TP d'informatique
Stéganographie

PCSI 2021 – 2022

I Introduction

La stéganographie est l'art de dissimuler des données. opposé à la cryptographie qui rend les données inintelligibles.

Différentes techniques existent, et ce depuis bien plus longtemps que les ordinateurs (écrire un texte dont on ne lit qu'une ligne sur deux par exemple, écrire à l'encre sympathique etc...).

Nous allons nous intéresser à une manière de cacher un message dans une image en modifiant les « bits de poids faible ».

Le principe est le suivant : une image correspond à un certain nombre de pixels, chacun étant composé de trois couleurs. Chacune des couleurs est (généralement) codée par un nombre entre 0 et 255. Un tel nombre se code sur 8 bits, c'est-à-dire sur un octet.
Toutefois, tous les bits n'ont pas « autant de sens ». Supposons qu'un nombre soit représenté en binaire par $\underline{1111\ 1111}_2$,

Ce chat veut vous dire quelque chose ... mais il va falloir le décoder.

Crédit photographique : Joseph SARDIN, license CC BY 2.0
`https://www.flickr.com/photos/14328577@N08/`

1. Que vaut ce nombre ?
2. Supposons qu'il y ait une modification du dernier chiffre, que devient ce nombre ?
3. Idem pour le premier chiffre ?
4. Laquelle des modifications à la plus grande influence ?

On va donc insérer du texte dans l'image en modifiant le bit de poids faible (celui qui le moins d'influence), ainsi, l'image sera visuellement la même ou presque, mais quelqu'un sachant ce qu'il doit chercher saura trouver l'information en ne regardant que le bit de poids faible.

Remarque : Il est ici important de manipuler un format d'image sans perte (pas jpg), nous manipulerons donc des images au format png (compression sans perte) [1].

II Quelques fonctions nécessaires

Pour parvenir à dissimuler notre texte, il nous faut des fonctions pour convertir un texte en liste de bits et réciproquement pour décoder l'image.

Exercice n°1 Écrire une fonction `liste_bit_vers_liste_nombre(liste_bit)` qui convertit une liste de nombre entre 0 et 1 de taille $8n$ en une liste de nombre entre 0 et 255 et de taille n. On supposera que chaque série de 8 bits représentera un nombre, avec le bit de poids faible en premier [2]
Par exemple, `liste_bit_vers_liste_nombre([0]*7+[1,1]+[0]*7)` doit renvoyer la liste `[128, 1]`. (Essayez de comprendre l'exemple avant de le coder).

1. Le format jpg compresse l'image mais en la détériorant, plus ou moins selon la qualité demandé.
2. en machine, il me semble qu'il existe des représentations où les octets sont dans un sens ou dans l'autre les uns par rapport aux autres (big endian, little endian). Toutefois, l'ordre des bits à l'intérieur d'un octet est le même a priori.

Exercice n°2 Écrire une fonction `nombre_vers_liste_bit(n)` permettant de convertir un entier n entre 0 et 255 en sa représentation binaire (avec la même convention que ci-dessus pour l'ordre des bits). Si vous avez du mal pour cette partie, une aide est disponible tout à la fin du TP.

Exercice n°3 Écrire la fonction `liste_nombre_vers_liste_bit(liste_nombre)` réalisant l'opération inverse de celle de l'exercice 1.

Exercice n°4 Pour convertir un caractère en nombre (code ASCII), python nous donne accès à la fonction **ord**. Écrire une fonction `char_vers_liste_nombre(chaine)` qui permet de convertir une chaine de caractère en une liste de nombre correspondant à leur représentation en ASCII.
Par exemple, `char_vers_liste_nombre('physique')` doit renvoyer la liste `[112, 104, 121, 115, 105, 113, 117, 101]`.

Exercice n°5 À l'aide de la fonction **chr** permettant de convertir un nombre en la lettre correspondante d'après la table ASCII, écrire une fonction `liste_nombre_vers_char(liste_nombre)` réalisant l'opération inverse.

III Manipulation d'image

Pour ouvrir une image, nous pouvons utiliser la bibliothèque PIL. Pour manipuler l'image, nous allons utiliser numpy dont nous connaissons déjà la structure.
Voici un exemple de code utilisant PIL :

```
import PIL.Image as im #attention à la majuscule d'Image
import numpy as np
image = im.open("image.png")  #ouverture
image = np.array(image)       #conversion
#faire ici diverse manipulation sur le tableau
sortie = im.fromarray(image)  #préparation à l'enregistrement
sortie.save('image_copie.png')#enregistrement avec un autre nom
```

Il est aussi possible d'utiliser scipy (pour les versions anciennes de python) ou imageio.

```
from scipy.ndimage import imread # ou pour les versions plus récentes :
from scipy.misc import imsave #from imageio import imread, imsave
im = imread('Chaton.png') # Lecture de l'image
imsave("image_copiee.png",im)
```

Exercice n°6 Utilisez PIL ou scipy tel que ci-dessus pour ouvrir une image et regarder la forme du tableau obtenu par numpy.
Rendre les 10 premières lignes noires (en mettant toutes les valeurs à 0) et sauvegardez l'image avec un autre nom. (Le but ici est simplement de manipuler un peu le tableau et la bibliothèque).

