Numéro d’inscription: 14085

La stéganographie digitale

L’art de la dissimulation

Le plan

01 **Motivation et cadre théorique**

A quoi sert la stéganographie digitale ? Comment représenter une image ?

02 **La méthode de la LSB**

Implémentation et résultats de la méthode spatiale

03 **La méthode de la TCD**

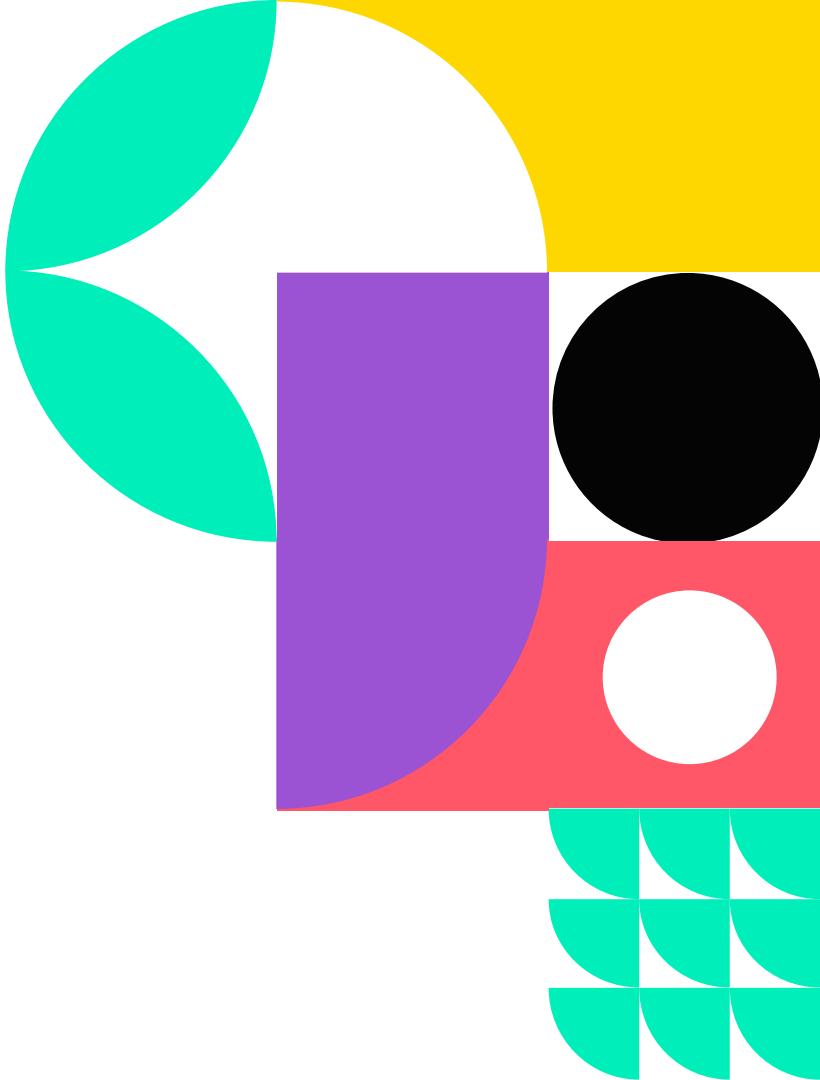
On peut représenter l'image autrement dans le domaine fréquentiel.

04 **CONCLUSION**

Les défis de la stéganographie

01

Motivation et cadre théorique



L’image couverture

L’image Stego

“Si ton œil était plus aigu tu verrais tout en mouvement. Nietzsche”

