

TS-ISN

Lycée Marguerite Yourcenar

Stéganographie et cryptographie d'images

Projet d'informatique et sciences du
numérique

Projet de Solène HIRLES et Ariane KAYVANTASH

dossier personnel d'Ariane KAYVANTASH

2018-2019

Sommaire :

<u>I. Présentation du projet</u>	3
<u>A)Présentation</u>	3
<u>B) Répartition des tâches</u>	3
<u>C) Cahier des charges</u>	4
<u>II. Réalisation</u>	5
<u>A) L'interface d'utilisateur</u>	5
<u>B) Le décodage stéganographique</u>	5
<u>C) La cryptographie</u>	6
<u>III. Conclusion</u>	8
<u>Annexes</u>	8

I. Présentation du projet:

A)Présentation

Notre projet est un programme permettant de coder et de décoder une image grâce à deux méthodes : la cryptographie et la stéganographie.

La cryptographie consiste à envoyer une image codée, de façon à ce qu'elle ne soit pas compréhensible pour une personne autre que le destinataire, et que seul ce dernier puisse la décoder. La stéganographie d'image, elle, est le fait de dissimuler une image dans une autre.

Pour faire ce programme, nous avons dû le diviser en six parties : l'interface utilisateur, le codage stéganographie, le codage cryptographique, l'algorithme de binarisation le décodage stéganographique et le décodage cryptographique.

B) Répartition des tâches:

Ces six tâches ont été distribuées de la manière suivante :

Solène HIRLES : Algorithme de binarisation, codage stéganographique et cryptographique
Ariane KAYVANTASH : Interface utilisateur, décodage stéganographique et cryptographique

C) Cahier des charges:

Le projet se rapporte aux domaines de compétences suivants :

- **Dimension algorithmique** : Création d'algorithmes qui permettent de coder une image (ou un message pour la cryptographie) selon deux méthodes : stéganographie et cryptographie à clé secrète.
- **Éléments de programmation** : Pour notre projet, nous utilisons le langage de programmation Python, un éditeur de programme et une console qui affiche les résultats du programme exécuté dans l'éditeur. Nous avons également utilisés les modules et bibliothèques suivantes : PIL Images (pour travailler sur les images et les afficher), numpy (pour créer des tableaux et les convertir en images) et Tkinter (pour l'interface utilisateur).

Production finale attendue : Un algorithme qui permet de coder n'importe quelle image selon une clé générée aléatoirement et des algorithmes de codage et de décodage permettant de cacher une image dans une autre. Il permet aussi de coder et décoder un message.

Caractéristiques de la production finale : la stéganographie (le codage et décodage) et l'interface fonctionnent. Néanmoins, le codage de cryptographie fonctionne partiellement et le décodage de cryptographie à des problèmes de fonctionnement.

Contraintes à respecter : Date butoir, travail en équipe, dossier écrit (5-10 pages)

Matériel et logiciel à mettre en oeuvre : Python

II. Réalisation

A) L'interface d'utilisateur

Pour faire cette interface, j'ai utilisé le module tkinter. Pour la faire, j'ai d'abord essayé d'afficher une fenêtre donnant la définition des deux méthodes (cryptographie et stéganographie) et affichant deux boutons, chacun proposant une des méthodes. Lorsque l'on appuyait sur un de ces bouton, une nouvelle fenêtre s'affichait proposant de choisir entre le codage et le décodage, puis, après avoir entré l'adresse url de l'image (ou des images), déclenchait le sous-programme concerné. Cependant, je l'ai finalement changé puisque cette interface avait plusieurs problèmes: elle affichait trop de fenêtres, je n'arrivais pas à faire apparaître le cadre pour rentrer le lien url...

J'ai finalement modifié l'interface. Il s'agit désormais d'une seule fenêtreproposant le choix entre les deux méthodes (cryptographie ou stéganographie) par l'intermédiaire de boutons qui, quand l'utilisateur appuie dessus, font tous deux apparaître un menu déroulant permettant de choisir entre le décodage et le codage (pour la cryptographie, on choisit entre le codage/décodage d'une image ou d'un texte) pour chaque méthode, appuyer sur ces boutons amène alors à une fenêtre pour rentrer l'adresse url des images concernées pour ensuite lancer la fonction correspondantes.

