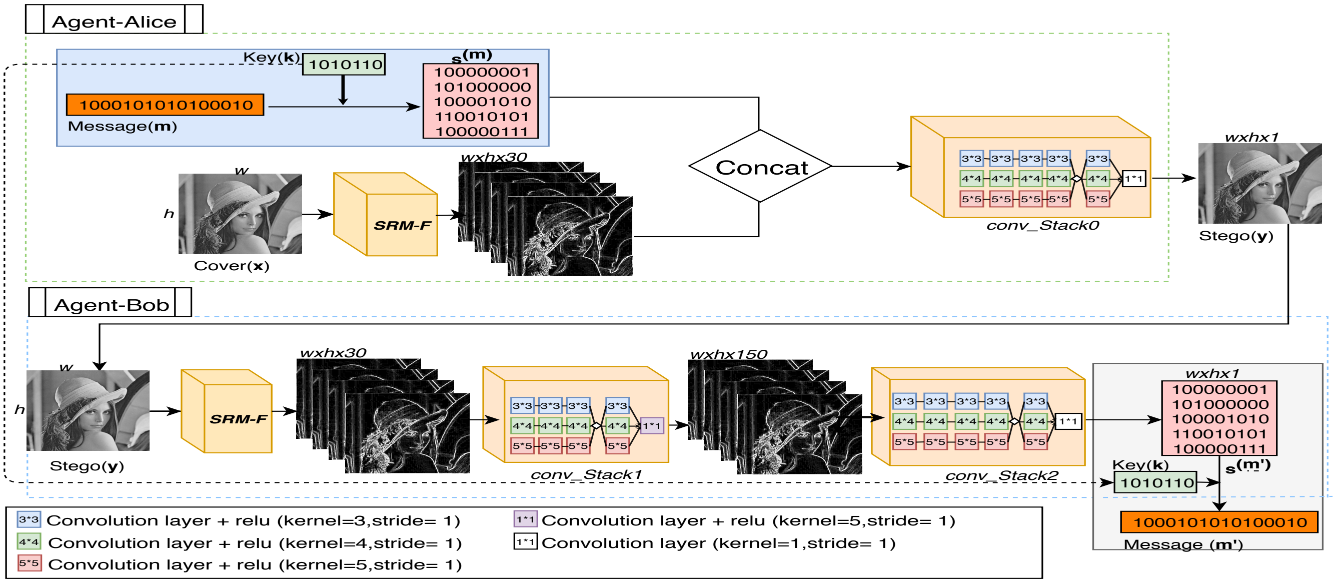
Agent-Alice
Key(k)
1010110
s(m)
100000001
101000000
100001010
110010101
100000111
1000101010100010
Message(m)
Concat
w
h
Cover(x)
SRM-F
wxhx30
3*3
4*4
5*5
1*1
conv_Stack0
wxhx1
Stego(y)
Agent-Bob
Stego(y)
wxhx150
conv_Stack1
conv_Stack2
Key(k)
s(m')
Message (m')
3*3 Convolution layer + relu (kernel=3,stride= 1)
4*4 Convolution layer + relu (kernel=4,stride= 1)
5*5 Convolution layer + relu (kernel=5,stride= 1)
1*1 Convolution layer + relu (kernel=5,stride= 1)
1*1 Convolution layer (kernel=1,stride= 1)

Fig. 5: Agent-Alice and Agent-Bob with the first architecture.

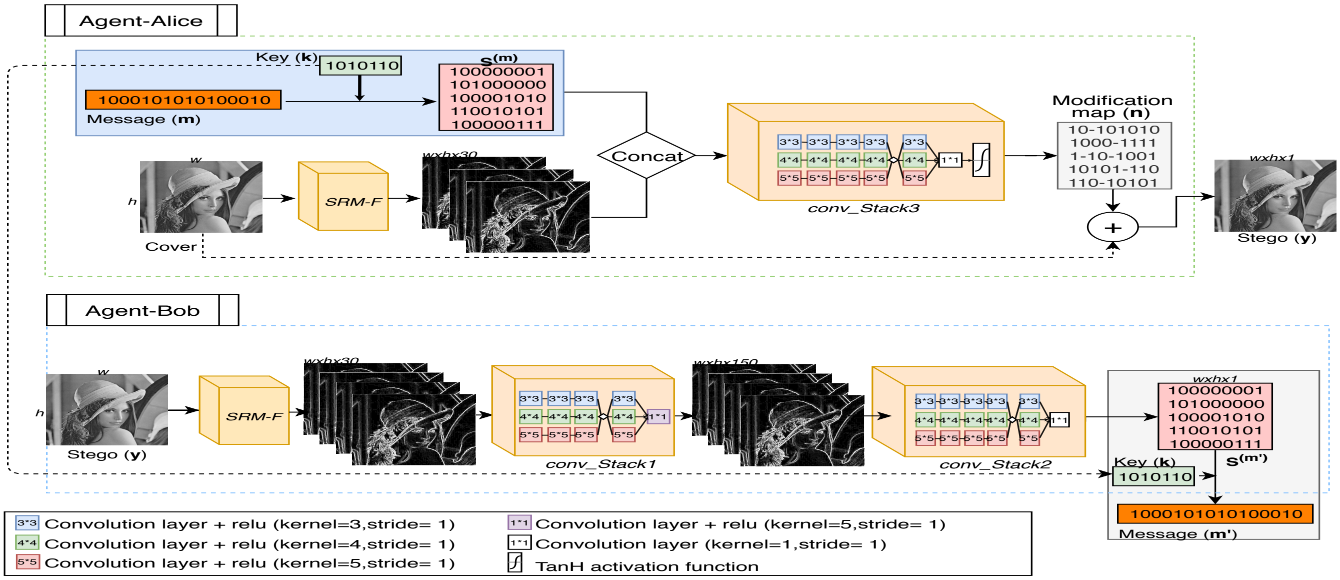

Fig. 6: Agent-Alice and Agent-Bob with the second architecture.