Exercice n°7 Rendez le chaton anonyme en plaçant un bandeau noir sur ses yeux (pixel allant de 340 à 640 en abscisse et de 245 à 305 en ordonnée par exemple, attention au lien abscisse/ordonnée/ligne/colonne).

IV Décodage du message dans une image

En python, nous n'allons pas directement manipuler les bits. Pour récupérer le bit de poids faible d'un nombre, il nous suffit de réaliser l'opération « modulo 2 ».

De plus, pour manipuler plus facilement le tableau numpy à trois dimension, nous allons le redimensionner pour n'avoir plus qu'un tableau à une dimension. Pour cela, la fonction ou la méthode reshape pourront vous servir[3]. Lorsque nous voudrons sauvegarder l'image pour l'encodage, il nous suffira de redonner sa forme initiale au tableau avant de le sauvegarder.
On propose l'algorithme suivant pour décoder :

1. ouvrir l'image ;
2. rendre le tableau 1D ;
3. récupérer le bit de poids faible pour chaque nombre du tableau ;
4. à partir de la liste de bit de poids faible et des fonctions réalisées au début, décoder le message.

Exercice n°8 Coder l'algorithme précédent en python sous forme d'une fonction `decode_message(image, N = 1000)` avec N le nombre de caractères que l'on veut décoder (la taille supposée du message, qui occupera donc $N \times 8$ bits dans l'image).
Tester votre fonction sur l'image fournie (cachotier_pour_TP.png sur le réseau du lycée dans P :/travail/informatique/IPT/PCSI)
Il y a un peu moins de 600 caractères dans le message qui y est caché, vous pouvez ne pas en décoder plus pour gagner du temps (surtout que les autres n'auraient pas de sens !).

V Codage dans une image

À vous de coder un message dans une image de votre choix maintenant.
On propose l'algorithme suivant pour coder :

1. ouvrir l'image ;
2. sauvegarder sa forme ;
3. rendre le tableau 1D ;
4. convertir le message à coder en liste de bits à l'aide des fonctions vues au début ;
5. rendre nul le bit de poids faible pour chaque nombre du tableau (ou du moins, sur une taille égale à celle du message que l'on souhaite coder). Ainsi, si le tableau représentant l'image est `[128, 235, 27]`, alors il sera modifié en `[128, 234, 26]` ;
6. ajouter au tableau case par case la valeur dans la liste de bit. Ainsi, si le tableau représentant l'image est `[128, 234, 26]` et le tableau de bit représentant le message est `[0, 1, 1]`, alors le tableau sera modifié en `[128, 235, 27]` ;
7. redonner à l'image sa forme correcte ;
8. enregistrer l'image.

Exercice n°9 Coder l'algorithme précédent en python sous forme d'une fonction `encode_message(message,image,image_sortie = 'message_code.png')` avec message le message à coder, image le chemin/nom de l'image utilisée pour cacher le message et image_sortie le chemin/nom de l'image à enregistrer avec le message codé.
Coder ensuite un message de votre choix et envoyer le à un autre étudiant pour qu'il le décode (tester aussi avec votre fonction programmée à la section précédente).

3. mais c'est une bonne idée de regarder la documentation et de faire un ou deux tests.

VI Un peu d'autonomie !

Il est aussi possible de cacher une image dans une autre (de même taille). Pour cela la technique est la suivante :

1. on cherche une image (n°2) de même taille que celle que celle que l'on veut dissimuler (image n°1) ;
2. on supprime les 4 bits de poids faible dans les deux images ;
3. on introduit les 4 bits de poids fort de l'image n°1 au niveau des bits de poids faible de l'image n°2 ;
4. on sauvegarde.

Exercice n°10 Réaliser le codage et le décodage d'une image dans une autre.
Vous pouvez tester votre décodage avec l'image image_cache.png fournie.

VII Un peu d'aide pour la décomposition en base 2

Pour la décomposition en base 2, je vous suggère l'algorithme suivant

- faire 8 fois :
  - prendre le modulo 2 du nombre et l'ajouter dans la liste
  - diviser le nombre par 2