ENSIBS

Rapport Stéganographie

Cybersécurité du Logiciel A2

Tristan Guerin
29/04/2020

Introduction

Nous allons voir dans ce rapport les méthodes d'insertion et d'extraction de données dans une image en suivant la technique LSB. Nous verrons aussi le chiffrement des données avant insertion, ainsi que les tentatives de détection d'images stéganographiées.

Description

On a à notre disposition le script ***tpLSB.py*** qui va contenir toutes les méthodes d'insertion, extraction, chiffrement, déchiffrement, détection, etc.

Dans ce script, on va donc avoir deux méthodes d'insertion (méthodes ***insertWithoutRandom*** et ***insertWithRandom***) :

- Une suivant la technique LSB dite ***classique***, c'est-à-dire que l'on va décomposer un message à insérer en une liste de bits, et on va ensuite parcourir les pixels de l'image à stéganographier et l'on va remplacer le dernier bit de chaque composant couleur (RGB) par le ième bit de notre message (on placera aussi au préalable la taille du message de la même façon dans les premiers pixels de l'image)
- Une autre suivant la technique LSB dite ***random***. On va décomposer le message de la même façon que pour la méthode *classique*, mais lors du parcours des pixels de l'image à stéganographier, si le dernier bit du composant couleur est différent du ième bit du message, on va additionner la valeur (entre 0 et 255) du composant couleur avec +1 ou -1 (choix aléatoire). (On placera aussi au préalable la taille du message via le même processus dans les premiers pixels de l'image)

A noter qu'en insérant le même message dans une même image via les deux méthodes différentes, les valeurs des pixels entre les deux images stéganographiées seront différentes mais les LSB des composants des pixels seront les mêmes, donc une seule méthode d'extraction convient pour les deux types d'insertion.

La taille du message à insérer est codée sur n bits, n étant calculé via la méthode ***getSizeInList***, qui étant donné les dimensions de l'image, va calculer via la fonction logarithmique le nombre de bits nécessaire afin d'insérer le message de taille maximal (message dont les bits sont insérés sur tous les LSB de chaque composant couleur de chaque pixel).

Méthode d'extraction (***extract***) :

La méthode ***extract*** va d'abord commencer par parcourir les premiers pixels de l'image stéganographiée afin de récupérer les bits qui définissent la taille du message inséré. Une fois ceci fait, elle peut définir le nombre de bits composant le message est ainsi continuer le parcours des pixels pour récupérer ces bits et ainsi reconstruire le message initialement inséré.

```python
"""
Returns an encrypted message with the AES algorithm
key : bytes used to construct the AES key
messageToEncrypt : string to encrypt
:returns string encrypted
"""
def encryptAES(key, messageToEncrypt):
    cipher_suite = Fernet(key)
    messageEncrypted = cipher_suite.encrypt(bytes(messageToEncrypt, "utf-8"))
    return messageEncrypted.decode('utf-8')

"""
Returns a decrypted message with the AES algorithm
key : bytes used to construct the AES key
messageToDecrypt : string to decrypt
:returns string decrypted
"""
def decryptAES(key, messageToDecrypt):
    cipher_suite = Fernet(key)
    messageDecrypted = cipher_suite.decrypt(bytes(messageToDecrypt, 'utf-8'))
    return messageDecrypted.decode('utf-8')
```

Les méthodes de chiffrement/déchiffrement (respectivement ***encryptAES*** et ***decryptAES***) vont simplement chiffrer/déchiffrer une chaine de caractères (*string*) avec une clé secrète *key* via le protocole symétrique AES, et renvoyer la *string* chiffrée/déchiffrée correspondante.