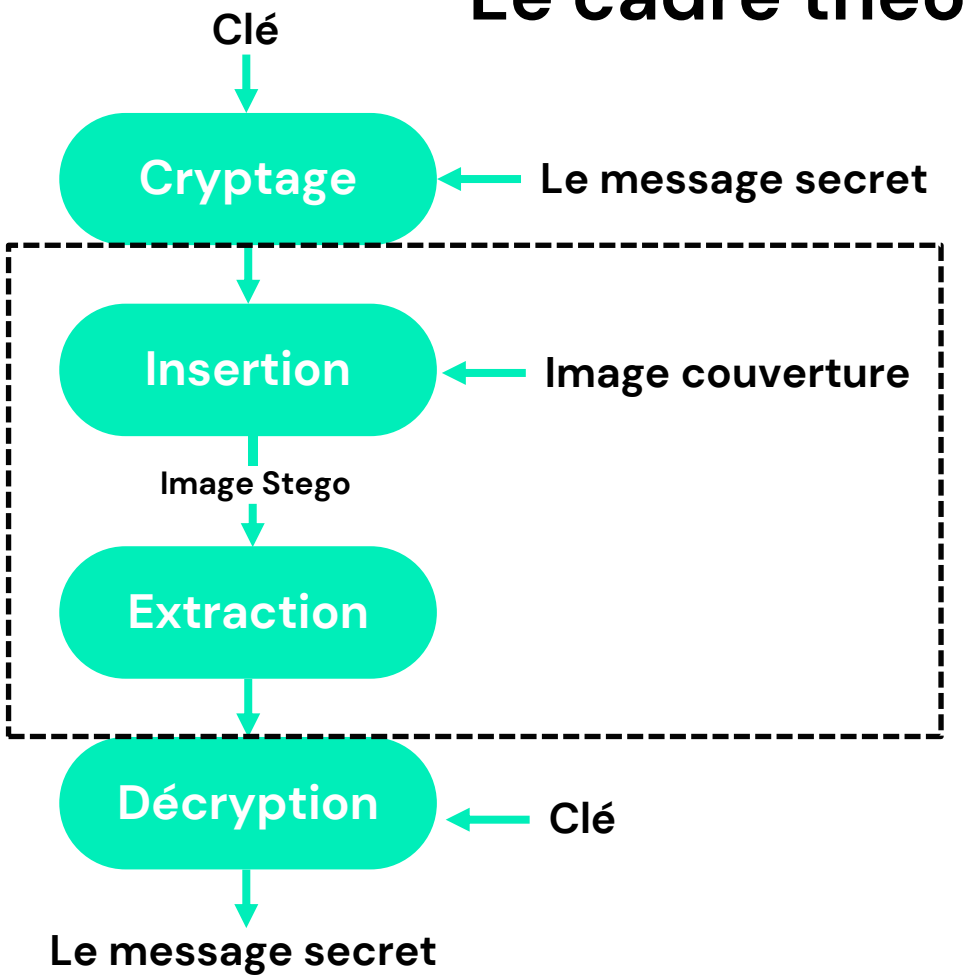
Le message caché

L’information peut être cachée dans n’importe quel media. L’étude sera menée sur l’image digitale.

problématique

Quel modèle retenir donc pour représenter l'image ? comment exploiter ce modèle pour cacher un message secret ?

Le cadre théorique

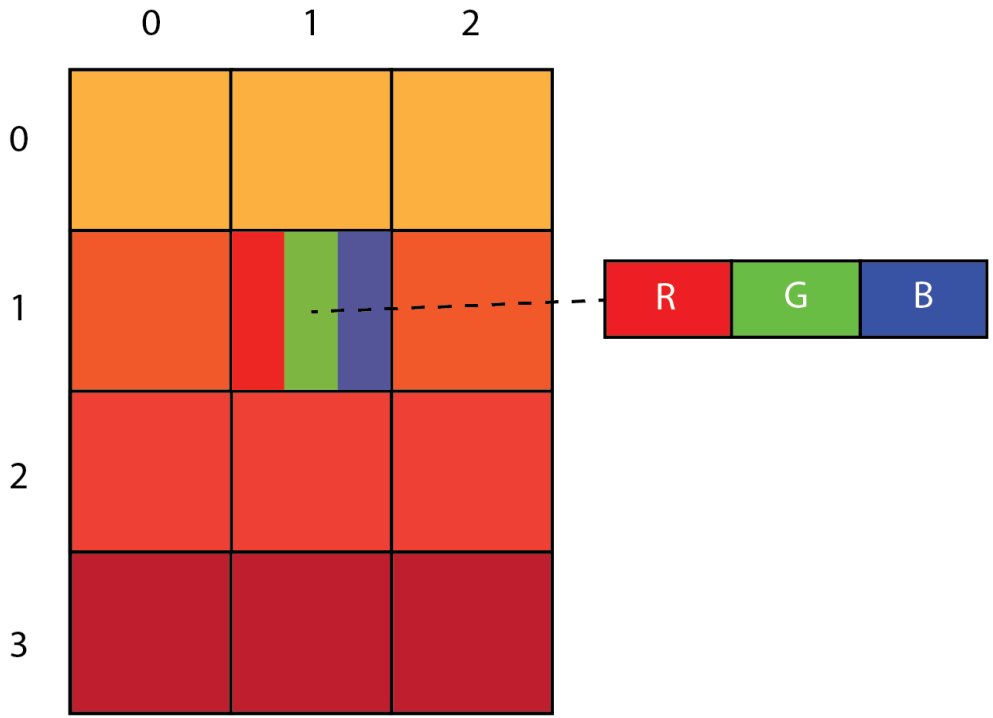


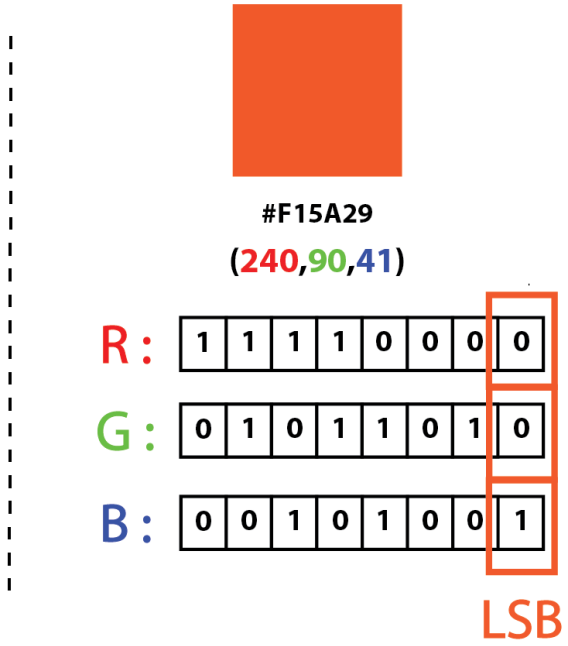
La stéganographie:

- $S: I \times M \rightarrow I$
- $S^{-1}: I \rightarrow M$

S : La fonction d'insertion
I : L'espace des images
M : L'espace des messages

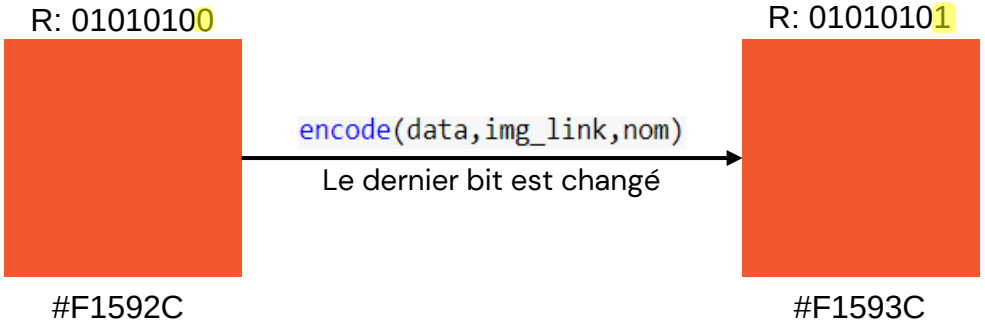
La représentation spatiale (RGB) et le principe de la LSB



