LYCEE BLAISE PASCAL – CLERMONT-FERRAND

Stéganographie et cryptologie visuelle

Projet d'Informatique et Sciences du Numérique

Groupe : Naouel KOUISS & Clémence VESSAIRE - TS5

Dossier personnel : Clémence VESSAIRE

2016-2017

Sommaire

Présentation du projet

Avec Naouel, nous nous sommes demandées s'il était possible de modifier une image de façon à ce que seul son destinataire puisse la voir. Notre professeur d'Informatique et Sciences du Numérique, M. Faury, nous a orienté sur deux méthodes, la stéganographie et la cryptologie visuelle.

La **stéganographie** est une méthode qui a pour but la *dissimulation* d'une information dans une autre. Cette méthode est connue depuis l'antiquité et elle a de multiples variantes, fonctions du support, de la méthode ou de la technologie existante. Dans ce projet, nous nous intéressons uniquement à la dissimulation d'une image dans une autre en en créant une troisième. En utilisant la stéganographie, on veut que, lorsqu'un autre que le destinataire découvre l'image obtenue, il ne puisse pas soupçonner l'existence d'une autre image dissimulée dans celle qu'il voit, ni même qu'il soupçonne un contenu caché. Ainsi, l'observateur ne peut pas empêcher la transmission de l'image cachée et ne peut pas non plus utiliser la même méthode pour faire passer d'autres informations fausses. Dans notre projet, pour dissimuler une image dans une autre, nous modifions les bits de poids faibles des pixels de l'image contenante.

Cependant, la stéganographie est une méthode connue et si l'on soupçonne une image de contenir une information secrète avec cette méthode, il est aisé de réussir à la récupérer. Aujourd'hui, il existe d'autres méthodes qui permettent de rendre incompréhensible une information à une personne autre que son destinataire.

La **cryptologie** permet de rendre une information *secrète* et englobe la cryptographie, le codage et la cryptanalyse, le décodage. Elle est également connue depuis l'antiquité mais acquiert le statut de sciences seulement dans les années 1970 en devenant thème de recherche avec l'essor de l'informatique puis d'internet et des communications. Le but de la cryptologie visuelle, c'est que seul l'œil humain doit être capable de décoder et d'authentifier le message. Dans ce projet, pour cacher une image, nous utilisons une méthode de cryptographie à clé secrète qui permet de chiffrer et déchiffrer à partir de la même clé. L'image doit être binarisée (seulement en noir et blanc, sans niveau de gris) et la clé est une image de taille proportionnelle à celle de l'image, composée aléatoirement de pixels noirs et blancs.

Cahier des charges

Le projet se rapporte aux domaines de compétence suivants :

- **Dimension algorithmique :** Création d'algorithmes qui permettent de coder une image selon deux méthodes : stéganographie et cryptologie à clé secrète.

- **Éléments de programmation :** Pour la réalisation, nous utilisons le langage de programmation Python avec la distribution Pyzo qui inclut des modules complémentaires, un éditeur de programme et une console qui affiche les résultats du programme exécuté dans l'éditeur. Nous avons utilisé ces modules ou bibliothèques : *imageio* (pour travailler avec les images), *matpotlib* (pour afficher les images), *pillow* (pour travailler avec les images et en créer à partir de tableaux de valeurs numpy), *numpy* (pour créer des matrices), et *tkinter* (pour l'interface utilisateur).

Production finale attendue : Un algorithme qui permet de coder n'importe quelle image selon une clé donnée (ou générée aléatoirement) *(cryptographie)* ou bien de la cacher dans une autre (*stéganographie*) et de la décoder dans les deux cas.

Caractéristiques de la production finale : Fonctionne correctement, interface simple qui permet de rentrer une image ou de choisir la méthode.

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

Matériel et logiciel à mettre en œuvre : Python et éditeur d'image

Répartition des tâches

Clémence	Naouel
Codage stéganographie	Décodage stéganographie
Codage cryptographie	Décodage cryptographie
Algorithme de binarisation d'image	Interface utilisateur

Réalisation

Pour toute la réalisation du projet, nous nous sommes basées sur un document ressource de l'Education Nationale écrit en 2014 par Jean-Christophe Sahakian qui décrit les fonctionnements de la stéganographie et de la cryptographie.

