TIPE :

Cryptage / Décryptage d'images en couleurs

Objectif :

Créer une application qui:

- Prend une « image source », la crypte et renvoi une clef de 24 bits.
- Prend une « image cryptée » et une clef de 24 bits, la décrypte.

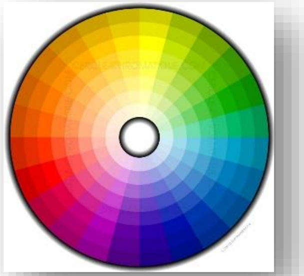


Image source

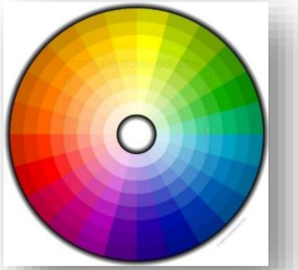


Image décryptée

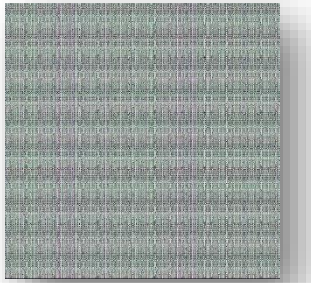
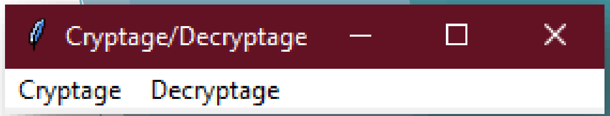
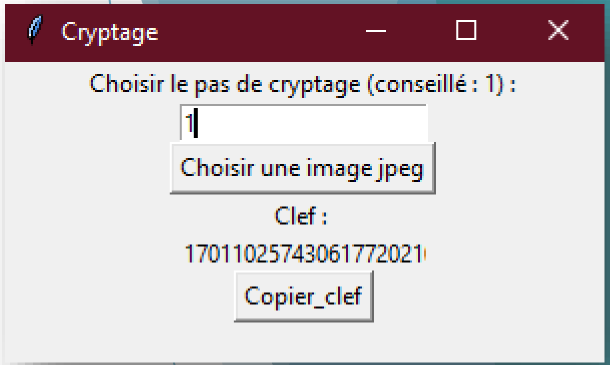
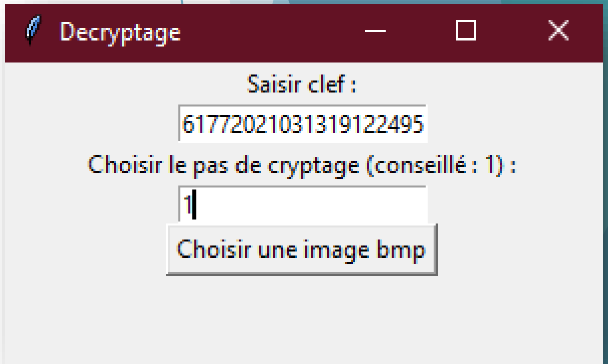


Image cryptée Méthode 1


Image cryptée Méthode 2







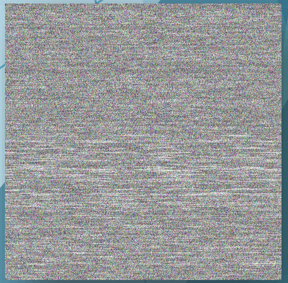


Image cryptée Méthode finale

Sommaire :

Explication du code :

1) Redimensionner « image source » où X et Y sont des nombres premiers :

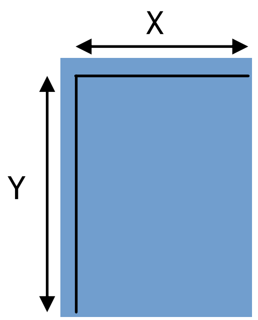
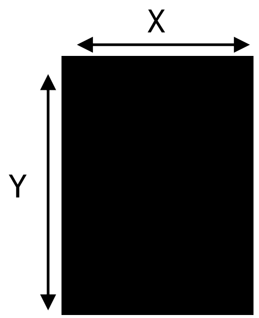
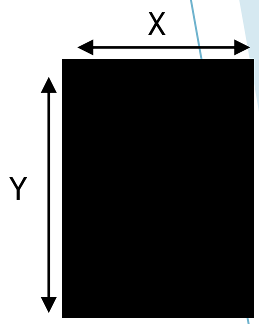