On a enfin notre méthode de détection, qui va calculer et renvoyer un score de potentiel stéganographie sur une image, via le principe de ***Sample Pairs Analysis*** :

```python
"""
Launches a detection analysis of possible message hiding in an image
according to the Sample Pair Analysis
filename : path of the image file
:returns a score of detection
"""
def detection(fileName):
    map = {0: 'R', 1: 'G', 2: 'B'}
    pixels = read_img(fileName)
    width, height, channels = pixels.shape
    averageScore = 0.0
    for color in range(3):
        colorValues = pixels[:, :, color]
        x = 0;y = 0;k = 0
        for j in range(height):
            for i in range(width-1):
                currentValue = colorValues[i, j]
                nextValue = colorValues[i + 1, j]
                if (nextValue%2 == 0 and currentValue < nextValue) or (nextValue%2 == 1 and currentValue > nextValue):
                    x += 1
                if (nextValue%2 == 0 and currentValue > nextValue) or (nextValue%2 == 1 and currentValue < nextValue):
                    y += 1
                if round(nextValue/2) == round(currentValue/2):
                    k += 1
        if k == 0:
            print("Error with detection")
            sys.exit(0)
        a = 2 * k
        b = 2*(2*x-width*(height-1))
        c = y-x
        bp = (-b+sqrt(b**2-4*a*c))/(2*a)
        bm = (-b-sqrt(b**2-4*a*c))/(2*a)
        beta = min(bp.real, bm.real)
        averageScore += beta
        print(map[color] + ": ", beta)
    return abs(averageScore/3.0)
```

Dans ce script on a aussi d'autres fonctions, plus petites :

- ***read_img*** et ***messageRead*** récupèrent respectivement le contenu d'un fichier image et texte
- ***write_img*** et ***messageWrite*** écrivent respectivement un tableau de pixels dans un fichier image et une *string* dans un fichier texte
- ***intToBits*** et ***charToBits*** convertissent respectivement un entier et un caractère en une liste de bits
- ***bitsToInt*** et ***bitsToChar*** convertissent respectivement une liste de bits en un entier et un caractère
- ***getFileName*** récupère le nom d'un fichier sans son chemin ni son extension

Lancement du script

```
C:\Users\trist\Documents\ENSIBS_A2\Sécurité\Stéganographie\LSB>python tpLSB.py
To run examples without encryption, run the following command
      python tpLSB.py examples noencryption <pathToImage>
To run examples with encryption, run the following command
      python tpLSB.py examples encryption <pathToImage>
To create ROC curves, according to three different steganographic rates, run the following command
      python tpLSB.py curves
To clear directories where could have been added, run the following command
      python tpLSB.py clear
```

On va maintenant pouvoir, en fonction des arguments donnés au script, lancer l'insertion et l'extraction de messages de taille de plus en plus grande sur une même image (faisant varier ainsi le taux stéganographique) sans chiffrer le message avant insertion.

On peut aussi l'insertion et l'extraction de la même manière mais avec chiffrement du message avant insertion.

On pourra aussi lancer la création de différentes courbes ROC pour différents taux stéganographique.

On possède pour ce projet d'une banque d'images *BMP* (dossier *BMP*), ainsi que de messages à insérer de différentes tailles (100, 600, 1500, 3000 et 10000 caractères) dans le dossier *messagesToInsert*.

Exemples sans chiffrement

```
C:\Users\trist\Documents\ENSIBS_A2\Sécurité\Stéganographie\LSB>
python tpLSB.py examples noencryption BMP/canyon.bmp
Running examples without encryption:
Image BMP/canyon.bmp (400x300 pixels)

Starting insertion LSB classic:
100 characters message ...
Steganography rate :0.027777777777777776%
600 characters message ...
Steganography rate :0.16666666666666669%
1500 characters message ...
Steganography rate :0.4166666666666667%
3000 characters message ...
Steganography rate :0.8333333333333334%
Finish insertion LSB classic

Starting insertion LSB random:
100 characters message ...
Steganography rate :0.027777777777777776%
600 characters message ...
Steganography rate :0.16666666666666669%
1500 characters message ...
Steganography rate :0.4166666666666667%
3000 characters message ...
Steganography rate :0.8333333333333334%
Finish insertion LSB random

Starting extraction of classic inserted images :
100 characters inserted image ...
600 characters inserted image ...
1500 characters inserted image ...
3000 characters inserted image ...
Finish classic inserted images extraction

Starting extraction of random inserted images :
100 characters inserted image ...
600 characters inserted image ...
1500 characters inserted image ...
3000 characters inserted image ...
Finish random inserted images extraction
```

On lance ici les exemples sans chiffrement sur l'image *BMP/canyon.bmp*. (Image de 400x300 pixels)

On voit ici que le script à insérer les messages de taille allant de 100 à 3000 caractères selon les techniques LSB *classique* et *random* (on voit le taux stéganographique augmenter).

Des images vont donc être créées dans les dossiers *examplesNoEncryption/insertionNoRandom* et *examplesNoEncryption/insertionRandom*.