→ *Voir* Sources

Stéganographie

On vient de voir que la stéganographie permet de cacher une image dans une autre. Le codage et le décodage se font en modifiant directement les bits de chaque octet de chaque pixel des deux images. Cela peut donc prendre longtemps, notamment si les deux images sont grandes.

Principe du codage

On travaille à partir de deux images de même taille que l'on nommera "*image conteneur*", celle qui sera celle visible et "*image secrète*", celle qui sera cachée.

Comme nous travaillons avec des images en couleurs, chaque pixel de l'image peut être défini par quatre octets maximum : les trois premiers servant à coder les proportions de rouge, de vert et de bleu et le quatrième pour une information de transparence. Le codage se fait en deux étapes :

- <u>Première étape :</u> Sur chaque pixel de l'image conteneur, on met à zéro le quartet de poids faible de chaque octet (les quatre bits qui se trouvent à droite et qui ont la plus faible valeur). Sur chaque pixel de l'image secrète, on décale vers la droite le quartet de poids fort de chaque octet (les quatre bits qui se trouvent à gauche) et qui devient alors le quartet de poids faible.
- <u>Deuxième étape :</u> Ici, on a donc une image qui n'a plus de quartet de poids faible, l'image conteneur, sur ses octets et une, l'image secrète, qui n'a plus de quartet de poids fort. On assemble donc ces deux composantes en effectuant un OU logique entre les deux octets. Les informations de l'image secrète se retrouvent alors dans le quartet de poids faible de l'image conteneur.

→ *voir illustration dans* Annexes *:* Figure 1

Mise en œuvre

→ *Pour voir le code du codage, voir* Annexes *:* Figure 2

J'utilise Pyzo pour travailler sur le projet. Nous utilisons le module *imageio* pour pouvoir lire des images à partir de leurs adresses url. Pour pouvoir les demander à l'utilisateur, nous utilisons la commande *input* et on leur attribue leurs noms : *image_secrete* et *image_conteneure*. Elles doivent être de la même taille.

Pour réaliser le codage de la stéganographie, on crée une fonction **codage_image** qui a comme arguments nos deux images.

On demande d'abord à l'utilisateur sur combien de bits il veut coder les informations de l'image secrète sur l'image conteneur en sachant que plus le nombre de bits de poids faibles utilisés pour cacher l'image secrète est faible, mieux l'image est cachée mais plus elle en perd en qualité. Encore une fois, on utilise une fonction input. Avec un dictionnaire, on associe ce nombre x à un autre, en binaire, dont nous reparlons plus tard. On crée le nombre correspondant au décalage avec `nombre = 8-x`.

On a ensuite besoin de deux listes qui contiennent autant de valeurs que la largeur et la hauteur des images La fonction *.shape* d'imageio permet d'obtenir la largeur et la hauteur d'une image sous forme d'un tuple que l'on peut appeler avec des crochets. On dispose donc des deux valeurs de largeur et hauteur. Alors, on utilise les fonctions `list(range(x))` qui permet d'obtenir une liste d'entiers de x valeurs en partant de 0 jusqu'à x.

Pour pouvoir modifier chaque octet de chaque pixel, on parcourt chaque ligne, puis chaque pixel de cette ligne avec deux boucles *for* imbriquées sur les deux images en même temps. On récupère les valeurs de ces deux pixels avec *imageio*. Celui le permet avec cette commande : `nom_image[coordonné_largeur][coordonnée_hauteur]` et retourne des tuples. Pour pouvoir les modifier, on les transforme en liste avec `list()`.

On commence la première étape du codage en créant des listes vides `pixel_conteneur` et `pixel_secret` qui contiendront les valeurs des pixels obtenues avec le processus de codage. Avec deux boucles for, on parcourt chaque octet du pixel en cours de chaque image. On les modifie avec la méthode expliquée en effectuant des opérations bit à bit.