Image source

Puis créer « fond crypte », « fond décrypte » sur lesquels on collera les pixels cryptés :



Fond crypte



Fond décrypte

2) Créer la clef :

- 16 caractères générés aléatoirement
- 8 caractères qui correspondent à la date du jour
- 8 caractères qui correspondent l'heure d'envoi (jusqu'au millième de seconde)

Exemple : 1879461975364879 11032021 12582425

Remarque : Grâce à la date et à l'heure, on se déplace dans la liste des 16 premiers caractères.

Explication du code :

3) Extraire 8 coefficients :

Exemple : Clef = [1, 8, 7, 9, 4, 6, 1, 9, 7, 5, 3, 6, 4, 8, 7, 9, 11, 03, 20, 21, 12, 58, 24, 25]

Position : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

Ax = Clef[11 mod (16)] = 6
Ay = Clef[3 mod (16)] = 9

Bx = Clef[20 mod (16)] = Clef[4 mod(16)] = 4
By = Clef[21 mod (16)] = Clef[5 mod(16)] = 6

Cx = Clef[12 mod (16)] = 4
Cy = Clef[58 mod (16)] = Clef[10 mod(16)] = 3

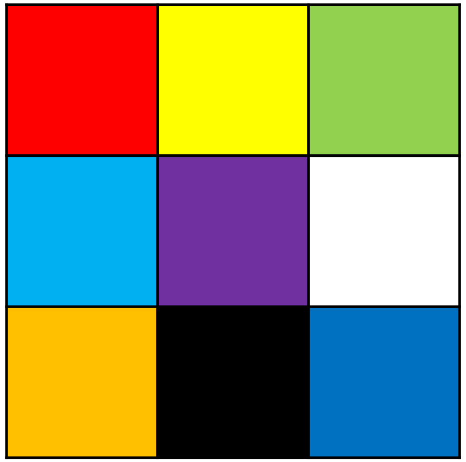
Dx = Clef[24 mod (16)] = Clef[8 mod(16)] = 7
Dy = Clef[25 mod (16)] = Clef[9 mod(16)] = 5

Explication du code :

4) Créer la matrice W (10x10) qui contient des coefficients aléatoires compris entre 1 et max(X, Y) :

<u>Exemple :</u> Sur une image 3x3 :

```
W = [[3, 1, 1, 3, 1, 2, 2, 3, 2, 2],
     [1, 1, 2, 3, 1, 3, 1, 2, 2, 3],
     [2, 1, 1, 3, 3, 2, 3, 3, 1, 2],
     [1, 1, 3, 1, 2, 3, 3, 3, 1, 2],
     [1, 2, 1, 1, 1, 3, 2, 1, 3, 2],
     [1, 1, 1, 3, 2, 1, 3, 3, 1, 1],
     [2, 3, 3, 1, 3, 2, 3, 3, 1, 2],
     [1, 3, 2, 1, 2, 3, 1, 3, 3, 2],
     [1, 1, 2, 3, 1, 3, 3, 3, 1, 2],
     [1, 2, 2, 2, 3, 3, 2, 1, 2, 2]]
```



Explication du code :

5) On prend 4 coefficients de la matrice à partir des coefficients sorti de la clef :

Exemple : Sur une image 3x3 (on restreint les coefficients pour ne pas sortir de la matrice W) :