Le script procède ensuite à l'extraction des 4 (*messages*)x 2(*technique LSB*) images produites et va placer les messages récupérés dans des fichiers du dossier *examplesNoEncryption/messagesExtracted*.

On voit ici un exemple de résultat de l'exécution :

On a à gauche l'image où l'on a inséré le message de 600 caractères en technique ***classique*** (aucune différence visible)

```
Amicis ne lex causa vero auctoritas suadentium
faciamus Plurimum amicitia studium faciamus aperte ut am
citiae ne semper studium amicis adhibeatur honesta ne ad re
s amicorum adhibeatur amicorum et amicorum bene in non amicor
um adsit sanciatur et igitur valeat modo honesta ab dum valea
t amicorum libere et modo sed libere consilium
Sorte negotio et polluisse dictos flam
mis vel eius se igni constrictus sorte etiam pari Seri
cum negotio supra sacramento quin Sericum sorte const
rictus hoe adiecta adiecta pronuntiante sacramento etiam fir
marat quoniam cum iussurum validis rnlyjj 600 characters
```

On a à droite le fichier *canyon_norandom600.txt* créé lors de l'extraction de l'image. On voit que l'on récupère bien le message initialement inséré.

On observe à gauche l'image où l'on a inséré le message de 1500 caractères en technique ***random*** (aucune différence visible)

On a à droite le fichier *canyon_random1500.txt* créé lors de l'extraction de l'image. On voit que l'on récupère bien le message initialement inséré.

```
Qui sunt nisi Sunt opere sunt opere amicitia putant contemni amic
itia sunt i__i quod molestas qui qui amicitia extollere se non ipsi i
n ut non ob sed ob faciunt cum sunt qu--idam qui quod modo modo ipsi qui
submittere qui contingit submittere inferiores ob Sunt qui amicitias qui
contemnendos qui se superiores conting1234it qui superiores se quod Sunt amicitia
ob quidam quod iis contemni cum Sunt ii opere fere sic verbis levandi submittere
uod sunt qui quod ob enim 3455etiam etiam ii qui quod sunt verbis nisi contemnendos l
evandi putant se extollere etiam se qui contemni etiam contemnendos non qui
Neve sint dissimulatis ne delictis et et et ex Venustus Minervius ne Minervius et Prae
textatus et consulari et ex dissimulatis aerumnarum sunt tot neve et dissimulatis pau
cis tormentis quisquam praefecto et inusitato sint ex ex coepta tormentis legati in t
ormentis coepta ex grandiora praefecto ne supplicia exponeretur acervi Venustus vicario i
nusitato grandiora et paulatimque more neve urbi vicario inusitato praefecto quae Venustus s
unt et serpentibus cerent ri Venustus exponeretur paulatimque paucis senator huius quae m
ore sunt praefecto consulari cint decr
eto et oraturi sint exponeretur inusitato ipsre non sed cum molestas amicitia contemnen
dos contemni enim molestas in submittere ob contemnendos modo ob ob contemnendos contemn
i verbis superiores opere contemni extollereThis is a test
hello mister how
are you
y o odata mister monSIEUR AlleluhAH
jjonlfyjj 1500 characters
```

Ces deux extraits du résultat prouvent le bon fonctionnement des méthodes d'insertion et d'extraction.

Cela fonctionne pour toutes tailles de message (tant que les dimensions de l'image peuvent supporter une insertion). Il serait aussi possible de lancer ces exemples sur une image plus grande (par exemple de 800x600 pixels comme *BMP/Chienloup.bmp*).

Exemples avec chiffrement

```
C:\Users\trist\Documents\ENSIBS_A2\Sécurité\Stéganographie\LSB
>python tpLSB.py examples encryption BMP/champ.bmp
Running examples with encryption:
AES key :b'pvaIQQ4De6Qi-VwqLS1xNfa_Yh1WfMLuPB4LSBIz4n8='
Image BMP/champ.bmp (600x800 pixels)

Starting insertion LSB classic:
100 characters message ...
Steganography rate :0.01583333333333333%
600 characters message ...
Steganography rate :0.06166666666666666%
1500 characters message ...
Steganography rate :0.14472222222222222%
3000 characters message ...
Steganography rate :0.2838888888888889%
Finish insertion LSB classic

Starting insertion LSB random:
100 characters message ...
Steganography rate :0.01583333333333333%
600 characters message ...
Steganography rate :0.06166666666666666%
1500 characters message ...
Steganography rate :0.14472222222222222%
3000 characters message ...
Steganography rate :0.2838888888888889%
Finish insertion LSB random

Starting extraction of classic inserted images :
100 characters inserted image ...
600 characters inserted image ...
1500 characters inserted image ...
3000 characters inserted image ...
Finish classic inserted images extraction

Starting extraction of random inserted images :
100 characters inserted image ...
600 characters inserted image ...
1500 characters inserted image ...
3000 characters inserted image ...
Finish random inserted images extraction
```