Pour le pixel de l'image conteneur, ses octets sont modifiés comme ceci : on met à 0 les octets sur lesquels vont être codés les informations de l'image secrète. Pour cela, on effectue un ET logique (&) entre la valeur de l'octet un octet composé de x 0 à droite et (8-x) 1 à gauche où x est le décalage choisi par l'utilisateur. Par exemple, si l'utilisateur choisit de coder sur 3 octets, l'octet utilisé pendant le calcul sera `0b11111000` (`0b` indique à python que le nombre est en binaire). Ainsi, si la valeur de l'octet est `0b22222222`, le ET logique donne : `0b22222222 & 0b11111000 = 0b22222000`.

Pour le pixel de l'image secrète, on décale vers la droite les bits de l'octet par le décalage, lequel est fonction du nombre choisi par l'utilisateur. En effet, si on utilise le nombre tel quel, on obtiendrait `0b12345678 >> 3 = 0b00012345`, qui ne s'assemblerait pas avec le résultat précédent. Sur python, on utilise la commande >>x pour cette opération bit à bit de décalage de x rangs. Pour les mêmes hypothèses que précédemment, l'opération

donne donc : `0b12345678 >> 5 = 0b00000123`. On ajoute ces deux valeurs dans les listes vides correspondantes. La première étape du codage est terminée.

Pour la deuxième étape, il faut assembler les résultats en un seul octet avec un OU logique : `|` sur python. Cela donne : `0b22222000 | 0b00000123 = 0b22222123.` La deuxième étape du codage est terminée. On transforme en tuple la valeur obtenue pour le pixel ainsi codé et on l'affecte au pixel de l'image conteneur actuellement dans la boucle pour coder ce pixel. La deuxième boucle passe au pixel suivant sur la même ligne et recommence. Une fois l'image ainsi modifiée, on l'affiche avec *Matplotlib* et la commande suivante `matplotlib.show(image_conteneure)`.La fonction **codage_image** est terminée.

On crée une structure conditionnelle if – else pour vérifier que les deux images données par l'utilisateur font bien la même taille et si oui, on appelle la fonction **codage_image**.

Interface

En plus de l'interface générale, j'ai fait une interface tkinter pour demander à l'utilisateur sur combien de bits il souhaite coder son image. Pour cela, j'ai d'abord crée une fenêtre. Ensuite, j'y ai inséré un label pour afficher le texte dont j'avais besoin pour expliquer à l'utilisateur à quoi correspond ce chiffre. J'ai ensuite crée des RadioButton auxquels sont associées les valeurs à choisir. Ils se présentent sous la forme de ronds que l'on peut cocher. Dès qu'il y en a un de coché, les autres s'annulent. Cependant, comme je n'ai pas réussi à récupérer la valeur, j'ai changé pour une fonction d'entrée d'une valeur.

→ *Voir* Figure 3 *et* Figure 4 *dans* Annexes *pour les deux fenêtres*

J'ai ensuite crée une fonction **choix_nombre** pour récupérer la valeur choisie et l'insérer dans l'algorithme de codage de la stéganographie. Celle-ci renvoie une fenêtre d'information contenant la valeur. Néanmoins, je n'ai pas réussi à me servir de cette valeur dans l'algorithme de stéganographie.

→ *Voir* Figure 5 *et* Figure 6 *dans* Annexes *pour la fenêtre de valeur et le code*

Cryptographie

Principe de codage

On travaille sur une image binarisée, c'est-à-dire seulement en noir et blanc, sans niveau de gris. Le chiffrement se fait en 3 étapes :

- Première étape : On crée un tableau aléatoire de 0 et de 1 qui a la même taille que l'image à chiffrer. Pour chaque pixel de cette image, il y a donc un nombre aléatoire, un aléa, qui lui est associé dans le tableau.
- Deuxième étape : On crée la clé. Pour cela, on utilise le tableau aléatoire et une clé choisie que les deux membres, destinataire et expéditeur connaissent. On associe ensuite cette clé avec le tableau aléatoire. Ici, nous avons décidé que la clé correspondrait à ceci : si l'aléa du tableau est 0, la clé donne un pixel blanc et un pixel noir ; si l'aléa du tableau est 1, la clé donne un pixel noir et un pixel blanc. La clé est une image qui doit rester secrète, elle doit donc être transmise par un autre biais que l'image. On a crée le bipixel clé.
- Troisième étape : On effectue un OU exclusif entre chaque composante de la clé, chaque bipixel et le pixel qui lui correspond sur l'image. On obtient une image chiffrée qui fait le double de l'image originale.