W = [[3, 1, 1, 3, 1, 2, 2, 3, 2, 2],
[1, 1, 2, 3, 1, 3, 1, 2, 2, 3],
[2, 1, 1, 3, 3, 2, 3, 3, 1, 2],
[1, 1, 3, 1, 2, 3, 3, 3, 1, 2],
[1, 2, 1, 2, 1, 3, 2, 1, 3, 2],
[1, 1, 1, 3, 2, 1, 3, 3, 1, 1],
[2, 3, 3, 1, 3, 2, 3, 3, 1, 1],
[1, 3, 2, 1, 2, 1, 1, 3, 3, 2],
[1, 1, 2, 3, 1, 3, 3, 3, 1, 2],
[1, 2, 2, 2, 3, 3, 2, 1, 2, 2]]

Donc :

A = W[Ax][Ay] = W[6][9] = 1
B = W[Bx][By] = W[4][6] = 2
C = W[Cx][Cy] = W[4][3] = 2
D = W[Dx][Dy] = W[7][5] = 1

Explication du code :

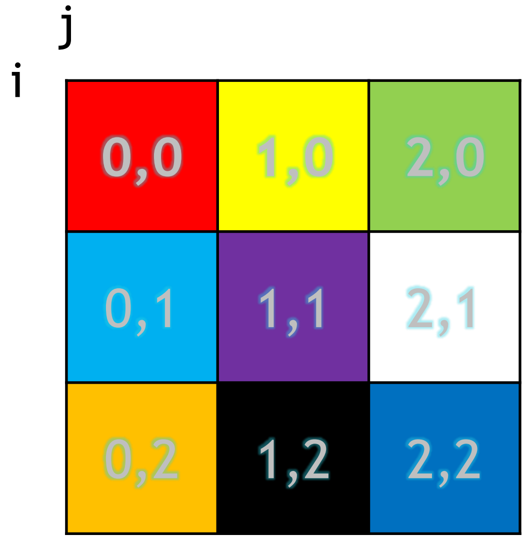
6) Cryptage affine :

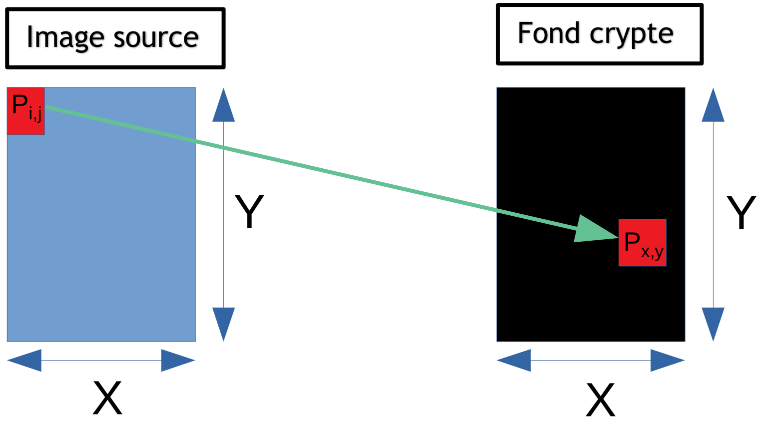
Exemple : Sur une image 3x3 :

A = 1, B = 2, C = 2, D = 1

On prend i ϵ [0, 3] et j ϵ [0, 3]:

X = (A * i + B) mod(3) = (1 * i + 2) mod(3)
Y = (C * j + D) mod(3) = (2 * j + 1) mod(3)





Remarque : le modulo sera toujours un nombre premier ce qui assure la bijectivité de notre fonction.

Explication du code :

6) Cryptage affine :

<u>Exemple :</u> On prend i ϵ [0, 3] et j ϵ [0, 3]:

X = (A * i + B) mod(3) = (1 * i + 2) mod(3)
Y = (C * j + D) mod(3) = (2 * j + 1) mod(3)