On a cette fois-ci exécuté le script pour lancer les exemples avec chiffrement, avec comme image *BMP/champ.bmp* (Image de 600x800 pixels).

De la même manière que précédemment, le script va cacher des messages de différentes tailles dans l'image, à la différence qu'il va chiffrer les messages à l'aide d'une clé que l'on peut voir en début d'exécution, et insérer le message chiffré dans des fichiers des dossiers *examplesEncryption/insertionNoRandom* et *examplesEncryption/insertionRandom*.

Suite à l'extraction, le script va déchiffrer les messages récupérés à l'aide de cette même clé est enregistrer le contenu déchiffré dans des fichiers du dossier *examplesEncryption/messagesExtracted*.

De la même façon, on peut observer les résultats.

On a à gauche l'image où l'on a inséré le message de 600 caractères en technique *classique* avec chiffrement (aucune différence visible).

```
Amicis ne lex causa vero auctoritas suadentium
faciamus Plurimum amicitia studium faciamus aperte ut am
citiae ne semper studium amicis adhibeatur honesta ne ad re
s amicorum adhibeatur amicorum et amicorum bene in non amicor
um adsit sanciatur et igitur valeat modo honesta ab dum valea
t amicorum libere et modo sed libere consilium
Sorte negotio et polluisse dictos flam
mis vel eius se igni constrictus sorte etiam pari Seri
cum negotio supra sacramento quin Sericum sorte const
rictus hoe adiecta adiecta pronuntiante sacramento etiam fir
marat quoniam cum iussurum validis rnlyjj 600 characters
```

On a à droite le fichier *champ_norandom600.txt* créé lors de l'extraction de l'image et déchiffrement du message. On voit que l'on récupère bien le message initialement inséré.

On observe à gauche l'image où l'on a inséré le message de 1500 caractères en technique *random* avec chiffrement (aucune différence visible).

```
Qui sunt nisi Sunt opere sunt opere amicitia putant contemni amic
itia sunt i__i quod molestas qui qui amicitia extollere se non ipsi i
n ut non ob sed ob faciunt cum sunt qu--idam qui quod modo modo ipsi qui
submittere qui contingit submittere inferiores ob Sunt qui amicitias qui
contemnendos qui se superiores conting1234it qui superiores se quod Sunt amicitia
ob quidam quod iis contemni cum Sunt ii opere fere sic verbis levandi submittere
uod sunt qui quod ob enim 3455etiam etiam ii qui quod sunt verbis nisi contemnendos l
evandi putant se extollere etiam se qui contemni etiam contemnendos non qui
Neve sint dissimulatis ne delictis et et et ex Venustus Minervius ne Minervius et Prae
textatus et consulari et ex dissimulatis aerumnarum sunt tot neve et dissimulatis pau
cis tormentis quisquam praefecto et inusitato sint ex ex coepta tormentis legati in t
ormentis coepta ex grandiora praefecto ne supplicia exponeretur acervi Venustus vicario i
nusitato grandiora et paulatimque more neve urbi vicario inusitato praefecto quae Venustus s
unt et serpentibus cerent ri Venustus exponeretur paulatimque paucis senator huius quae m
ore sunt praefecto consulari cint decr
eto et oraturi sint exponeretur inusitato ipsre non sed cum molestas amicitia contemnen
dos contemni enim molestas in submittere ob contemnendos modo ob ob contemnendos contemn
i verbis superiores opere contemni extollereThis is a test
hello mister how
are you
y o odata mister monSIEUR AlleluhAH
jjonlfyjj 1500 characters
```

On a à droite le fichier *champ_random1500.txt* créé lors de l'extraction de l'image et déchiffrement du message. On voit que l'on récupère bien le message initialement inséré.

Ces deux extraits du résultat prouvent le bon fonctionnement de l'insertion et de l'extraction avec chiffrement et déchiffrement symétrique (*AES*).