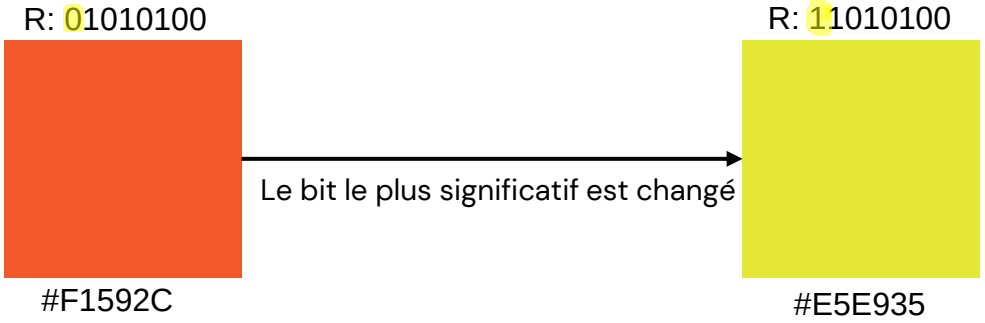


Le message est inséré dans cette partie.
L'information est découpée et convertie en un nombre binaire.

Le principe

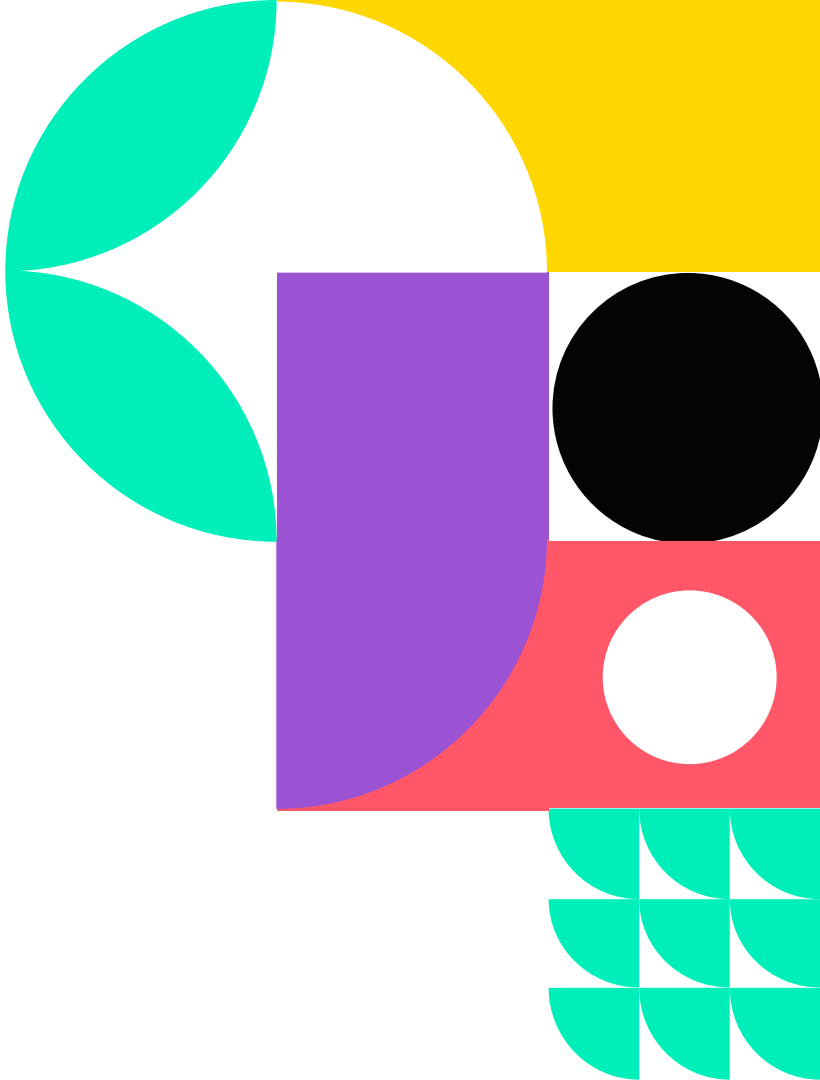


Les bits les moins significatifs sont redondants.
On peut les changer par notre message.

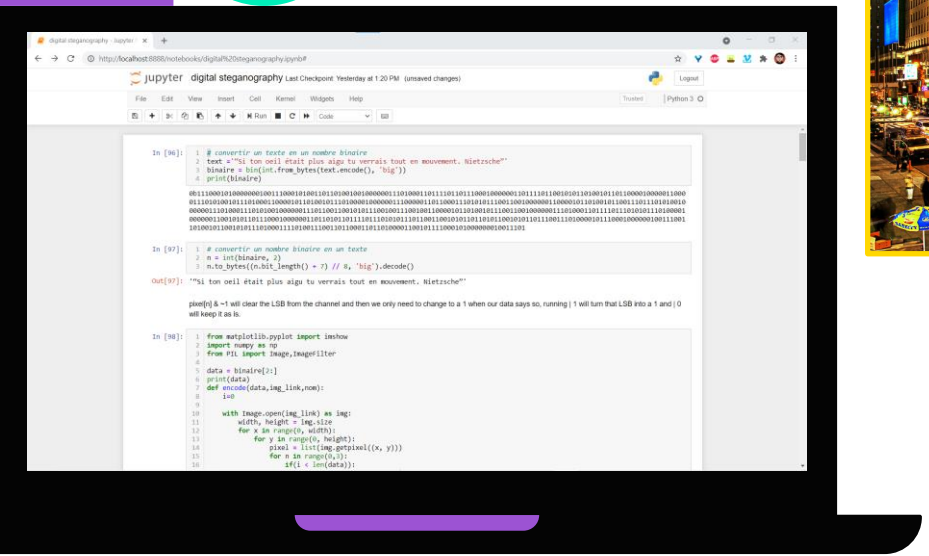


02

La méthode spatiale (La LSB)



