Master 1 Informatique

Traitement d'image – 2024

TD3 – Messages cachés dans les images

Le but de ce TD est de mettre en place quelques méthodes de sécurité multimédia en utilisant la bibliothèque OpenCV en Python. Nous allons implémenter les techniques suivantes :

- Extraction d'un message caché d'une image.
- Insertion d'un message dans une image.
- Communication des messages secrets à travers des images (cryptographie visuelle).

Exercice 1. Tatouage – Extraction d'un message caché dans une image

Utilisez l'image `stego.png`

Retrouvez le message caché dans l'image `stego.png` en implémentant la fonction d'extraction d'un message caché dans une image. Suivez des étapes suivantes :

1. Lisez l'image `stego.png`.
2. Remplacez l'intensité représenter par un nombre pair par 0, l'intensité représenter par un nombre impair par 1.

Remarque : La taille du message caché dans l'image est 1176.

3. Convertissez chaque octet en nombre décimal et puis en caractère correspondant aux codes ASCII obtenus

Bonus : le message caché lui-même porte aussi une information supplémentaire. C'est un « piem » qui nous permet de retenir 23 premières décimales de pi. Le nombre de lettres de chaque mot donne le chiffre correspondant. Une particularité de ce « piem » : pour représenter le chiffre 0, le poète utilise un mot de 10 lettres.

4. Proposez un algorithme qui nous permet de récupérer le nombre caché dans ce texte.

Exercice 2. Tatouage – Insertion d'un message dans une image

Algorithme de tatouage :
Une image peut être vu comme une suite des nombres compris entre 0 et 255. La technique de tatouage à implémenter est la suivant :

– un nombre pair correspond à un 0 du texte ;
– un nombre impair correspond à un 1.

On va donc modifier certains pixels de l'image, mais ces altérations seront invisibles à l'œil.

Implémentez cet algorithme de tatouage.

1. Choisissez un message à cacher et convertissez-le en nombre binaire.
2. Insérer le message choisi en modifiant les intensités de pixels d'image :
   - Si l'intensité d'un pixel est un nombre pair et le bit de message est 0, on le laisse inchangé ;
   - Si l'intensité d'un pixel est un nombre pair et le bit de message est 1, on ajoute 1 à l'intensité de ce pixel.

Attention aux débordements : si la valeur d'un pixel est 255, on soustraira 1.

Exemple : Lettre « A » a le code ASCII `65`, en binaire c'est `01000001` :

`Image originale`	`255`	`255`	`255`	`153`	`219`	`5`	`102`	`0`	…
`Texte binaire à cacher`	`0`	`1`	`0`	`0`	`0`	`0`	`0`	`1`	…
`Image tatouée`	`254`	`255`	`254`	`154`	`220`	`6`	`102`	`1`	…

3. Testez l'algorithme implémenté sur une image en format `png` à votre choix.

Attention : Cette méthode ne prends pas en compte des modifications faites par les méthodes de compression d'image. C'est pourquoi, il fonctionne seulement avec des images en formats `bitmap` ou `png`**.** En effet, beaucoup de formats compressent les données et modifient les bits de l'image, et donc cela détruit le message caché.

Travail en autonomie

La cryptographie visuelle a été inventée en 1994 par Moni Naor et Adi Shamir. Elle permet de communiquer des messages secrets à travers des supports visuelles (images, texte, etc.). Dans la méthode classique on manipule des images simples en noir et blanc. Les images sont constituées uniquement de pixels noirs et blancs.

Schéma (2,2)-VCS (Visual Cryptography Scheme)

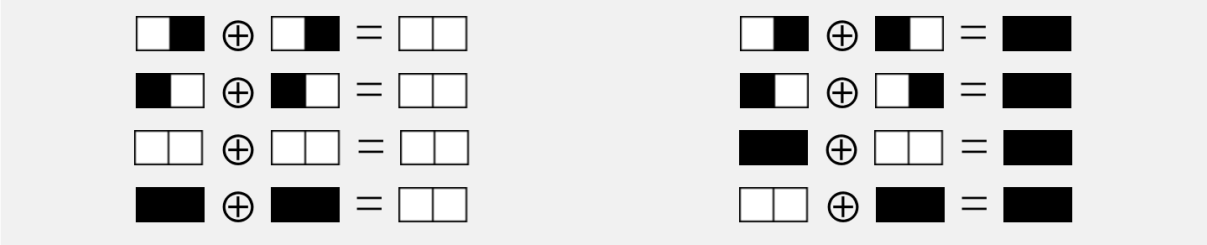
L'image contenant l'information secrète est partagée en 2 images masque (*shadow image*) et peut être reconstituée en superposant ces 2 images.

Par « superposition », on entend une opération logique OU entre chaque sous-pixel de mêmes coordonnées (image transparente superposée à une image opaque) ou bien XOR.

Principe

Chaque pixel de l'image à chiffrer est partagé en 2 « blocs » de `m` sous-pixels (appelés *shares*). Ces blocs constituent alors les pixels des images masque, dont la taille ne sera pas conséquent `m` fois plus grande (en nombre de pixel) que celle de l'image à chiffrer.

L'opération de « superposition » entre 2 *shares* doit donner :
- pour un pixel d'origine noir : `m` sous-pixels noirs
- pour un pixel d'origine blanc :
  - soit `m` sous-pixels blancs (opération XOR)
  - soit `m` sous-pixels noirs ou blancs (opération OR)



Le codage

L'image est tout d'abord binarisée (deux niveaux sont conservés, noir et blanc). Après binarisation, l'image source est donc constituée de 0 (pixels noirs) et de 1 (pixels blancs).
Ensuite on crée 2 images masque vides.

Puis le chiffrement est effectué :

Pour chaque pixel de l'image d'origine :

1. On génère aléatoirement un *share* de `m` sous-pixels et on le place dans la 1ère image masque.
2. On détermine le 2ème *share*, associé au 1er de telle sorte que la superposition des 2 donne la valeur du pixel de l'image d'origine :
   `share1 ⊕ share2 = m` pixels noirs ou `m` pixels blancs
3. On place ce 2ème *share* à la place dans la 2ème image masque.

Exercice 3. – Cryptographie visuelle

1. Implémentez en Python une fonction `vcs(nom_fichier_image)` qui partage l'image `nom_fichier_image` en deux images masque, en utilisant des blocs de 4 sous-pixels, puis qui les enregistre sous deux noms de fichier : `nom_fichier_image_1` et `nom_fichier_image_2`.
2. Implémentez maintenant une fonction de décodage : fusion de deux masques pour retrouver l'image initiale.

Lecture supplémentaire :

[1] M. Naor , A. Shamir, "Visual Cryptographie"
https://link.springer.com/content/pdf/10.1007/BFb0053419.pdf
[2] Cryptographie visuelle https://info.blaisepascal.fr/nsi-cryptographie-visuelle
[3] Cryptographie Visuelle https://www.lama.univ-savoie.fr/mediawiki/index.php/Cryptographie_Visuelle