Mise en œuvre

On travaille sur Pyzo avec les modules imageio, numpy, random qui sont déjà installés sur Pyzo et PIL Image que l'on installe avec pip. Comme pour la stéganographie, on récupère l'image de l'utilisateur avec la commande `imread` d'imaegio.

Première étape : Toujours avec Imageio, on récupère la taille de l'image avec la fonction .shape. Le module Numpy sert à créer et exploiter des matrices. On l'utilise donc, couplé à la fonction `random.randint` (qui génère des entiers) pour créer le tableau aléatoire :

```
tableau_aleatoire = numpy.random.randint(2, size =taille_image)
```

Deuxième étape : On veut donc associer chaque aléa du tableau ainsi créé aux bipixels que l'on a choisi auparavant pour obtenir la clé dont la taille est le double de celle de l'image. J'ai essayé de plusieurs manières :

- Avec PIL, en créant une image de la taille de la clé, et de type 'L' soit, en noir et blanc uniquement. Je voulais modifier directement la valeur des pixels pour modifier leur couleur. Il faut d'abord récupérer les valeurs de ces pixels avec la fonction `getdata()` qui renvoie les données sous forme de liste. On parcourt alors cette liste pour en changer les valeurs. Mais, elles ne changent pas.

- Avec numpy, en créant la clé en une seule fonction. Celle-ci parcourt le tableau aléatoire directement ligne par ligne, puis dimension par dimension puis valeur par valeur. Ensuite à chaque valeur, on affecte à chaque valeur d'aléa une liste qui contient les valeurs du bipixels de la clé 0 pour blanc et 1 pour noir. Mais, je me suis rendue compte que les valeurs 0 et 1 que j'affectais comme valeurs aux pixels n'étaient pas les bonnes car elles correspondent à du noir et du gris très foncé. J'ai donc modifié mon code en remplaçant le 0 par 255 qui donne

un pixel blanc et le 1 par 0 qui donne un pixel noir. Cependant, en utilisant l'outil numpy.size qui donne le nombre d'éléments de la matrice, celui-ci n'a pas changé alors qu'il devrait être doublé.

- En créant la clé d'abord, en tant que tableau aléatoire de la taille de la clé finale, puis en la modifiant. Je voulais la parcourir colonne par colonne, puis si la colonne est paire, on la parcourt ligne par ligne. Ensuite, sur chaque valeur, on regarde dans le tableau aléatoire quelle est la valeur de l'aléa et on affecte le bipixel correspondant dans la clé, sur la colonne paire et sur l'impaire qui la suit. Néanmoins, je n'ai pas trouvé comment parcourir une colonne sur deux sur ce tableau.

Algorithme de binarisation

Principe

Binariser une image, c'est la transformer en une autre image mais qui ne contient que des pixels noirs et des pixels blancs. La segmentation d'une image rassemble les pixels entre eux selon des critères définis. La binarisation d'une image correspond donc à une segmentation dont les pixels sont regroupés en seulement deux groupes : noirs et blancs. Le critère, pour binariser une image, c'est un seuil en dessous duquel les pixels deviennent blancs et au dessus duquel ils deviennent noirs.