Alice and Bob, so Bob can easily retrieve the message from **y** by calculating the matrix product as:

$$\mathbf{m}= \mathbf{h} \cdot \mathrm{lsb}(\mathbf{y}),$$

with lsb (.) the function extracting the LSB plane.

In our proposed method, no parity-check matrix is shared between Agent-Alice and Agent-Bob, nor any related information. If Agent-Bob has to mimic an STC to perform the message extraction in the *3-player game*, it should learn a matrix in conjunction with Agent-Alice, in order to retrieve the message correctly. Without any topological or mathematical construction, this can be difficult, especially when the system used has many loss terms to minimize (Nash equilibrium issue).

A solution for this problem could be to inject a coding/decoding block inside Agent-Alice and Agent-Bob. Nevertheless, this is not an easy task, and before trying this solution, we preferred exploring the impact of increasing the link between the two agents. The integration of a coding and decoding block is postponed for future work.

We continue in the direction of increasing the link between Agent-Alice and Agent-Bob by forcing their topology to be more anti-symmetric. Referring to Fig. 7, we draw two blue rectangles to show where this anti-symmetry has been injected. On one of the rectangles, we note g, the *point-wise summation* block, and g', the *sources separation* block. g and g' can be seen as inverse functions such that:

$$\begin{aligned} g : \{-1, 0, +1\}^{w \times h} \times \mathbb{R}^{w \times h} &\rightarrow \mathbb{R}^{w \times h} \\ g(\mathbf{n}, \mathbf{x}) &= \mathbf{n} + \mathbf{x} = \mathbf{y} \\ g' : \mathbb{R}^{w \times h} &\rightarrow \{-1, 0, +1\}^{w \times h} \times \mathbb{R}^{w \times h} \\ g'(\mathbf{y}) &= (\mathbf{n}', \mathbf{x}') \end{aligned}$$

For a given cover, **x**, and a stego noise, **n**, $g'((g(\mathbf{n}, \mathbf{x}))$ gives an estimated cover, **x′**, and a estimated stego noise, **n′**.

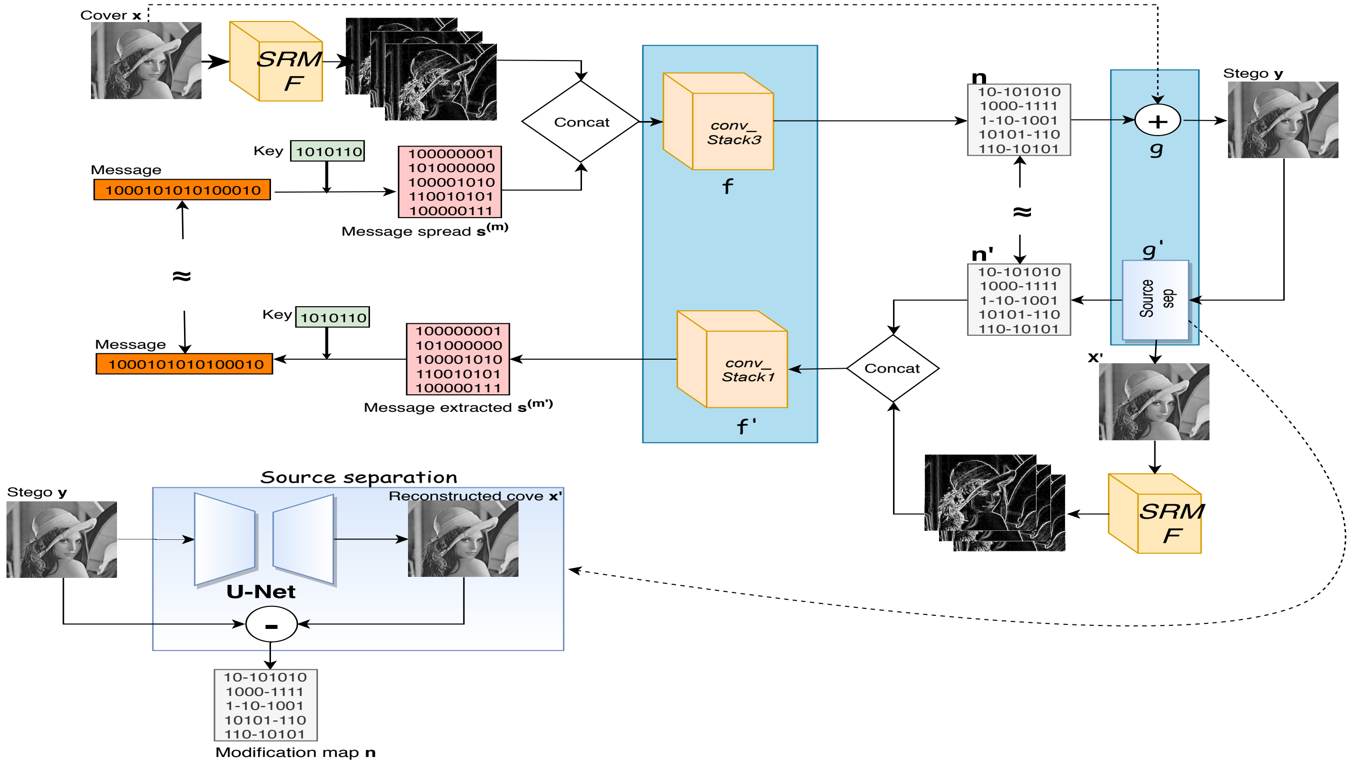

Fig. 7: Agent-Alice and Agent-Bob with the third architecture.

During learning, we then minimize the distance between **x** and **x′** (which is equivalent to minimizing the distance between **n** and **n′**); See Eq. 11.

The *sources separation* block act as a denoiser. The well-known U-Net [31] has been used and integrated into Agent-Bob's architecture (see Fig. 7). The goal of the U-Net is to reconstruct the cover image **x** from the stego **y** image. The modification map **n** is resulting from the subtraction of the reconstructed cover **x′** from the stego image **y**.