Implémentation en python et résultats



L'objet de couverture

L'image à cacher

fusionner `merge(img1, img2, output)`

La complexité temporelle: $O(n \times m)$
n : la hauteur de l'image à cacher
m : la largeur de l'image à cacher

Résultat de la LSB

L'image extraite

Dégradation de la qualité

L’image Stego


```
unmerge(img, output)
```

défusionner

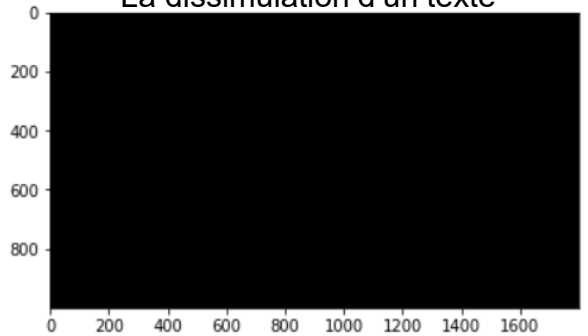
Comparaison

```
difference = np.asarray(original) - np.asarray(modified)
```

L'image originale

L'image Stego

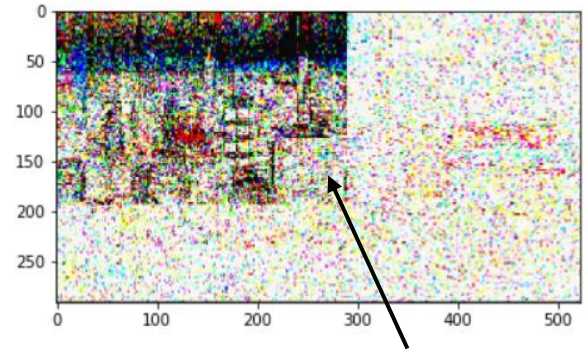
La dissimulation d'un texte



L'image originale

L'image Stego

La dissimulation d'une image



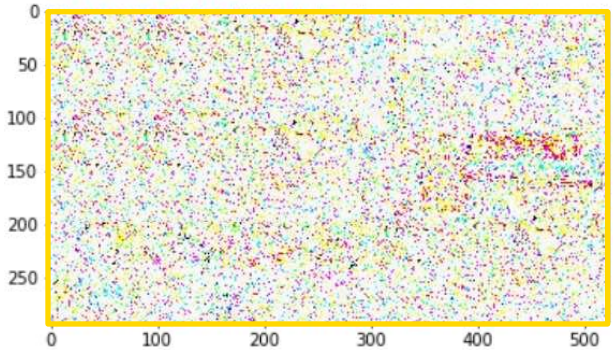
Proposition pour améliorer la sécurité

- Les approches de la stéganographie sont développées récemment, ainsi leur détection. C'est la Steganalysis.
- Parmi les méthodes élémentaires de la Steganalysis est la différence, on compare l'image suspecte avec l'image la plus ancienne.

Solution

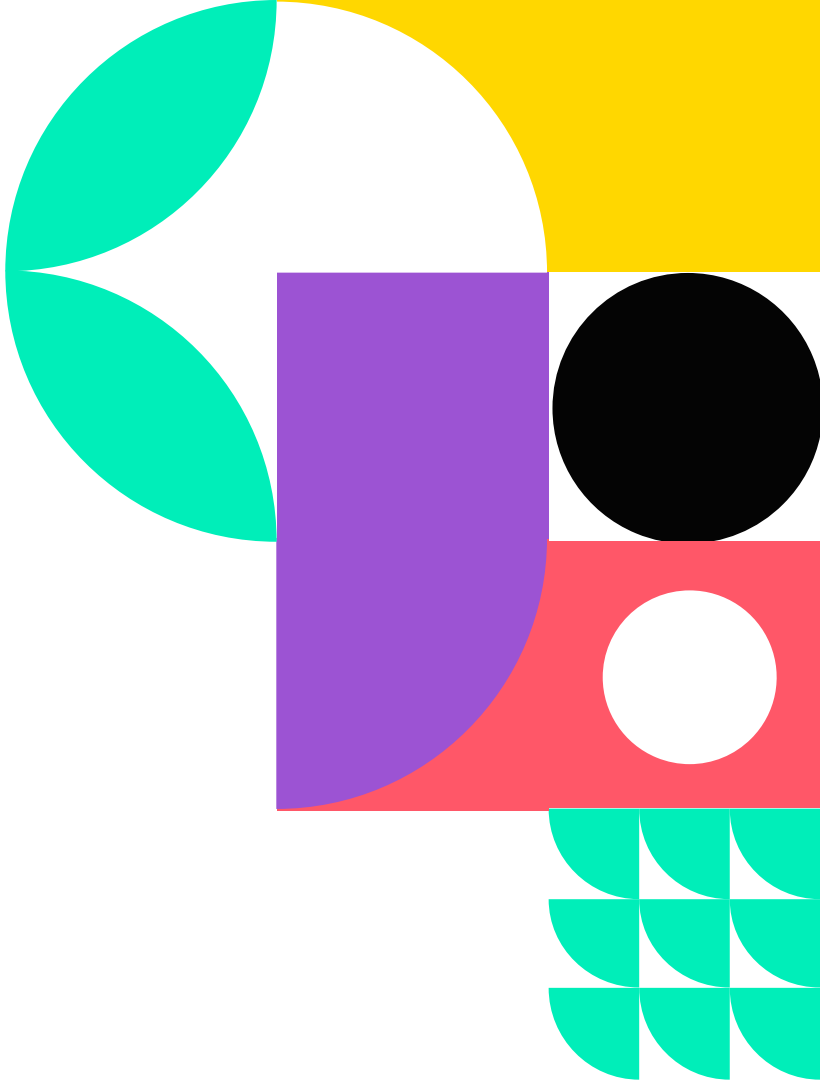
- L'image couverture originale ne doit pas être publiée.
- Le message doit être inséré dans un désordre. Cette méthode est très sensible.
- L'ensemble des positions de l'insertion est enregistré pour défusionner l'image.

```
difference = np.asarray(original) - np.asarray(modified)
```