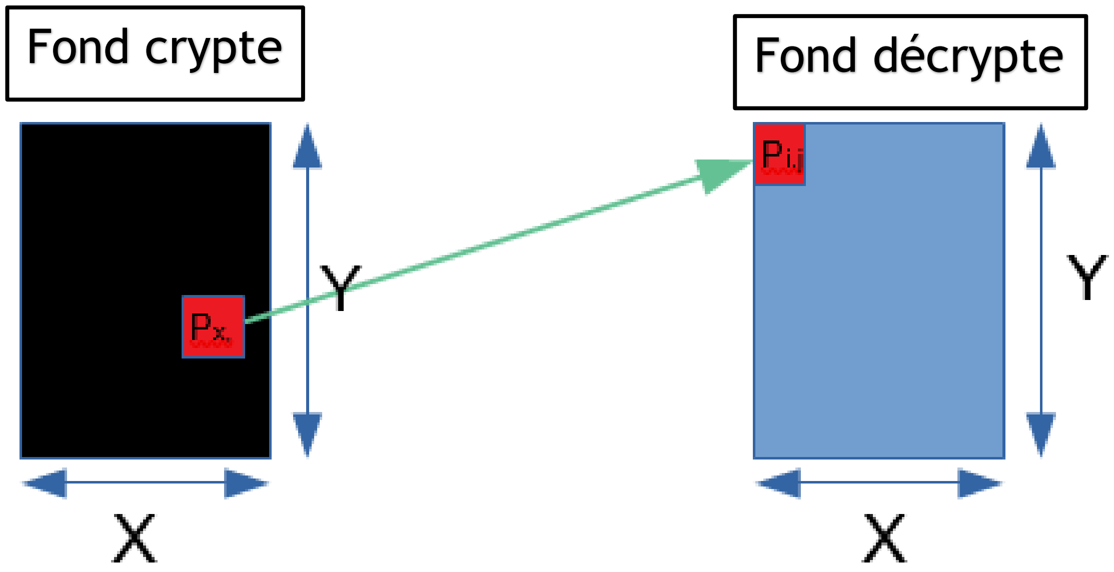
(0, 0) => ((1*0+2)mod(3), (2*0+1)mod(3)) = (2, 1)
(1, 0) => ((1*0+2)mod(3), (2*1+1)mod(3)) = (2, 0)
(2, 0) => ((1*0+2)mod(3), (2*2+1)mod(3)) = (2, 2)

(0, 1) => ((1*1+2)mod(3), (2*0+1)mod(3)) = (0, 1)
(1, 1) => ((1*1+2)mod(3), (2*1+1)mod(3)) = (0, 0)
(2, 1) => ((1*1+2)mod(3), (2*2+1)mod(3)) = (0, 2)

(0, 2) => ((1*2+2)mod(3), (2*0+1)mod(3)) = (1, 1)
(1, 2) => ((1*2+2)mod(3), (2*1+1)mod(3)) = (1, 0)
(2, 2) => ((1*2+2)mod(3), (2*2+1)mod(3)) = (1, 2)

Explication du code :

7) Décryptage affine :




```python
def cryptage(img):
    x_img, y_img = img.size

    taille_x = dimensions(y_img // pas) + 1
    taille_y = dimensions(x_img // pas) + 1

    grille_4, grille_2 = quadrillage(img)
    grille_crypte4, grille_crypte2 = zeros(taille_x, taille_y)

    for i in range(1, len(grille_2)):
        for j in range(1, len(grille_2[0])):
            x = (a * i + b) % (len(
                grille_2) - 1)  # car sinon ce n'est plus un no
            y = (c * j + d) % (len(grille_2[0]) - 1)  # idem

            grille_crypte4[x + 1][y + 1] = grille_4[i][j]
            grille_crypte2[x + 1][y + 1] = grille_2[i][j]

    return grille_crypte4, grille_crypte2
```

```python
def decryptage(img):
    x_img, y_img = img.size

    taille_x = dimensions(y_img // pas) + 1
    taille_y = dimensions(x_img // pas) + 1

    grille_crypte4, grille_crypte2 = cryptage(img)
    grille_decrypte4, grille_decrypte2 = zeros(taille_x, taille_y)

    for i in range(1, len(grille_crypte2)):
        for j in range(1, len(grille_crypte2[0])):
            x = (a * i + b) % (len(grille_crypte2) - 1)  # car sinon
            y = (c * j + d) % (len(grille_crypte2[0]) - 1)  # idem

            grille_decrypte2[i][j] = grille_crypte2[x + 1][y + 1]

    return grille_decrypte4, grille_decrypte2
```

Remarque : On peut appliquer cette méthode car on utilise une clef fixé pour le cryptage et décryptage d'image.