Referring to Fig. 7, on the other rectangle, we note f, the *conv_Stack3* block, and f', the *conv_Stack1* blocks. f and f' can be seen as inverse functions from the point view of the spread message $\mathbf{s}^{(m)}$ and the modification map **n** :

$$f : \mathbb{R}^{w\times h\times 30} \times \{0,1\}^{w\times h} \rightarrow \{-1,0,+1\}^{w\times h}$$
$$f_{\mathbf{r}}(\mathbf{s^{(m)}}) = \mathbf{n}$$
$$f' : \mathbb{R}^{w\times h\times 30} \times \{-1,0,+1\}^{w\times h} \rightarrow \{0,1\}^{w\times h}$$
$$f'_{\mathbf{r}'}(\mathbf{n}') = \mathbf{s}'^{(m')}$$

with **r** and **r′** the 30 residual images obtained by *SRM-F* filtering.

For a given spread message $\mathbf{s}^{(m)}$, and residual images **r** and **r′**, $f'_{\mathbf{r}'}((f_{\mathbf{r}}(\mathbf{s^{(m)}}))$ gives an estimated spread message $\mathbf{s}'^{(m')}$. During the learning, we then minimize the distance between **m** and **m′** (which isthe equivalent to minimizing the distance between $\mathbf{s}^{(m)}$ and $\mathbf{s}'^{(m')}$; see Eq. 6); See Eq. 12. Note that *conv_Stack1* and *conv_Stack3* are the same as in the second architecture.

Therefore Agent-Bob's loss is composed of two loss terms:

- the cover reconstruction loss:

$$\mathcal{L}_{cover\_recons} = MSE(\mathbf{x'}, \mathbf{x}) \qquad (11)$$

- the message extraction loss:

$$\mathcal{L}_{message\_extract} = MSE(\mathbf{m}, \mathbf{m'}) \qquad (12)$$

Agent-Bob loss is given as:

$$\mathcal{L}_{Bob} = 1/2(\mathcal{L}_{cover\_recons} + \mathcal{L}_{message\_extract}) \qquad (13)$$

5 Experiments

5.1 Dataset and software platform

In this paper, the experiments are carried out on two image sources. The first is the well-known database BOSSBase 1.01 [32] which contains 10,000 8-bit grayscale 512×512 pixels sized images. This database was created for steganalysis purposes in 2011. Images from this database offer different texture characteristics, which explains why it is widely used in steganalysis. BOWS2 [33] is our second database. It was created for a watermarking contest and consists of 10,000 8-bit grayscale 512×512 pixels sized images.

Due to our GPU computing platform, time limitation and nature of *3-player game*, which takes much time to train due to two-phase learning. We conduct all the experiments on images of 256×256 pixels. To this end, we resampled all images from the two databases from 512×512 pixels to 256×256 pixels, using the imresize() Matlab function with default parameters.

We implemented the proposed architectures presented in section 4 using TensorFlow V 1.6. As for comparison, we use S-UNIWARD [6], and WOW [7], two well-known content-adaptive methods for spatial domain embedding. All our experiments were conducted on the NVIDIA Titan X GPU platform.

5.2 Training, Validation, Test

We start the training phase by preparing a pre-trained model of Agent-Eve. For this, we use the S-Uniward algorithm to generate 10,000 stegos from the BOSSBase. 10,000 cover/stego pairs are then obtained. We use 4,000 pairs to train a Yedroudj-Net network. After several epochs of training, we obtain a learned model of Yedroudj-Net, which we transfer to Agent-Eve; Agent-Eve does not learn from scratch.

Once Agent-Eve is pre-trained, we start the learning phase of the 3-players (Agent-Alice, Agent-Bob and Agent-Eve). We use BOSSBase as the image source. The messages are generated using a PRNG [4] with a chosen payload. We use Adam, an adaptive optimizer, to train our system for all three networks where the mini-batch size is set to 4. The values of λ_A, λ_B, λ_E are set to 0.2, 0.4, and 0,4 respectively. These values are chosen empirically. However, they may not be optimum.

During training, we set the maximum number of iterations *max-iter* to 1 million iterations. The training is alternated between Agent-Alice and Agent-Bob from one side, and Agent-Eve from the other side. We set *it1* to 50 iterations, and *it2* to 1 iteration (see Algorithm.1, so Agent-Alice and Agent-Bob are trained for 50 iterations, compared to 1 iteration for Agent-Eve).

When the losses of the three networks appear to be stable, we freeze their training, and we integrate the discretization module into Agent-Alice (see Fig. 1). Then, we resume the training for several iterations (more or less 50,000 iterations), so the system learns how to generate stego images with discrete pixel values.

We should note that activating the discretization module at the beginning of the training phase prevents the system from converging (as the round function is not differentiable). So, We let the system converge towards a solution, then we force it to work on images with discrete pixel values. Once our system is well trained (loss curves are stable as shown in Fig. 8), we stop the training phase.

To evaluate the performance of our method, we measure the *security* and the *transmission errors*. The security is measured with the probability of error (Pe) of a given steganalyzer, where Pe is computed as the average of the false alarm rate and the missed detection rate. For the transmission error, we use the Bit Error Rate (BER)[5].

The reader should understand that Agent-Alice is embedding a *secret message*, noted $\mathbf{m}$, which is a binary vector. You should consider that this vector $\mathbf{m}$ is the resulting of, first, the encryption of a *clear message*, with a classical cryptographic algorithm, and second, its encoding by an error-correcting code[6]. Making this assumption is natural, because it ensures that there are no security leaks in the secret message, $\mathbf{m}$, and that there is an equiprobability of 0 and 1 in the secret message.