B) Le décodage stéganographique

Pour le décodagestéganographique, j'ai d'abord commencé par créer un cadre pour entrer l'adresse url de l'image codée, puis, j'ai créé deux fonctions : la première nommée *cachea* pour rôle de mettre à 0 les quatre premiers bits de points forts de chaque octets et la

deuxième nommée *decalage* qui a pour rôle de décaler les quatre bits de poids faible des octets vers la gauche, ils deviennent ainsi des bits de poids fort. Dans le corps principal du programme, je récupère la longueur et la largeur de l'image puis, j'ai créé une autre image de mêmes dimensions et couleurs que l'image codée. Puis j'ai créé deux boucles **for** pour que pour tout les pixels de chaque ligne puis de chaque colonne, on récupère les composants RVB des pixels de l'image. Ensuite, je leur ai appliqué les fonctions *cache* et *decalage*. J'obtiens alors les trois composants des pixels de l'image finale. Enfin, je sauvegarde l'image finale et je l'affiche.

C) La cryptographie

Le décodage cryptographique a quelques soucis de fonctionnement.

Pour décoder une image, j'ai fait comme ceci :

On commence par faire apparaître des fenêtres pour rentrer l'adresse url de l'image et la clé obtenue grâce au codage, puis, j'ai créé deux listes d'éléments. On crée ensuite une liste vide avec *liste_finale=[]*. J'ai ensuite initialisé l'indice de l'élément, *i*, puis, tant que *i* sera inférieur à la longueur de l'image, on compare des éléments des deux listes (celle de l'image codée et celle de la clé). Si les deux sont égaux, on met la valeur 0 dans la liste finale et on obtient un pixel noir. À l'inverse, s'ils sont différents, on ajoute la valeur 1 dans la clé et on obtient un pixel blanc. Ensuite, on sauvegarde et on affiche l'image. Cependant il manque quelques lignes de code pour que la fonction fonctionne correctement.

Nous avons rajouté des fonctions pour procéder à la cryptographie de messages (codage et décodage), cela consiste à cacher un message dans une image. Pour le codage, après avoir lu l'image, créer une autre image de mêmes dimensions et créer une liste à partir de la première image, on commence par créer une fonction, nommée *transforme_message_en_liste*, pour transformer la chaîne caractère en liste de digits correspondant au code unicode du

caractère (pouvant aller de 0 à 255) grâce à *ord(caractere)* que l'on convertit en binaire grâce à *bin()*. Grâce à *[2:]*, on retire les deux premiers éléments de la liste et on renvoie le reste. Ainsi, on renvoie les éléments de la liste chaîne sauf les deux premiers. On crée ensuite une chaîne composée de 8 caractères contenant la chaîne et préfixé par des caractères "zéros".

On transforme alors tout le message en une liste de digits, chaque caractère (code supposé entre 0 et 255) étant transformé en son code unicode sur une longueur 8 par la fonction *transforme_message_en_liste*. On utilise pour cela une boucle **for** et la méthode *extend()* pour appliquer cette fonction et mettre l'élément calculé dans la liste. On utilise ensuite une autre fonction pour effectuer une division euclidienne de l'entier compris entre 0 et 255 par 2, puis on renvoie un nombre entier inférieur divisible par deux + digit (digit peut prendre la valeur de 0 ou de 1). Ensuite, on crée la fonction *cacheMessage* pour construire une liste digit. Il s'agit d'une liste de triplets (r, g, b) d'entiers entre 0 et 255. Le message sera caché dans les triplets à raison d'un bit par triplet (chaque bit de poids faible de la composante b est remplacé par un bit du code unicode d'un caractère du message). On commencera cependant par le triplet (r, g, b) d'indice 1, le triplet d'indice 0 sera réservé au stockage de la longueur du message. Enfin, on rentre le message, on le cache dans l'image en appelant la fonction *cacheMessage* et en récupérant la liste et on sauvegarde l'image sous le format **png**.

Pour le décodage du message, on commence par ouvrir l'image et créer une liste de pixels. Puis, on fait une fonction nommée *recupereLongueur* pour récupérer la longueur de la liste. On crée ensuite une nouvelle fonction nommée *recupereMessage* où l'on appellera la fonction précédente pour donner la longueur de la liste, que l'on multipliera ensuite par 8. On crée ensuite les variables message et lettre et une boucle **for** qui, pour tout i entre 1 et la longueur plus 2, si la longueur de la chaîne lettre est inférieur à 8, on lui ajoute un élément, sinon on ajoute dans le message le caractère correspondant au nombre entier compris entre 0 et 255, après cela,

on renvoie le message. Enfin, on applique cette fonction sur la liste de pixels de l'image et on affiche le message.