Cahier des charges de l'interface

Fonctionnalités (dans l'ordre de l'utilisation par l'utilisateur) :

- indiquer le principe de chacune des méthodes (avec peut-être une bulle qui s'affiche quand on clique - passe la souris sur le nom de chaque méthode)
- pouvoir choisir entre différentes méthodes de codage d'une image CRYPTOGRAPHIE ou STEGANOGRAPHIE
- choisir CODAGE ou DECODAGE après avoir choisi la méthode et adapter les ajouts d'images en fonction
- pouvoir charger des images depuis son ordinateur ou rentrer un lien (ou autre, à voir sur d'autres sites) mais peut-être cela dépend-t-il de la partie codage ?
- proposer d'ajouter ses clés ou de les choisir (après) pour la cryptographie
- afficher les deux images (ou une image et un texte), la secrète et la conteneur avant la conversion puis un bouton CONVERTIR selon la méthode choisie au préalable
- choisir CODAGE ou DECODAGE après avoir choisi la méthode et adapter les ajouts d'images en fonction
- être utilisable sur le site l'Informatique c'est Fantastique !

Conclusion

Ce projet d'Informatique et Sciences du Numérique m'a permis d'approfondir les connaissances acquises durant l'année sur python et la structure d'un algorithme. J'ai aussi expérimenté de nouveaux acquis, sur python également, mais aussi sur les différentes bibliothèques et le traitement des images. De plus, cet approfondissement laisse apercevoir les possibilités autorisées par la programmation. Par ailleurs, le travail en groupe permet l'entraide mutuelle et grâce à l'échange, il est plus facile de résoudre des erreurs ou d'apporter des améliorations.

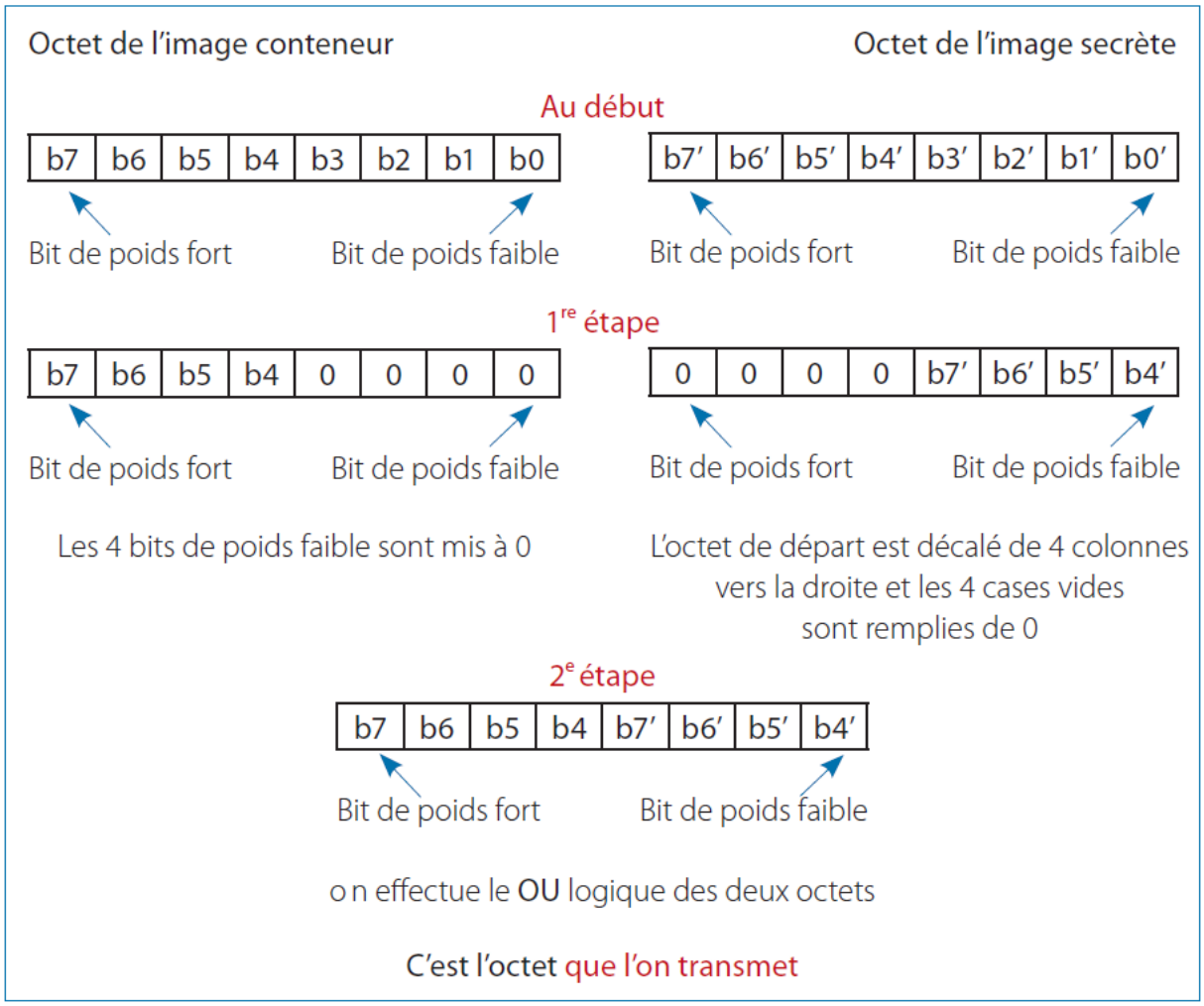
La mise en œuvre du projet m'a permis de découvrir comment peuvent être codées des images grâce à l'informatique et au codage. Les recherches préalables ont renforcé ma compréhension de l'importance et de l'omniprésence de la cryptologie aujourd'hui. En effet, avec internet, elle est nécessaire à la protection de nos données personnelles, bancaires notamment mais aussi de notre vie privée. Les données confidentielles des Etats et des entreprises doivent également être cryptées afin d'être préservées. La cryptologie doit ainsi garantir l'authenticité de l'émetteur pour le destinataire, la confidentialité et l'intégrité du message.