So, for the evaluation of the *transmission error*, we use the Bit Error Rate (BER) such that BER = $dist(\mathbf{m}, \mathbf{m}')/m$, with $dist(.)$, the Hamming distance. In this paper, the BER is thus the ratio between the number of bit errors in the *secret* binary message, $\mathbf{m'}$, *extracted* by Agent-Bob, on the number of bits from the *secret* binary message, $\mathbf{m}$, *embedded* by Agent-Alice. SO, Agent-Bob is extracting a secret **coded**-message, $\mathbf{m'}$, whith a BER which is possibly not null, but the secret **decoded**-message will have a null BER, because Alice sends only stego images to Bob, whose **decoded**-message is free from errors. Note that we will not integrate the error-correcting code block, but we will discuss it. Indeed, this paper is primarily on the 3-player concept definition and the proposition of three architectures. The reader should consider that $\mathbf{m'}$ is an encrypted and coded binary vector. All of this also implies that the "payload rate" (i.e. the embedding rate), is equals to $m/(w \times h)$, and the "real payload rate", is equal to the size of the **decoded**-message divided by $w \times h$.

Therefore for the testing phase, Agent-Alice starts generating 10,000 Stego images from BOWS2, where a random message is embedded in a cover image using a random key. The generated images are used to evaluate the accuracy of Bob's message retrieval on the one hand and the security of our steganographic system against the steganalyzer Yedroudj-Net on the other. Please note that the steganalysis with Yedroudj-Net (learning and testing) is achieved on BOWS2Base, as is the training of the 3-players (Agent-Alice, Agent-Bob, and Agent-Eve) it is carried out using the different and disjointed BOSSBase. The results are presented in the following subsection.

5.3 Results findings and discussion

5.3.1 Extraction and security comparison of the three architectures

First architecture:

In Table 1, we report the Bit Error Rate (BER) and the Probability of error (Pe) obtained using the first architecture. These tests are carried out on BOWS2 database using different payloads 0.2 bpp, 0.4 bpp and 1 bpp. The steganalyzer used is Yedroudj-Net. Regardless of the payload, Bob, who uses the Agent-Bob can correctly recover the secret message with a BER equal to zero. We can observe that this architecture is not at all secure since the detection accuracy obtained using Yedroudj-Net is between 96% and almost 98%.

The first architecture offers good embedding capacity, and we can perfectly recover the secret message even when the payload is significant, although the low security given by this architecture does not make it interesting for steganography purposes.

Note that this architecture is the closest form of architecture to GSIVAT or HiDDeN since the main principle is to spread the message in the spirit of a spread spectrum watermarking approach. This first architecture, GSIVAT, and HiDDeN are definitively insecure approaches since the stego noise power is too strong. In the second architecture We investigated a way to constrain pixels modifications to -1 or +1.

[4] PRNG: Pseudo-Random Number Generator. The PRNG generates a binary vector $\mathbf{m}$ whose values 0 and 1 are uniformly distributed.

[5] Note that the BER is not useful in a standard steganographic scenario since no channel error should be considered during the transmission of the stego image. Unfortunately, none of the architecture of existing literature for the *3-player game* are able to embed a message that could be retrieved without errors

[6] In many embedding schemes, the encoding by an error-correcting code is optional.

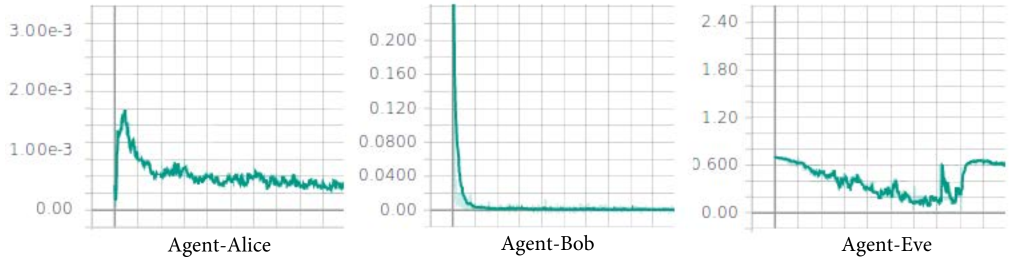

Fig. 8: Generic loss evolution of Agent-Alice, Agent-Bob and Agent-Eve after 300000 iterations.

TABLE 1: First architecture BER and Pe for different payloads.

Payload	Bite Error Rate	Probability of error
0.2	0	3.7%
0.4	0	3.3%
1	0	2.5%

Second architecture:

In the second architecture, we manage to control the noise power introduced by Agent-Alice during the embedding process. To do this, we have conducted several experiments with different values of the change rate β of the Eq. 10. In Fig. 9, we present the Bit Error Rate (BER) and the Probability of error (Pe) of the second architecture, when different β values are used, for a fixed payload of 0.4 bpp. The used steganalyzer was Yedroudj-Net. We can see that, for β equal to 0.4, Eve, the steganalyzer (Yedroudj-net) gets a Probability of error (Pe) close to 6% when Bob makes a 5% Bit Error Rate (BER). When β is set to 0.2, the security of our system improves with a Pe equal to 9%, although the message extraction becomes more difficult for Bob with a 12% Bit Error Rate.

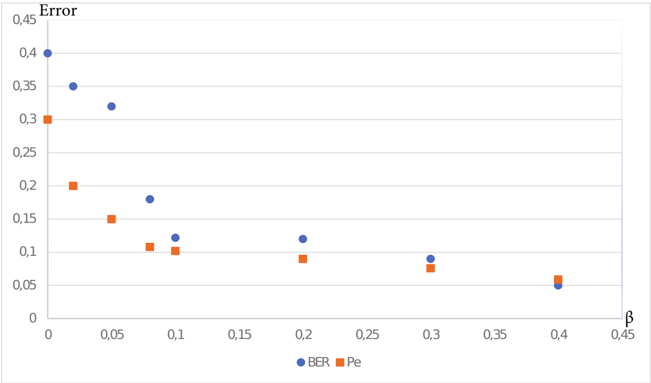

Fig. 9: BER and Pe for the second architecture for a payload size of 0,4 bpp in function of change rate β.