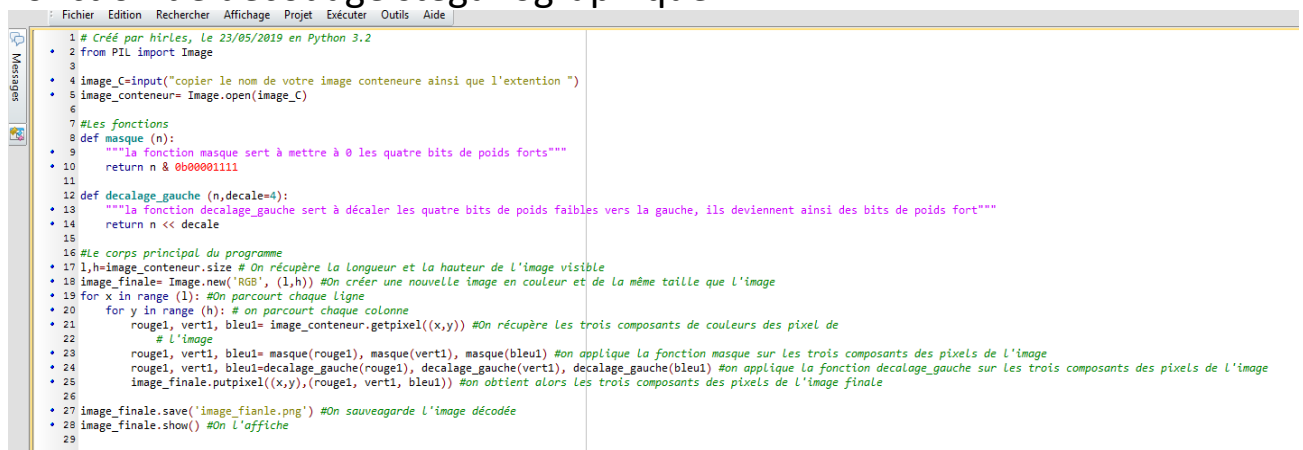
III. Conclusion

Ce projet m'a aidé et m'a appris des méthodes que je ne connaissais pas : la cryptographie et la stéganographie. Ce projet et la spécialité d'Informatique et Sciences du Numérique m'ont donné de l'expérience en informatique et en python qui me sera sans doute utile dans ma vie future. Il m'a aussi permis de connaître tkinter et de savoir m'en servir.

Je regrette seulement de ne pas avoir réussi à faire la fonction de décodage cryptographique. Néanmoins, je suis satisfaite du fait que l'interface et la stéganographie fonctionnent correctement.

Annexes :

Fonction de décodage stéganographique :



```
1 # Créé par hirles, Le 23/05/2019 en Python 3.2
2 from PIL import Image
3
4 image_c=input("copier le nom de votre image conteneure ainsi que l'extention ")
5 image_conteneur= Image.open(image_c)
6
7 #Les fonctions
8 def masque (n):
9     """la fonction masque sert à mettre à 0 les quatre bits de poids forts"""
10    return n & 0b00001111
11
12 def decalage_gauche (n,decale=4):
13    """la fonction decalage_gauche sert à décaler les quatre bits de poids faibles vers la gauche, ils deviennent ainsi des bits de poids fort"""
14    return n << decale
15
16 #Le corps principal du programme
17 l,h=image_conteneur.size # On récupère la longueur et la hauteur de l'image visible
18 image_finale= Image.new('RGB', (l,h)) #On crée une nouvelle image en couleur et de la même taille que l'image
19 for x in range (l): #On parcourt chaque ligne
20     for y in range (h): # on parcourt chaque colonne
21         rougel, verti, bleu1= image_conteneur.getpixel((x,y)) #On récupère les trois composants de couleurs des pixel de
22         # l'image
23         rougel, verti, bleu1= masque(rougel), masque(verti), masque(bleu1) #on applique la fonction masque sur les trois composants des pixels de l'image
24         rougel, verti, bleu1=decalage_gauche(rougel), decalage_gauche(verti), decalage_gauche(bleu1) #on applique la fonction decalage_gauche sur les trois composants des pixels de l'image
25         image_finale.putpixel((x,y),(rougel, verti, bleu1)) #on obtient alors les trois composants des pixels de l'image finale
26
27 image_finale.save('image_finale.png') #On sauvegarde l'image décodée
28 image_finale.show() #On l'affiche
29
30
```