Sources

Le document ressource décrivant stéganographie et cryptographie :
http://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/techniques/6777/6777-190-p56.pdf

Annexes

Figure 1



Codage Stéganographie2.py

```python
##Codage
import imageio
import matplotlib.pyplot as plt

print("Pour saisir les adresses url des images à coder, il faut les mettre entre guillemets ou apostrophes")
image_1 = input("Copier l'adresse url de votre image conteneur :")
image_2 = input("Copier l'adresse url de votre image secrete :")
image_conteneur = imageio.imread(image_1)
image_secrete = imageio.imread(image_2)

def codage_image(image_conteneur, image_secrete):
    print("Plus le nombre de bits de poids faibles utilisés pour cacher l'image secrète est faible, mieux l'image est cachée mais plus elle en perd en qualité")
    choix_nombre = int(input("Choisissez ce nombre entre 1 et 7 :"))
    dico = { 1 : 0b11111110, 2 : 0b11111100, 3 : 0b11111000, 4 : 0b11110000, 5: 0b11100000, 6 : 0b11000000, 7 : 0b10000000 }
    nombre = 8 - choix_nombre
    decalage = dico[choix_nombre]
    largeur = list(range(list(image_conteneur.shape)[0])) #pour obtenir une liste contenant le même nombre que la valeur de la largeur
    hauteur = list(range(list(image_conteneur.shape)[1])) #idem pour la hauteur

    for pixel_l in largeur: #sur chaque ligne
        for pixel_h in hauteur: #sur chaque pixel de cette ligne
            #fonction codage_pixel :
            pixel_c = list(image_conteneur[pixel_l][pixel_h])  #pour transformer les tuples des valeurs des pixels en listes modifiables
            pixel_s = list(image_secrete[pixel_l][pixel_h])
            pixel_conteneur= [] #pixel conteneur une fois codé avec les bits de poids faible à 0
            pixel_secret=[]  #pixel secret une fois codé avec les bits de poids forts décalés à gauche
            for octet in pixel_c:  #pour coder chaque octet du pixel conteneur
                pixel_conteneur.append(decalage & octet) #mettre a zero le bit de poids faible
            for octet in pixel_s:   #pour coder chaque octet du pixel secret
                pixel_secret.append(octet>>choix_nombre)  #décalage du premier octet de quatre colonnes vers la droite
            #etape 1 du codage finie
            pixel_code = list(range(len(pixel_conteneur)))
            for octet in pixel_code:  #etape 2 du codage :
                pixel_code[octet] = pixel_conteneur[octet] | pixel_secret[octet] #OU logique des deux octets
            image_conteneur[pixel_l][pixel_h] = tuple(pixel_code)
    plt.show(image_conteneur)


if image_conteneur.shape != image_secrete.shape: # on verifie que les deux images font la même taille
    print("Les deux images doivent avoir les mêmes dimensions")
else:
    codage_image(image_conteneur, image_secrete)
```

