Cryptographie et Sécurité
Série 4, version Hidden Blue :
Stéganographie - Image Chiffrée, Image Cachée

13 Octobre 2022

Exceptionnellement cette semaine, deux TP4 différents vous sont proposés. Vous êtes libres de choisir lequel des deux vous préférez faire et rendre.
Cette version, Hidden Blue, propose une "chasse au trésor" basée sur la stéganographie.
L'autre version, Secret Red, propose un challenge CTF (Capture The Flag) basé sur un système d'encryption inhabituel.

A rendre sur **Moodle**, avec un fichier python **.py**, au plus tard le Mercredi **19 Octobre 2022 à 23h59**.

Commentez votre code.

Le titre de cette section est caché dans l'exemple.

La stéganographie vise plus ou moins le même but que l'encryption : faire en sorte que le message ne soit lisible que par les personnes qui doivent le lire. Mais cela se fait de manière totalement différente : dans le cas de la cryptographie, le message n'est pas caché au sens propre du terme, il est plutôt masqué : on peut voir le message, mais sous une forme cryptée, qui n'est pas intelligible sans les bons outils (clé, mot de passe, etc).
La stéganographie emploie une autre approche : le message n'est pas crypté, mais il est caché, au sein d'une autre information beaucoup plus grande.

Exemple :

Soit un exemple simple historique, qui consiste à utiliser les majuscules d'un texte. Toutes les majuscules d'un texte forment le "vrai" message. Et ce message est caché dans le texte intégral, qui le masque. Globalement, on ne suppose pas qu'un message est caché dans les majuscules si on ne cherche pas à decrypter

quelque chose. Alors on peut faire passer un message. Néanmoins, c'est une méthode facile à détecter. On observe souvent des structures de phrases étranges. Globalement, c'est très facile de repérer ce procédé et de le décrypter. Retrouver le message est donc facile. A la place, on va donc utiliser un procédé plus poussé. Par exemple, cacher un message dans certains bits d'une image. Hélas, cela nécessite d'utiliser une grande image pour que l'altération ne soit pas visible. Il faut en effet que l'image paraisse inaltérée, sinon on se doute qu'un message y est caché. Et notez que ce texte est étrangement écrit car il contient un tel message caché.

Images, pixels et bits

Une méthode plus efficace et passe partout qui est aussi très connue, et qui est celle à laquelle on va s'interesser principalement ici, est de cacher des messages dans certains bits d'une image :

Une image est généralement représentée comme un tableau de pixels. Par exemple, une image de taille 1080x720 pixels est un tableau, dont la longueur est 1080 pixels, et la hauteur 720 pixels.

Ici, on utilise ce qu'on appelle le codage RGB (Red, Green, Blue). Un pixel est une liste de 3 valeurs, chacune codée sur un octet : la première correspond à la couleur rouge, la deuxième à la couleur verte, et la troisième à la couleur bleue. Chaque valeur est un entier entre 0 et 255.

Comme les changements très légers sont difficilement perceptibles par l'oeil humain, on s'intéresse généralement au least significant bits :

Sur une certaine couleur d'un pixel exprimée entre 0 et 255, c'est à dire 8 bits, changer le bit de poids faible, ou les deux bits de poids les plus faibles, n'a que peu d'impact sur l'image. Mais cela peut permettre de cacher de l'information. Par exemple, utiliser seulement le least significant bit de la couleur rouge, permet d'avoir 1 bit sur 24 (1 par pixel) dans lequel on cache de l'information. Sur une image de 24 Mo, cela permet de cacher 1 Mo d'information.

On peut également n'utiliser que le 2ème least significant bit, ou bien les deux plus faibles, utiliser toutes les couleurs, ou une seule, on peut utiliser toute l'image, ou seulement une partie, etc. Certaines méthode plus poussées utilisent les coutours des objets ou les zones de forte variance pour y cacher l'information.

L'une des faiblesses de ce principe, c'est qu'on est très sensible aux erreurs et aux compressions avec pertes notamment. On introduit donc souvent de la redondance pour éviter cela. Des méthodes beaucoup plus poussées permettent de meilleurs résultats. En général, on a souvent un compromis à faire entre la quantité d'information qu'on peut cacher, la redondance, et la discretion.