For a β value equal to or less than 0.05, the generated images are more secure. Nevertheless, the BER increases considerably. The BER is 32% and 40%, and the Pe is 15% and 30% for β equals 0,05 and 0 respectively. These BER values are very high and would be difficult to correct with a correcting code without strongly reducing the real payload size. The best β value is those that in the same time, maximize the detection error value (Pe), and minimize the size of the coded message; the message has to be encoded such that a correct extraction is ensured. Since these considerations are a little bit of topic, we propose a rapid analysis by choosing a particular point where there is a sudden variation of BER (i.e. the inflection point). For this architecture and for a payload size 0.4 bpp, the value of β corresponding to a sudden variation of BER is around $\beta = 0.1$. At $\beta = 0.1$, the BER=12%. Using the [7,4,3] Hamming Error Correcting Code (ECC) ensures a correction to most 14% BER, so we would, on average, correct all the errors. With this ECC the *real* payload size is 0.23 bpp, and we measure for this point a probability of Error of 10.2%. It is clear that this architecture provides better results compared to the first one, where for a payload of 0.2 bpp the Pe is 3%. Nevertheless, these results are still not convincing, and as explained before, the problem of the second architecture is the weak relationship between Agent-Alice and Agent-Bob. The third architecture aims to counter this weakness.

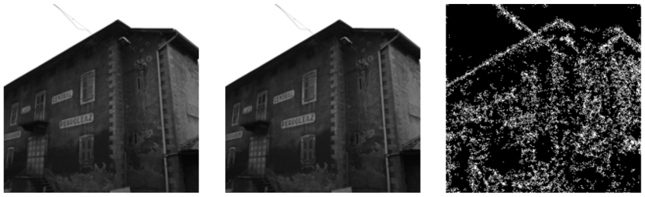

Fig. 10: (Left) The BOSSBase cover image. (Middle) the corresponding stego images with 0.4 bpp and $\beta = 0.1$ using the second architecture. (right) the modification maps between the cover image and the corresponding stego where black=0, white=+/- 1.

In Fig. 10 we can observe that this architecture can learn to concentrate the embedding on textured regions, which are more difficult for a steganalyzer to detect. This architecture has the capacity to learn and find interesting zones to implement steganography.

Third architecture:

As previously mentioned, the third architecture has been proposed to improve the extraction part of the second architecture. However, this requires more time to converge, but the up side is that it offers a better message retrieval accuracy. In Fig. 11, we present the bit error rate (BER) and the probability of error (Pe) of the third architecture as a function of β the change rate, for a fixed payload 0.4 bpp.

Compared to the second architecture, the error of Bob (BER) is smaller regardless of the value of β, even though the detection accuracy remains almost the same for both architectures. The BER obtained with the third architecture

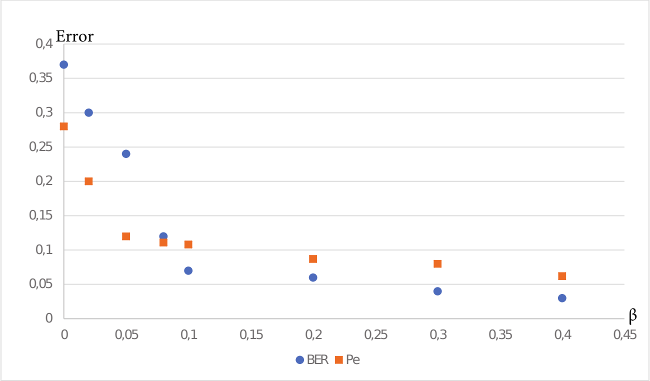

Fig. 11: BER and Pe for the third architecture for a payload size of 0.4 bpp in function of change rate β.

and in comparison with the second architecture is 2% lower when $\beta = 0.4$, 6% lower when we set $\beta = 0.2$, and 5% lower when $\beta = 0.1$. By observing these results, we see that Agent-Alice and Agent-Bob have nevertheless benefited from the third architecture. More joint learning between Agent-Alice and Agent-Bob seems to be the right research direction.

Looking at Fig. 11, similarly to the second architecture, the inflection point is around $\beta = 0.1$. At this point, the BER value is 0.06. The errors can be corrected with a Hamming code [15,11,3].

In this case the *real* payload size is $(0,4 \cdot 11)/15 = 0.293bpp \approx 0.3$ bpp. Using this payload we get a Pe of about 11%.

To provide a comparison, we run a steganalysis with Yedroudj-Net against two steganographic algorithms S-UNIWARD and WOW at a payload size of 0.3 bpp using the database BOWS2. The probability of error (Pe) is 27.3% (resp. 22.4%) for S-UNIWARD (resp. WOW). There is undoubtedly a security gap with the actual embedding schemes, but again, the objective of the paper is to define the *3-player game* concept, and to analysis, its potential compared to other modern embedding schemes and steganalysis scheme. In this, the third architecture shows that there is a real potential, and this paper paves the way to many research possibility.

Fig. 12 shows the modifications zone using the third architecture. We can observe some adaptivity, as the embedding is more dense in textured zones.

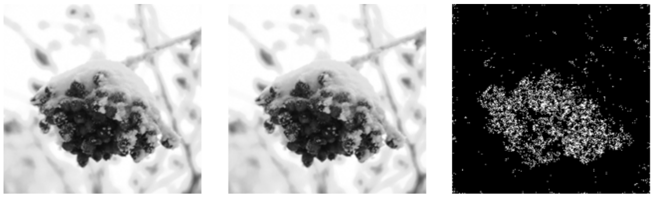

Fig. 12: (Left) The BOSSBase cover image. (Middle) The stego image with payload size 0.4 bpp and $\beta = 0.1$ using the third architecture. (right) the modification maps between the cover image and the corresponding stego where black=0, white=+/- 1.