Comment coder ces idées ? :

Problème : Pour les images supérieurs à 3x3, i et j doivent être différents de 0 pour assurer la bijectivité de la fonction.

Or la fonction img.crop() coupe une image en prenant pour origine le coin gauche de l'image. Donc si on parcours toute l'image sous forme de tableau, i et j peuvent valoir 0.

Solution : On translate l'image tel que pour une image 11x11 (avec un pas de 1) :

```
[(0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0), (0, 0)]
[(0, 0), (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0)]
[(0, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1)]
[(0, 0), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (6, 2), (7, 2), (8, 2), (9, 2), (10, 2)]
[(0, 0), (0, 3), (1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3), (8, 3), (9, 3), (10, 3)]
[(0, 0), (0, 4), (1, 4), (2, 4), (3, 4), (4, 4), (5, 4), (6, 4), (7, 4), (8, 4), (9, 4), (10, 4)]
[(0, 0), (0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5), (7, 5), (8, 5), (9, 5), (10, 5)]
[(0, 0), (0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (7, 6), (8, 6), (9, 6), (10, 6)]
[(0, 0), (0, 7), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7), (8, 7), (9, 7), (10, 7)]
[(0, 0), (0, 8), (1, 8), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8)]
[(0, 0), (0, 9), (1, 9), (2, 9), (3, 9), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9), (10, 9)]
[(0, 0), (0, 10), (1, 10), (2, 10), (3, 10), (4, 10), (5, 10), (6, 10), (7, 10), (8, 10), (9, 10), (10, 10)]
```

Donc : la position du pixel (0,0) s'obtient maintenant en parcourant cette matrice A en faisant : A[1][1]

Première application du code :

Image source

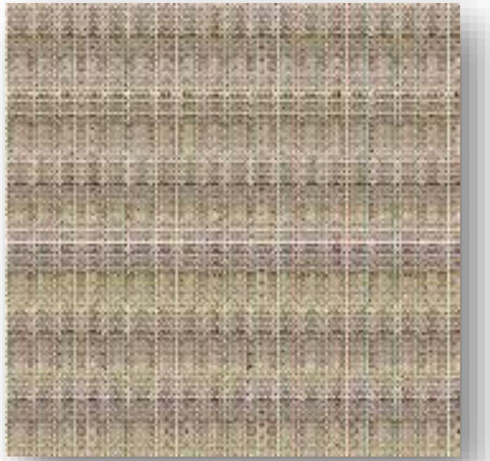


Image cryptée

Image décryptée

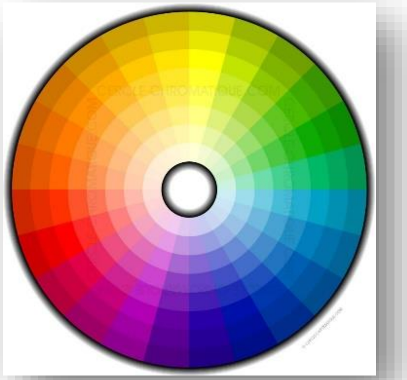
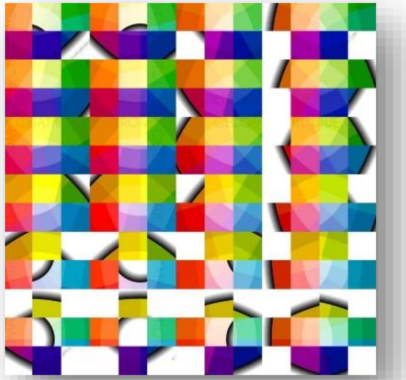
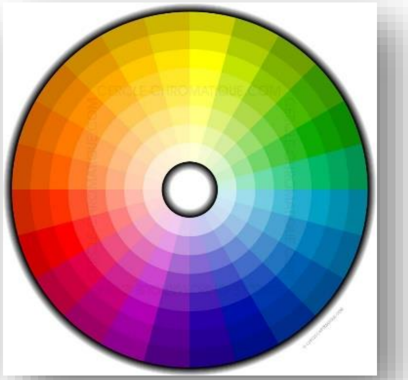
Complexité : Quadratique en O(X * Y) car on parcours l'image entière.