Par exemple, pour cacher le mot "bonjour" dans une image, on devrait par exemple l'exprimer en ascii (b = 01100010, o = 01101111, n = 01101110, ...), puis cacher chaque bit quelque part dans l'image (disons, pour faire simple, le least significant bit (bit de poids faible) du rouge).

Supposons que les valeurs des huit premiers pixels, pour la couleur rouge, sont (24,22,20,17,15,13,12,10).

Pour chaque valeur, exprimee sur 8 bits, on cache le bit nécessaire à encoder le "b".

Par exemple, pour 24 = 00011000, et le premier bit à cacher, "0", on a rien à changer : le dernier bit de 24 est déjà à 0.

Pour le suivant, 22 = 00010110, et le second bit à cacher, "1", on doit modifier cette valeur. On la change donc en 23 = 00010111.

Et ainsi de suite, on cache chaque bit dans les pixels rouges consécutifs. On obtiendrait ici : (24,23,21,16,14,12,13,10). Et on a ici caché le "b" dans le least significant bits de ces 8 valeurs de la couleur rouge.

On cache parfois même des images (de qualité moindre) ainsi dans d'autres images !

Le TP

Vous aurez besoin d'installer deux librairies pour ce TP : Numpy et Pillow.

Sur moodle, vous trouverez un package contenant :

- Une image encryptée par un one-time pad : la clé est donc une autre image de même taille. On a appliqué un XOR entre les deux images, pixel par pixel, et couleur par couleur (pour un pixel donné $[rgb]$ et le pixel de la clé $[r'g'b']$, la pixel résultant est $[r \oplus r', g \oplus g', b \oplus b']$).
  Ici, vous avez le résultat de cette encryption, appelé "Cipherimage".
- Trois fonds d'écrans. Chacun contient quelque chose de caché par stéganographie : deux contiennent des messages, et la troisième contient la clé utilisée pour le one-time pad.
- Enfin, un fichier Python contenant un "starterpack" pour bien débuter ce TP : quelques commandes utiles pour charger une image, en récupérer le tableau de pixels, le modifier, le sauvegarder.

Le but de ce TP est de dechiffrer l'image chiffrée. Pour cela, vous allez devoir suivre les indices : vous trouverez-ci après une indication pour chercher le 1er message caché dans l'un des fonds d'écran. Celui-ci vous expliquera ou et comment trouver le second message caché, qui lui-même vous expliquera comment trouver l'image clé cachée dans le dernier fond d'écran (que vous pourrez alors utiliser pour déchiffrer l'image "cipherimage").

Votre rendu devra, à partir des images, sortir les différents messages et images cachées (printer les messages à l'écran et sauvegarder les images comme des fichiers).

Enfin, quelques détails complémentaires pour vous aider :

- Le premier message est caché **dans l'image Aerith. Il est dans le least significant bit de la couleur verte, dans les pixels de la 43ème ligne à la 47ème ligne de l'image.** Vous devez donc récupérer le least significant bit de la couleur verte de chaque pixel de ces 5 lignes, et cela vous donnera une grande chaine de bits. Cette chaine de bits, traduite en ascii 8-bit (chaque octet est un caractère) vous donnera un message.
- Ce message, que vous trouverez comme indiqué précedemment, ne donne pas directement la marche à suivre pour la suite : il s'agit simplement des paroles d'une chanson. Mais dans ces paroles, une méthode pour cacher de l'information dans un texte à été utilisée (méthode mentionnée précédemment dans ce TP).
  Aide : La fonction ".isupper()" permet de déterminer si un caractère x est une majuscule ou non. Cela pourrait aider à lire plus facilement ce message caché.
- Ce message caché vous indique ou chercher le deuxième message. Celui-ci vous explique dans quelle image chercher et quels bits utiliser pour reconstruire l'image-clé.
- Enfin, cette image-clé à récupérer dans la dernière image, pour être un bon one-time pad, n'est pas une image compréhensible (elle ressemble à une bouillie de pixels). Il est donc normal que cette image-clé ne ressemble à rien. En revanche, lorsque vous ferez le XOR de cette image-clé avec l'image chiffrée, vous devriez retomber sur une image finale cohérente.