The results show that this architecture can implement adaptive embedding. It also indicates that the link between Agent-Alice and Agent-Bob should be reinforced in the future to reduce the BER and eventually reduce the change rate and increase the security level. These results should not deter from the 3-player concept, even if the security level at the moment is lower compared to the traditional adaptive embedding algorithms such as S-UNIWARD and WOW. Indeed, the paper shows real improvements compared to GSIVAT and HiDDEeN.

5.3.2 Model's security analysis

To further investigate the security of the proposed model additional tests have been carried out. We list below the points we wanted to investigate with these tests, with a brief description for each of them:

Using a different steganalysis algorithm for the testing phase: In previous experiments, we used the Yedroudj-Net (for playing the role of Agent-Eve) in order to train Agent-Alice and Agent-Bob, but also to evaluate Agent-Alice's security level at test time. To this end, and in order to determine whether the embedding algorithm, Agent-Alice, has a similar security level against other steganalysts, we use CovPool-Net [34], which is a recent state-of-the-art steganalyzer, that was published in July 2019. Thanks to its deep architecture design, this network achieves a good level of performance. The results are presented in Table 2. As one can observe, the error probability obtained with Yedroudj-Net and CovPool-Net on BOWS2, whatever the architecture, are similar. These results show that Agent-Alice is not specifically tuned to defeat Yedroudj-Net (Agent-Eve), otherwise, CovPool-Net would achieve a smaller error probability. Indeed, CovPool-Net is obtaining similar results compared to SRNet [34], and SRNet is slightly better than Yedroudj-Net [29]; meaning that CovPool-Net is supposed to be slightly better than Yedroudj-Net. So, our embedding algorithm (Agent-Alice), seems sufficiently generic and does not show security leaks when steganalyzed with a steganalysis algorithm different from Yedroudj-Net.

TABLE 2: Steganalysis error probability obtained with Yedroudj-Net and CovPool-Net on BOWS2 base. For data embedding, the second and third architectures are used at payload 0.4 bpp.

	Yedroudj-Net	CovPool-Net
Third architecture / real payload 0.3bpp	10.8 %	11.3 %
Second architecture / real payload 0.23bpp	10.2 %	10.2 %

Using another database to evaluate the security level: The purpose of this test is to study the influence of the database on the security level of the embedding algorithm, Agent-Alice. That is to say, during the test phase, instead of using a database (BOWS2 base) that is statistically similar to the one used during the training phase (BOSS base), we used a database whose statistics are different. For this, we used the ALASKA-10K base [35]. Please note that this database is more secure than BOSS and BOWS2 i.e. more difficult to steganalyze.

Table 3 shows that for an embedding with the same modification rate, ALASKA-10K is a more secure database

TABLE 3: The BER and steganalysis error probability of Yedroudj-Net on BOWS2 and ALASKA-10K bases, with an embedding with Agent-Alice obtained using the third architecture on BOSSBase with a payload 0.4 bpp.

	Probability of error	Bite Error Rat
BOWS2	10.8 %	6 %
ALASKA-10K	12.1 %	13 %

than BOWS2, which is consistent with previous studies [35]. Indeed, using this database results in a 1% increase in Pe compared to BOWS2. However, this database produces more decoding errors. Indeed, the BER is 13% for Alaska-10K versus 6% for BOWS2. Thus, the correction ability of a Hamming code [15,11,3] is no more sufficient. We can, for example, use a Hamming code [7,4,3]. In thus case, the real payload is reduced to 0.23 bpp on ALASKA-10K, compared to 0.3bpp on BOWS2.

Testing different payloads: The objective of this test is to analyze the efficiency of Agent-Alice, when the embedding payload size is not the "designed payload size". Indeed, Agent-Alice is trained at a given payload, and it is conceptually designed to be capable of supporting embedding at different payload sizes during its use in the test phase. So, the objective of this test is to study if Agent-Alice's embedding algorithm obtains relatively good results when embedding at a higher or lower payload size. To do this, we trained Agent-Alice on a payload of 0.4 bpp, and then, thanks to the scalability property of our model, we generated stego images with different payload sizes (0.14 bpp, 1bpp).

TABLE 4: Steganalysis error probability of Yedroudj-Net on BOWS2 base. For data embedding the third architecture is used at different payloads.

Payload size (bpp)	Real payload size (bpp)	Pe
0.14	0.1	11.1%
0.4	0.3	10.8%
1	0.8	NaN

Table 4 illustrates that our method achieves the same level of security for both payloads (0.14 and 0.4 bpp). However, when the payload is equal to 1, Agent-Bob is no longer able to extract the message. This last result suggests that the quantity of modification is insufficient. So, the change rate β (Eq. 10) has to be modified, and the learning of Agent-Alice has to be re-done for a higher payload. When embedding with a lower payload, the network can successfully embed and extract the message, but the security level remains the same as the one obtained for the payload size used during the training phase. For better security performances, the change rate β (Eq. 10) should be changed, and at worst, we should relaunch training with the target payload size. Another solution, in order to ensure "payload scalability" could be "transfer learning", or to run learning with variable payload sizes. This is postponed to future studies.

Embedding with different secret keys: A major weakness of previous models lies in the use of a single secret key for data embedding, even though it is highly discouraged to do so in steganography, as leads to very high detectability [19]. To avoid such a limitation, we propose a dedicated solution that integrates the use of a different key for each data embedding process (stego generation).

To prove that our model handles the secret key in the way it is intended to, we run the embedding process twice by using the same image and the same message, but with two different secret keys.

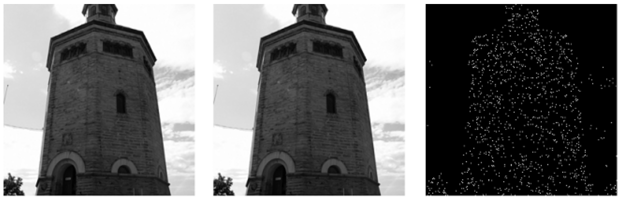

Fig. 13: (Left) The stego image generated with secret key 1 . (Middle) The stego images generated with secret key 2. (right) the difference between the two stego images where black=0, white=+/- 1.