Figure 2

Figure 3

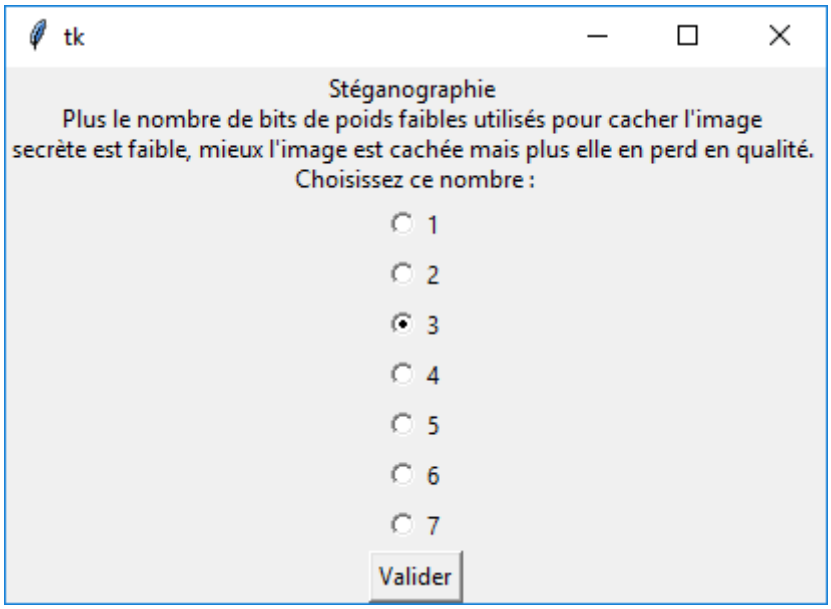


Figure 5

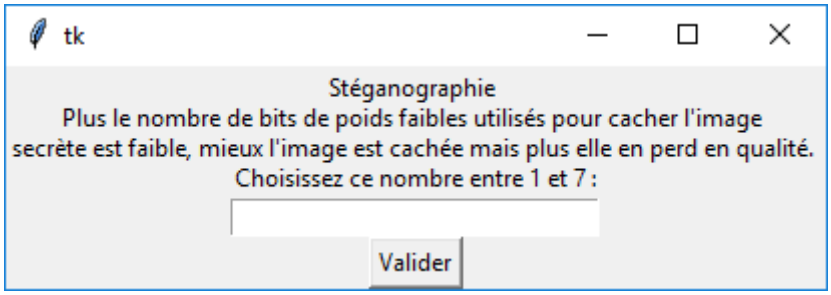


Figure 4

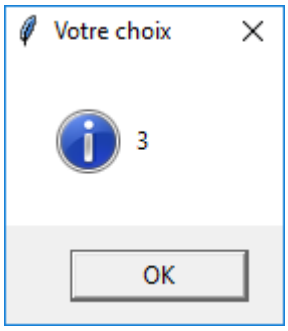


Figure 6

Choix nombre 2.py

```
from tkinter import *
from tkinter.messagebox import showinfo

fenetre = Tk()
def recupere():
    showinfo("Votre choix", entree.get())

value = StringVar()
value.set("")
note = Label(fenetre, text="Stéganographie \nPlus le nombre de bits de poids faibles utilisés pour cacher l'image \nsecrète est faible, mieux l'image est cachée mais plus elle en perd en qualité. \nChoisissez ce nombre entre 1 et 7 :")
note.pack()
entree = Entry(fenetre, textvariable=value, width=30)
entree.pack()
bouton = Button(fenetre, text="Valider", command=recupere)
bouton.pack()
fenetre.mainloop()
```