03

La méthode fréquentielle (La TCD)



La représentation fréquentielle

Image bloc de 8×8 pixels

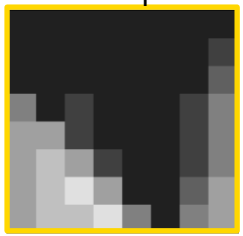
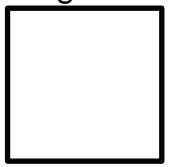
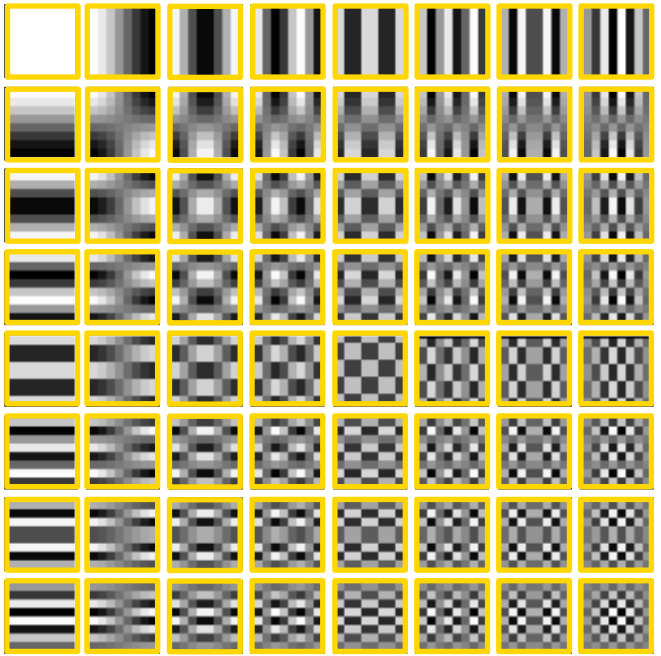
$$I = \text{  } = \sum_{1 \le i,j \le 8} C_{ij} \times \text{  }_{ij} \in$$

Image base

Images bases avec les différentes fréquences



La fonction de la transformée (TCD)

$$T : M_8(\mathbb{N}) \to M_8(\mathbb{R})$$
$$I \mapsto C$$

- Représentation spatiale: On stocke les valeurs des bits I_{ij} dans une matrice.
- Représentation fréquentielle: On stocke les coefficients C_{ij} de chaque image base dans une matrice.

Comment exploiter cette **représentation** pour cacher un message ?

Pour récupérer l'image $0(N^4)$

$$T: I_{mn} = \sum_{i,j} C_{ij} \cos \pi \frac{(2m+1)i}{16} \cdot \cos \pi \frac{(2n+1)j}{16}$$

Pour calculer les coefficients $0(N^4)$

$$T^{-1}: C_{mn} = \sum_{i,j} I_{ij} \cos \pi \frac{(2i+1)m}{16} \cdot \cos \pi \frac{(2j+1)n}{16}$$

Implémentation python

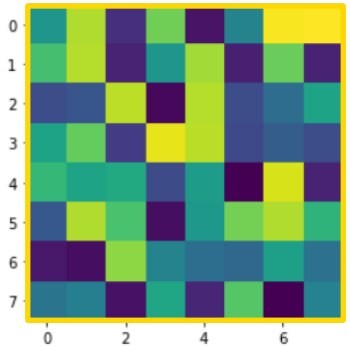
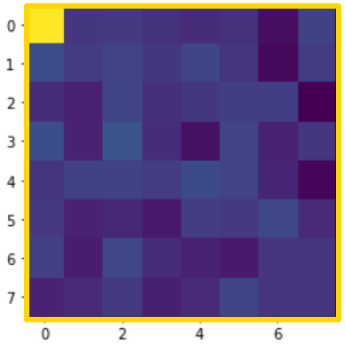