Fig. 13 demonstrates that the model was able to generate a completely new stego image by only changing the secret key. This proves that the proposed method makes correct use of the shared secret key.

6 Conclusion and perspectives

In this paper, we first recalled the four different GAN families used in steganography. Then we presented the *3-player game approach*, and defined the general concept and how to correctly use it for steganography. Three architectures based on the *3-player game approach* have been proposed. The first architecture fixed the flaws made in GSIVAT and HiDDEeN. Nevertheless, this first architecture behaves similarly to GSIVAT and HiDDEeN and is not adapted for steganography purposes due to its extreme detectability. With the second architecture, we suggested a new way to embed a message in the cover image. Instead of directly modifying the cover image, which implies a significant noise power signal, we proposed to generate a modification map with values belonging to {-1,0,1}. The stego is then generated by adding the modification map to the cover. This architecture is much more secure than the first one, but it generates more errors during message extraction. Finally, the third architecture imposes a more joint learning approach between Agent-Alice and Agent-Bob in order to reduce the errors during message extraction. This third architecture takes more time to converge, but achieves better results.

The third architecture, with a *real* payload size of 0,3 bpp, can perfectly retrieve an embedded message (with the help of error-correcting code), for a security level Pe = 10,8%. Visually speaking, see Fig. [10,12] demonstrate that the proposed model was able to learn how to focus the embedding on textured regions, which are known to be more difficult to detect by a steganalyst. Thus, a better level of security is achieved.

Even if the obtained results do not surpass current state-of-the-art embedding algorithms, these experimental results verify the promises of such a method. The major contribution of this paper is really to propose a formalization of the *3-player game* concept, and an end-to-end method using neural networks that can learn to simulate algorithms using human-based rules (steganography and steganalysis).

We expect this work to lead to fruitful avenues for further research. In future work, we can study the possibility of more synchronizing between Agent-Alice and Agent-Bob, and therefor improving general performances. We can also try using a more subtle loss function. This can help the networks to converge to a better solution. Furthermore, finding a theoretical way to compute the change rate β can help to accelerate the learning process. The values of λ_A, λ_B, λ_E could also be more extensively studied.

Acknowledgments

The authors would like to thank the University of Montpellier (LIRMM) and the University of Nîmes. We extend our gratitude to the French Direction Générale de l'Armement (DGA) for its support through the Alaska project ANR (ANR-18-ASTR-0009). We would also like to thank the Algerian Ministry of Higher Education / Scientific Research, for its scholarship support.

References

[1] J. Hayes and G. Danezis, "Generating Steganographic Images Via Adversarial Training," in *Proceedings ofAdvances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NIPS'2017*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Dec. 2017, pp. 1951–1960.

[2] J. Zhu, R. Kaplan, J. Johnson, and L. Fei-Fei, "HiDDeN: Hiding Data With Deep Networks," in *Proceedings of the 15th European Conference on Computer Vision, ECCV'2018*, ser. Lecture Notes in Computer Science, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11219. Springer, Sep. 2018, pp. 682–697.

[3] G. J. Simmons, "The Subliminal Channel and Digital Dignatures," in *Proceeding of Crypto'83*, E. by D. Chaum, Ed. New York, Plenum Press, Aug. 1983, pp. 51–67.

[4] A. D. Ker, P. Bas, R. Böhme, R. Cogranne, S. Craver, T. Filler, J. Fridrich, and T. Pevný, "Moving Steganography and Steganalysis from the Laboratory into the Real World," in *Proceedings of the 1st ACM Workshop on Information Hiding and Multimedia Security, IH&MMSec'2013*. Montpellier, France: ACM, Jun. 2013, pp. 45–58.

[5] A. Kerckhoffs, "La Cryptographie Militaire," *Journal des Sciences Militaires*, vol. IX, pp. 5-38 Jan. 1883, pp. 161-191, Feb. 1883.

[6] V. Holub, J. Fridrich, and T. Denemark, "Universal Distortion Function for Steganography in an Arbitrary Domain," *EURASIP Journal on Information Security, JIS*, vol. 2014, no. 1, Jan. 2014.

[7] V. Holub and J. Fridrich, "Designing Steganographic Distortion Using Directional Filters," in *Proceedings of the IEEE International Workshop on Information Forensics and Security, WIFS'2012*, Tenerife, Spain, Dec. 2012, pp. 234–239.

[8] V. Sedighi, R. Cogranne, and J. Fridrich, "Content-Adaptive Steganography by Minimizing Statistical Detectability," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 2, pp. 221–234, Feb 2016.

[9] P. Schöttle and R. Böhme, "A Game-theoretic Approach to Content-adaptive Steganography," in *Proceedings of the 14th International Conference on Information Hiding*, ser. IH'12, M. Kirchner and D. Ghosal, Eds. Springer Berlin Heidelberg, 2012, pp. 125–141.

[10] ——, "Game Theory and Adaptive Steganography," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 4, pp. 760–773, April 2016.

[11] M. Chaumont, "Deep Learning in steganography and steganalysis," in *Digital Media Steganography: Principles, Algorithms, Advances*, M. Hassaballah, Ed. Elsevier, Jul. 2020, ch. 14, pp. 321–349. [Online]. Available: http://arxiv.org/abs/1904.01444

[12] J. Kodovsky, J. Fridrich, and V. Holub, "On Dangers of Overtraining Steganography to Incomplete Cover Model," in *Proceedings of the Thirteenth ACM Multimedia Workshop on Multimedia and Security*, ser. MM&MMSec'2011. New York, NY, USA: ACM, 2011, pp. 69–76.

[13] S. Kouider, M. Chaumont, and W. Puech, "Adaptive Steganography by Oracle (ASO)," in *Proceedings of the IEEE International Conference on Multimedia and Expo, ICME'2013*, July 2013, pp. 1–6.