Remarque : On observe une sorte de motif périodique sur l'image cryptée.

Points forts du code :

- Rapide, efficace, sans perte d'information (car on redimensionne l'image, on le la coupe pas)

- Rendu visuel ludique « puzzle » pour un pas élevé on a :

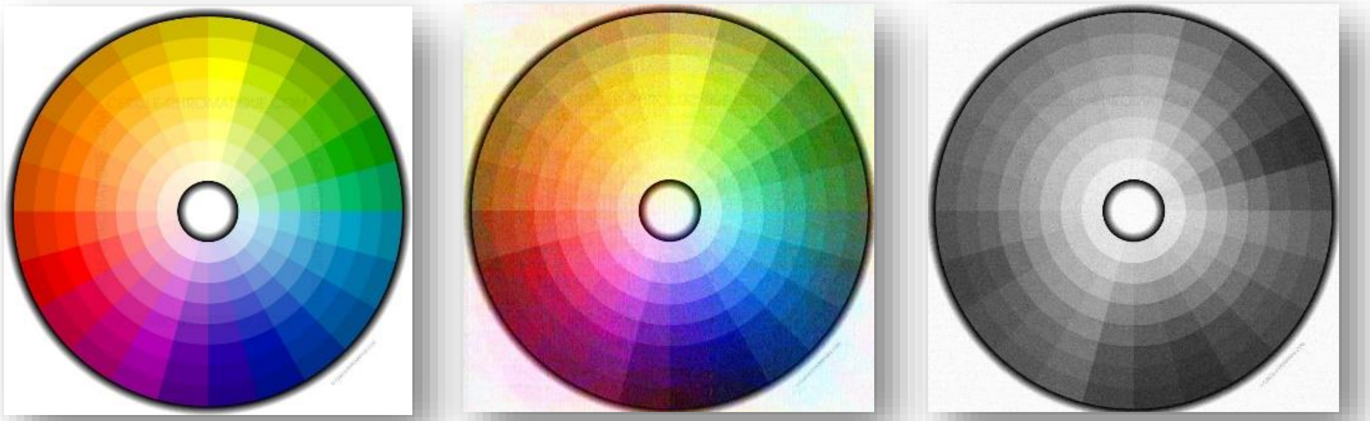
- Impossible à craquer par force brute :

Nombre d'opérations à effectuer = $10^{24} * \binom{24}{8} * \binom{100}{4} * 10^{1080} = 3.10^{1956}$

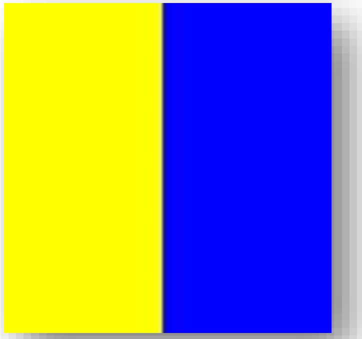
$$<=> 9.10^{1924}\ \textit{milliards d'années}$$

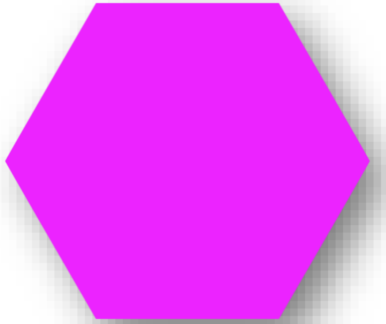
<u>Limites du code :</u>

- <u>Format :</u> BMP au lieu de JPG (qui compresse le fichier et conduit à une perte d'information) :