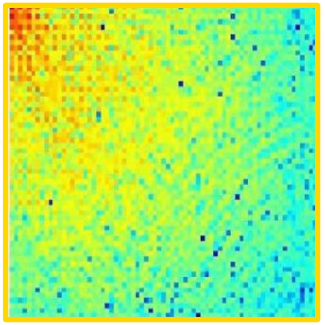
Image block I

`dct(imgray)`

$T:$



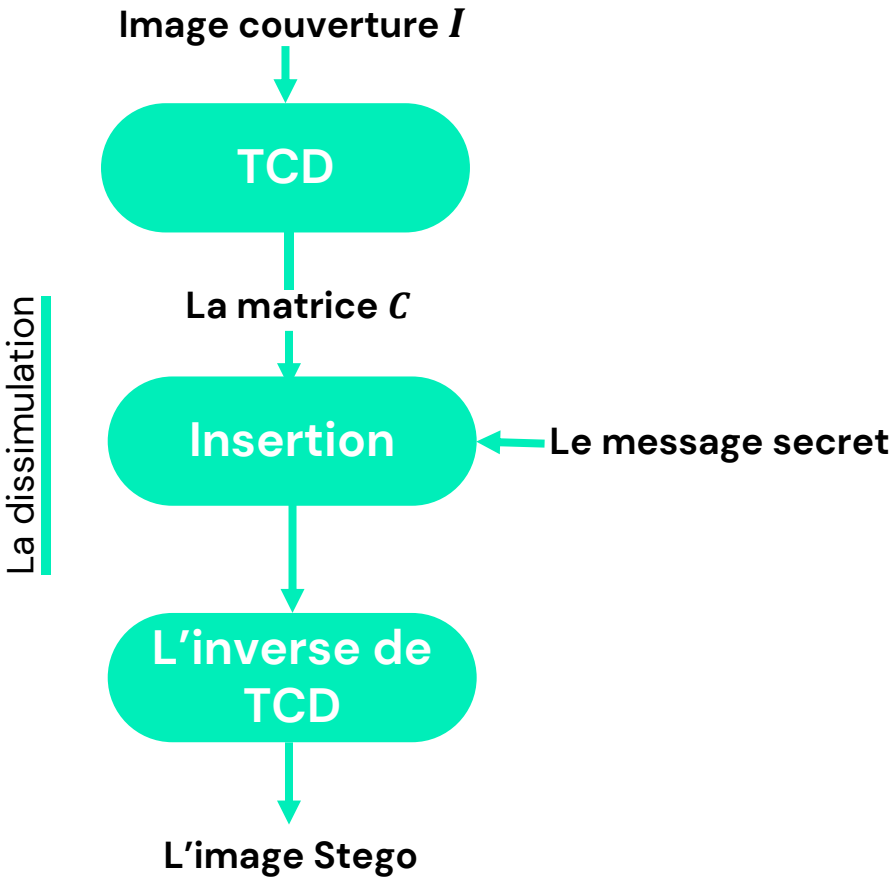
Matrice des coefficients C

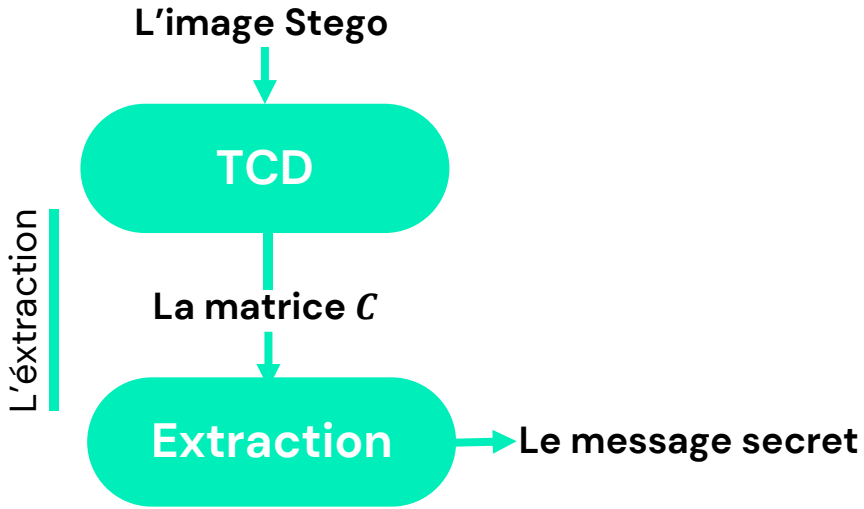


La matrice C

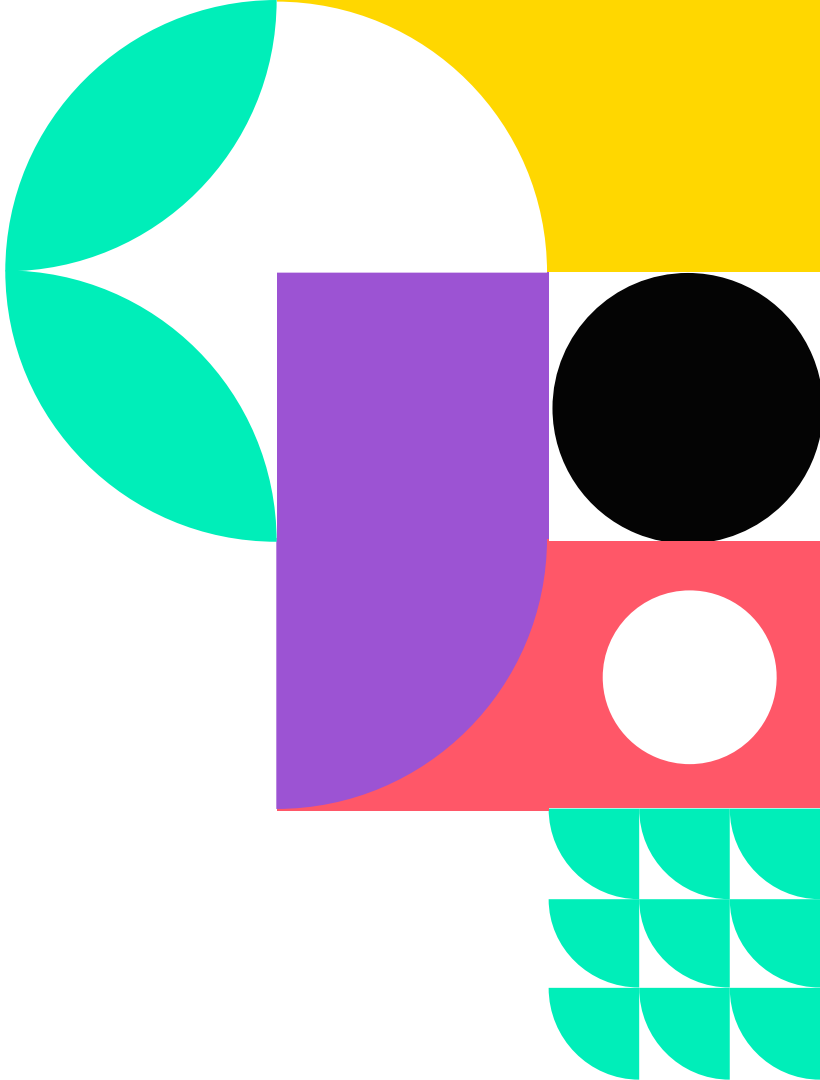
- Les faibles fréquences sont les plus significatives, ils sont en rouge.
- Les hautes fréquences sont moins significatives, on peut les supprimer. En fait c'est le principe de la compression(filtre passe-bas). Car notre œil n'est sensible qu'à des faibles fréquences.
- Plutôt on insère le message dans les hautes fréquences.
- On peut appliquer la méthode LSB sur la matrice C.