[14] M. Abadi and D. G. Andersen, "Learning to Protect Communications with Adversarial Neural Cryptography," in *ArXiv; Rejected from the 5th International Conference on Learning Representations, ICLR'2017.*, vol. abs/1610.06918, 2016. [Online]. Available: http://arxiv.org/abs/1610.06918

[15] D. Hu, L. Wang, W. Jiang, S. Zheng, and B. Li, "A Novel Image Steganography Method via Deep Convolutional Generative Adversarial Networks," *IEEE Access*, vol. 6, pp. 38 303–38 314, 2018.

[16] W. Tang, S. Tan, B. Li, and J. Huang, "Automatic Steganographic Distortion Learning Using a Generative Adversarial Network," *IEEE Signal Processing Letters*, vol. 24, no. 10, pp. 1547–1551, Oct 2017.

[17] W. Tang, B. Li, S. Tan, M. Barni, and J. Huang, "CNN-based Adversarial Embedding for Image Steganography," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2019.

[18] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *Proceedings of ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[19] L. Pibre, J. Pasquet, D. Ienco, and M. Chaumont, "Deep Learning is a Good Steganalysis Tool When Embedding Key is Reused for Different Images, Even if There is a Cover Source-Mismatch," in *Proceedings of Media Watermarking, Security, and Forensics, MWSF'2016, Part of I&ST International Symposium on Electronic Imaging, EI'2016*, San Francisco, California, USA, Feb. 2016, pp. 1–11.

[20] J. Fridrich, *Steganography in Digital Media*. Cambridge University Press, 2009, cambridge Books Online.

[21] T. Taburet, P. Bas, J. Fridrich, and W. Sawaya, "Computing Dependencies between DCT Coefficients for Natural Steganography in JPEG Domain," in *Proceedings of the 7th ACM Workshop on Information Hiding and Multimedia Security, IH&MMSec'2019*. Paris, France: ACM, Jul. 2019, pp. 57–62.

[22] D. Lerch-Hostalot and D. Megìas, "Unsupervised Steganalysis Based on Artificial Training Sets," *Engineering Applications of Artificial Intelligence*, vol. 50, pp. 45 – 59, 2016.

[23] J. Kodovský, J. Fridrich, and V. Holub, "Ensemble Classifiers for Steganalysis of Digital Media," *IEEE Transactions on Information Forensics and Security, TIFS*, vol. 7, no. 2, pp. 432–444, 2012.

[24] J. Fridrich and J. Kodovský, "Rich Models for Steganalysis of Digital Images," *IEEE Transactions on Information Forensics and Security, TIFS*, vol. 7, no. 3, pp. 868–882, June 2012.

[25] M. Yedroudj, F. Comby, and M. Chaumont, "Yedroudj-Net: An Efficient CNN for Spatial Steganalysis," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'2018, Calgary, AB, Canada, April 15-20, 2018*, 2018, pp. 2092–2096.

[26] B. Li, W. Wei, A. Ferreira, and S. Tan, "ReST-Net: Diverse Activation Modules and Parallel Subnets-Based CNN for Spatial Image Steganalysis," *IEEE Signal Processing Letters*, vol. 25, no. 5, pp. 650–654, May 2018.

[27] M. Boroumand, M. Chen, and J. Fridrich, "Deep Residual Network for Steganalysis of Digital Images," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2018.

[28] M. Yedroudj, M. Chaumont, and F. Comby, "How to Augment a Small Learning Set for Improving the Performances of a CNN-based Steganalyzer?" in *Proceedings of Media Watermarking, Security, and Forensics, MWSF'2018, Part of I&ST International Symposium on Electronic Imaging, EI'2018*, Jan. 2018.

[29] R. Zhang, F. Zhu, J. Liu, and G. Liu, "Depth-Wise Separable Convolutions and Multi-Level Pooling for an Efficient Spatial CNN-based Steganalysis (previously named "efficient feature learning and multi-size image steganalysis based on cnn" on ArXiv)," *IEEE Transactions on Information Forensics and Security, TIFS*, vol. 15, pp. 1138–1150, 2020.

[30] T. Filler, J. Judas, and J. Fridrich, "Minimizing Embedding Impact in Steganography Using Trellis-Coded Quantization," in *Proceedings of IS&T/SPIE Annual Symposium on Electronic Imaging, Security, Steganography, and Watermarking of Multimedia Contents edited by*

Edward J. Delp III, Ping Wah Wong, SPIE'2010, vol. 7541, Jan. 2010, pp. 1 – 14.

[31] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention, MICCAI'2015*, ser. LNCS, vol. 9351. Springer, 2015, pp. 234–241.

[32] P. Bas, T. Filler, and T. Pevný, "'Break Our Steganographic System': The Ins and Outs of Organizing BOSS," in *Proceedings of the 13th International Conference on Information Hiding, IH'2011*, ser. Lecture Notes in Computer Science, vol. 6958. Prague, Czech Republic: Springer, May 2011, pp. 59–70.

[33] P. Bas and T. Furon, "BOWS-2 Contest (Break Our Watermarking System)," organised within the activity of the Watermarking Virtual Laboratory (Wavila) of the European Network of Excellence ECRYPT, 2008, organized between the 17th of July 2007 and the 17th of April 2008. http://bows2.ec-lille.fr/.

[34] X. Deng, B. Chen, W. Luo, and D. Luo, "Fast and Effective Global Covariance Pooling Network for Image Steganalysis," in *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, ser. IH&MMSec'2019, Paris, France, Jul. 2019, pp. 230–234.

[35] M. Yedroudj, M. Chaumont, F. Comby, A. Oulad-Amara, and P. Bas, "Pixels-off: Data-augmentation Complemen-tary Solution for Deep-learning Steganalysis," in *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, ser. IH&MMSec'2020, PNew York, NY, USA, Jul. 2020.