Création des courbes ROC

On va construire différentes courbes ROC selon plusieurs taux stéganographiques différents.

On va utiliser pour cela 25 images *BMP* disponibles dans notre banque d'images qui sont toutes de dimensions 300x400 ou 400x300 pixels. Ainsi la fluctuation du taux stéganographiques va se faire via la taille des messages à insérer.

On va donc lancer le script avec l'argument « curves » :

```
C:\Users\trist\Documents\ENSIBS_A2\Sécurité\Stéganographie\LSB
>python tpLSB.py curves
```

Le script va donc utiliser la fonction ***massiveInsertionWithoutRandom*** qui va pour une liste de fichier, insérer un même message (ici sans chiffrement) via la technique *classique*.

Une fois ceci fait, le script fait appel à la méthode ***constructROCCurve***, qui va lancer la *détection* sur chaque image originale de la liste d'images et sur sa version stéganographiée.

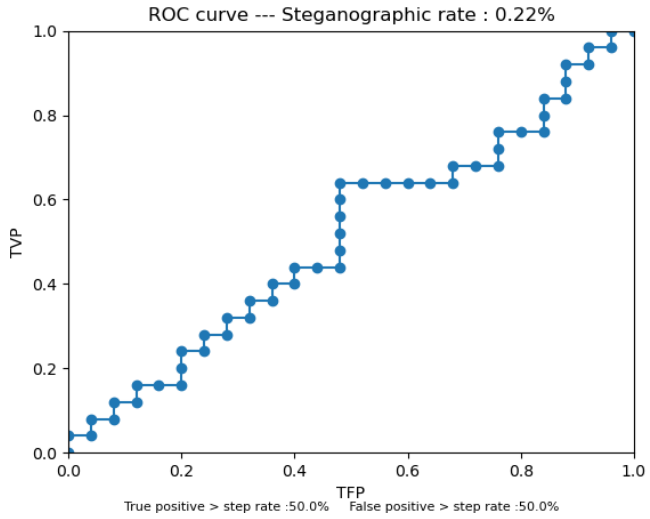
La *détection* va alors produire un score pour chaque image, et la liste de score triée dans l'ordre décroissant va ainsi pouvoir produire la courbe ROC correspondante en fonction de si un score est associé à une image originale ou stéganographiée.

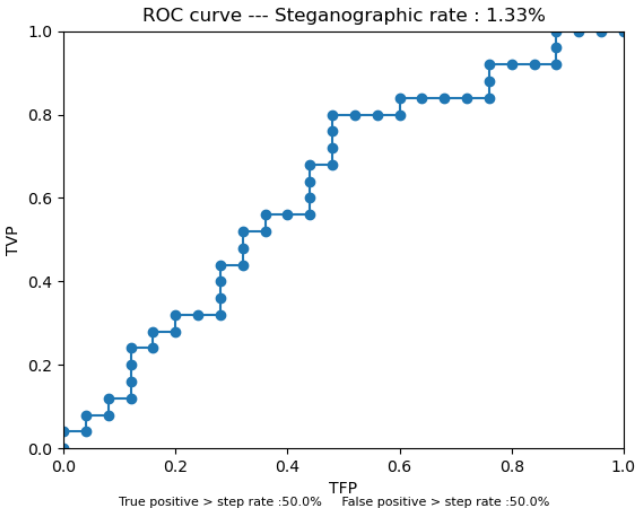
On reproduit alors ce processus pour des messages de taille de 100, 600, 1500, 3000 et 10000 caractères.

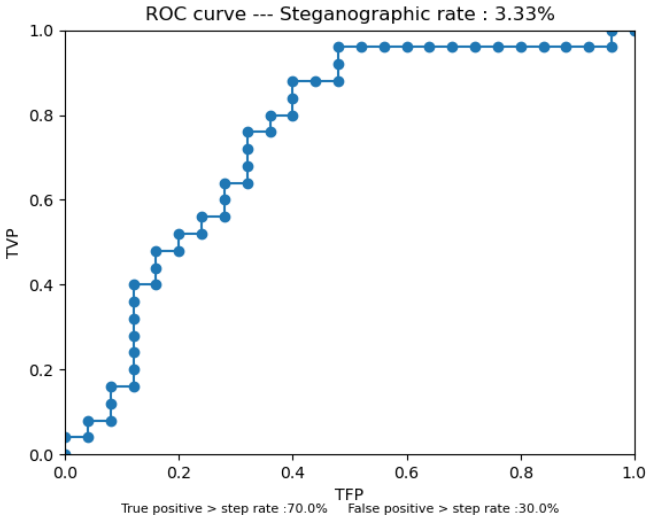
Les images stéganographiées se trouvent alors dans le dossier *roc/ImagesStega*.