Principe de la stéganographie (TCD)





04
Conclusion

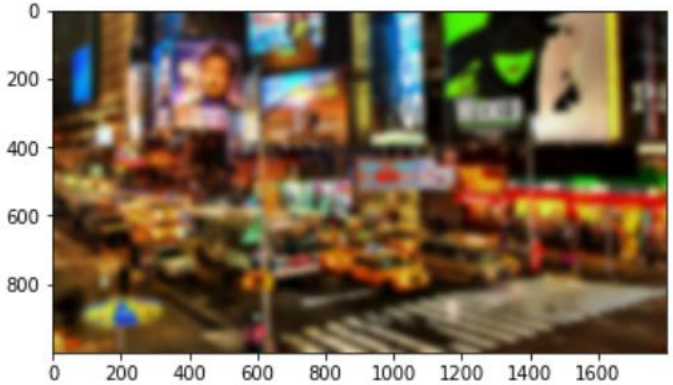


Résultats:

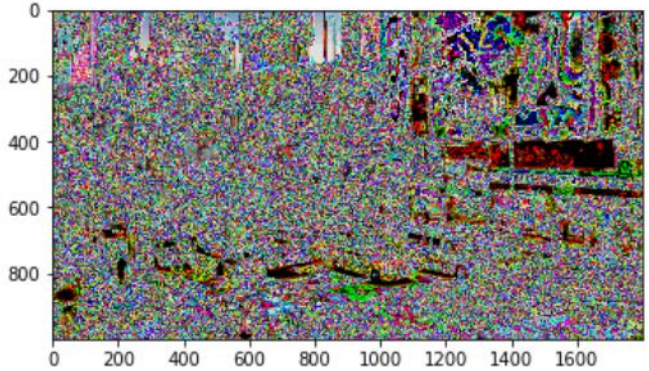
Le message n'est pas récupéré. Je trouve que les coefficients changent, cette méthode est très sensible. Néanmoins, une version puissante est utilisée dans ce domaine.

Les défis de la stéganographie:

L'image est sensible au bruit, et aux différentes transformations. Un simple changement altère le message.



Test de la robustesse



L'image cachée est perdue

La capacité de stockage: nous pouvons en théorie stocker $L \times H \times 3/8$ octets de données (1ére méthode)

ANNEXE

In [ ]:
```
from matplotlib.pyplot import imshow
import numpy as np
from PIL import Image,ImageFilter

data = binaire[2:]
def encode(data,img_link,nom):
    i=0

    with Image.open(img_link) as img:
        width, height = img.size
        for x in range(0, width):
            for y in range(0, height):
                pixel = list(img.getpixel((x, y)))
                for n in range(0,3):
                    if(i < len(data)):
                        pixel[n] = pixel[n] & ~1 | int(data[i])
                        i+=1
                img.putpixel((x,y), tuple(pixel))
        img.save(nom, "PNG")
encode(data,"image_couverture.jpg","source_secret.png")
originel = Image.open("image_couverture.jpg")
modified = Image.open("source_secret.png")
```

In [ ]:
```
def decode(secret_img_link,l):
    ''' l = la longueur de data'''
    extracted_bin = []
    with Image.open(secret_img_link) as img:
        width, height = img.size
        for x in range(0, width):
            for y in range(0, height):
                pixel = list(img.getpixel((x, y)))
                for n in range(0,3):
                    extracted_bin.append(pixel[n] & 1)
    extracted = "".join([str(x) for x in extracted_bin[:l]])
    return extracted

code = '0b'+decode("source_secret.png",len(data))
n = int(code, 2)
n.to_bytes((n.bit_length() + 7) // 8, 'big').decode()
```

In [ ]:
```
from PIL import Image

class Steganography(object):
    @staticmethod
    def __int_to_bin(rgb):
        """convertir un integer tuple en  binary (string) tuple.

        :param rgb: un  integer tuple (e.g. (220, 110, 96))
        :return: A string tuple (e.g. ("00101010", "11101011", "00010110"))
        """
        r, g, b = rgb
        return ('{0:08b}'.format(r),
                '{0:08b}'.format(g),
                '{0:08b}'.format(b))
    @staticmethod
    def __bin_to_int(rgb):
        """convertir  un nombre binaire (string) tuple en un integer tuple.

        :param rgb: A string tuple (e.g. ("00101010", "11101011", "00010110"))
        :return: Return an int tuple (e.g. (220, 110, 96))
        """
        r, g, b = rgb
        return (int(r, 2),
                int(g, 2),
                int(b, 2))
    @staticmethod
    def __merge_rgb(rgb1, rgb2):
        """fusionner les deux tuples RGB .

        :param rgb1: A string tuple (e.g. ("00101010", "11101011", "00010110"))
        :param rgb2: Another string tuple
        (e.g. ("00101010", "11101011", "00010110"))
        :return: An integer tuple with the two RGB values merged.
        """

        r1, g1, b1 = rgb1
        r2, g2, b2 = rgb2
        rgb = (r1[:4] + r2[:4],
               g1[:4] + g2[:4],
               b1[:4] + b2[:4])
        return rgb
    @staticmethod
    def merge(img1, img2):
        """fusionner les deux images . la deuxième sera insérée dans la première.


        :param img1: 1er image
        :param img2: 2eme image
        :return: le fichier Stego(Image).
        """

        # Vérifiez les dimensions des images
        if img2.size[0] > img1.size[0] or img2.size[1] > img1.size[1]:
            raise ValueError("L'image 2 ne doit pas être plus grande que l'image 1 !")

        # Obtenir la carte des pixels des deux images
        pixel_map1 = img1.load()
        pixel_map2 = img2.load()

        # Créez une nouvelle image qui sera éditée
        new_image = Image.new(img1.mode, img1.size)
        pixels_new = new_image.load()

        for i in range(img1.size[0]):
            for j in range(img1.size[1]):
                rgb1 = Steganography.__int_to_bin(pixel_map1[i, j])

                # Utiliser un pixel noir par défaut

                rgb2 = Steganography.__int_to_bin((0, 0, 0))

                # Vérifiez si la position de la carte de pixels est valide pour la deuxième
 image.
                if i < img2.size[0] and j < img2.size[1]:
                    rgb2 = Steganography.__int_to_bin(pixel_map2[i, j])

                # Fusionner les deux pixels et les convertir en un tuple d'entiers.
                rgb = Steganography.__merge_rgb(rgb1, rgb2)

                pixels_new[i, j] = Steganography.__bin_to_int(rgb)

        return new_image
    @staticmethod
    def unmerge(img):
        """Défusionner une image.

        :param img: The input image.
        :return: L'image extraitée.
        """

        # Charger la carte de pixels
        pixel_map = img.load()

        # Créez la nouvelle image et chargez la carte des pixels.
        new_image = Image.new(img.mode, img.size)
        pixels_new = new_image.load()

        # Tuple utilisé pour stocker la taille originale de l'image
        original_size = img.size

        for i in range(img.size[0]):
            for j in range(img.size[1]):
                # Obtenir le RGB (sous forme de tuple de chaîne) du pixel actuel
                r, g, b = Steganography.__int_to_bin(pixel_map[i, j])

                # Extraire les 4 derniers bits (correspondant à l'image cachée)
                # Concaténer 4 bits de zéro parce que nous travaillons avec 8 bits.
                rgb = (r[4:] + '0000',
                       g[4:] + '0000',
                       b[4:] + '0000')

                # Convertissez-le en un tuple d'entiers
                pixels_new[i, j] = Steganography.__bin_to_int(rgb)

                #S'il s'agit d'une position "valide",
                #enregistrez-la comme la dernière position valide.

                if pixels_new[i, j] != (0, 0, 0):
                    original_size = (i + 1, j + 1)

        # Recadrer l'image sur la base des pixels "valides".
        new_image = new_image.crop((0, 0, original_size[0], original_size[1]))

        return new_image




def merge(img1, img2, output):
    merged_image = Steganography.merge(Image.open(img1), Image.open(img2))
    merged_image.save(output)
img1="image_couverture.jpg"
img2='img2.jpg'
merge(img1,img2,'secret.png')


original= Image.open("image_couverture.jpg")
modified = Image.open("secret.png")
%matplotlib inline

imshow(np.asarray(original))


def unmerge(img, output):
    unmerged_image = Steganography.unmerge(Image.open(img))
    unmerged_image.save(output)
```

In [ ]:
```
#TCD
M = N = 8
def dct(imgray):
    B=np.zeros([8,8])
    [M,N]=[len(imgray),len(imgray[0])]
    print(M,N)
    for i in range(0,M):
        for j in range(0,N):
            if i ==0:
                AlphaP = sqrt(1/M)
            else:
                AlphaP = sqrt(2/M)
            if j==0:
                AlphaQ = sqrt(1/N)
            else:
                AlphaQ = sqrt(2/N)
            temp = 0
            for x in range(0,M):
                for y in range(0,N):
                    cs1 = cos(pi * (2*x+1)*i/(2*M))
                    cs2 = cos(pi * (2*y+1)*j/(2*N))
                    temp += imgray[x,y] * cs1 * cs2
            B[i,j] = AlphaP*AlphaQ*temp
    return B
```

In [ ]:
```
#inverse dct or idct
def idct(coeffs):
    imgray = np.zeros([8,8])
    M,N = 8,8
    for m in  range(0,M):
        for n in range(0,N):
            temp = 0
            for i in range(0,M):
                for j in range(0,N):
                    if i == 0 :
                        AlphaP = sqrt(1/M)
                    else:
                        AlphaP = sqrt(2/M)
                    if j == 0:
                        AlphaQ = sqrt(1/N)
                    else:
                        AlphaQ = sqrt(2/N)

                    cs1 = cos(pi * (2*m+1)*i/(2*M))
                    cs2 = cos(pi * (2*n+1)*j/(2*N))
                    temp += AlphaP * AlphaQ * cs1 *cs2 *coeffs[i,j]
            imgray[m,n] = temp
    return imgray
```