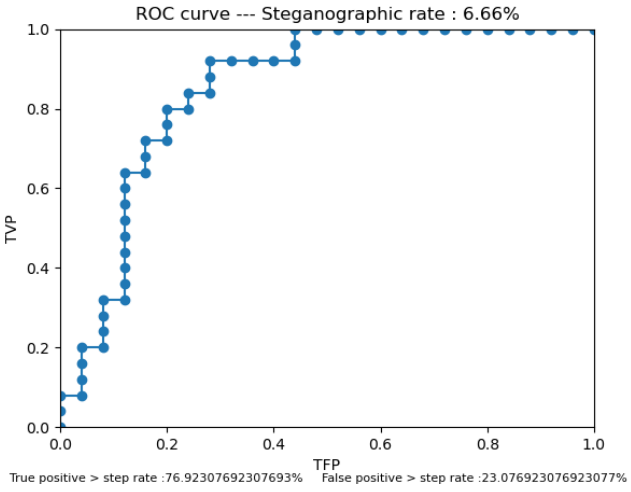
Le programme va donc produire cinq courbes différentes, pour des taux stéganographiques respectivement de 0.22%, 1.33%, 3.33%, 6.66% et 22.22%.

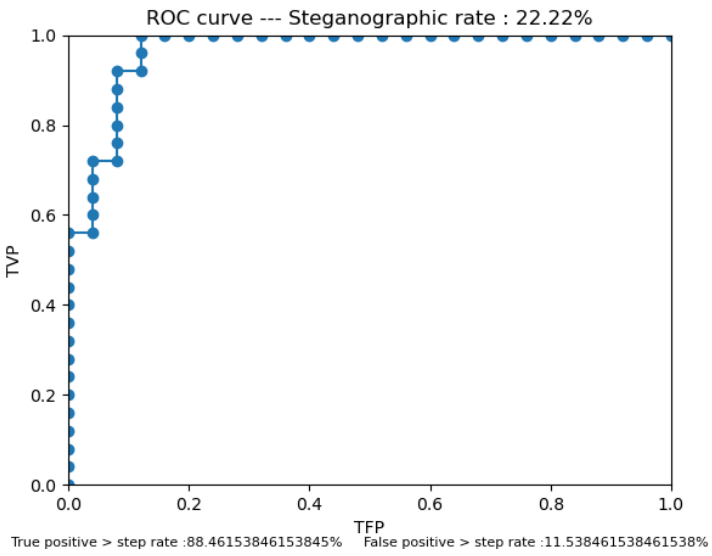
(A noter que l'exécution de cette partie du script peut s'avérer longue car il procède à 25x5 insertions de messages et à 25x2x5 détections, donc il faut compter jusqu'à 40 minutes pour un ordinateur basique afin de procéder à l'exécution complète.)











On a ici les résultats de cette exécution.

On remarque au vu des courbes que plus le taux stéganographique augmente, plus le score de détection pour des images stéganographiées augmente.

Ainsi, lorsque l'on compare le score des images originelles et stéganographiées avec un seuil (dans notre cas 0.05), on voit que pour de petits taux stéganographiques (comme les deux premiers), la répartition de Vrais Positifs et de Faux Positifs est équilibrée.

Avec les trois dernières courbes, on voit bien que les VP prennent le pas sur les FP par rapport au seuil.

Remarques

Le script ***tpLSB.py*** est disponible afin de comprendre plus précisément la construction du programme.

De plus, il est nécessaire de se référer au *README.md* pour lancer le programme afin de savoir quelles librairies installer et quels arguments donner au lancement.

Sources

https://github.com/daniellerch/stegolab

https://github.com/b3dk7/StegExpose

http://eric.univ-lyon2.fr/~ricco/cours/slides/roc_curve.pdf

https://ieeexplore.ieee.org/document/1625749

https://fr.wikipedia.org/wiki/LSB

https://link.springer.com/chapter/10.1007/978-3-642-40099-5_6

https://ieeexplore.ieee.org/document/8938486