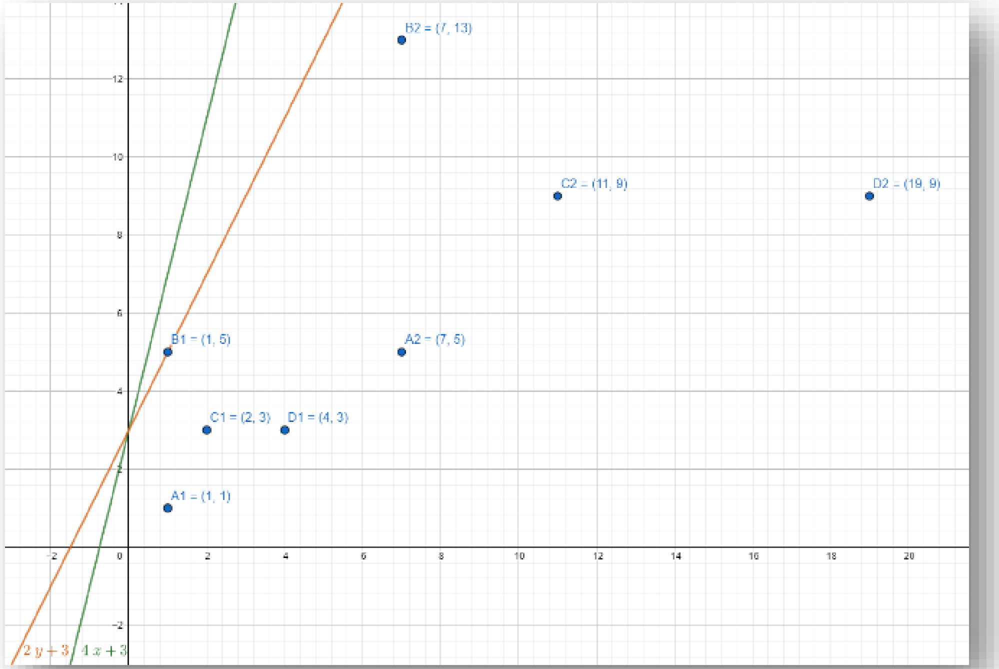
Associé à la <u>limite de la bibliothèque :</u>





Limites du code :

- Craquage par intelligence :



Ceci montre pourquoi on observe des motifs sur notre image cryptée.

Remarque : Il serait donc facile de trouver la combinaison des coefficients (a, b, c, d) si on procède sur une portion de notre image.

Solutions contre le craquage intelligent :

- Méthode récursive (sur les lignes d'une image ou X = 2k) :

Exemple : Sur une image comportant 8 pixels sur chaque lignes et Y = 5 lignes :

1	2	3	4	5	6	7	8

On divise par 2 le nombre de colonne (X) tel que :

1	2	3	4	5	6	7	8

On fait des rotations des pixels sur UNE ligne à l'intérieur suivant un nombre compris entre [1, Y] qui sont indiqués dans la clef :
Si on prend (1, 3) pour chaque groupe respectif alors on obtient :

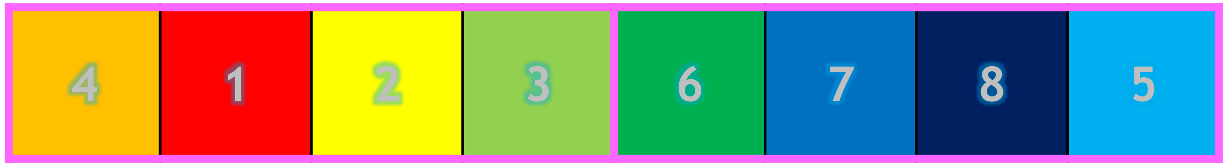
4	1	2	3	6	7	8	5

Remarque : Sur les autres lignes on pose une condition (par exemple : pair ou impair) où les coefficients sont différents.

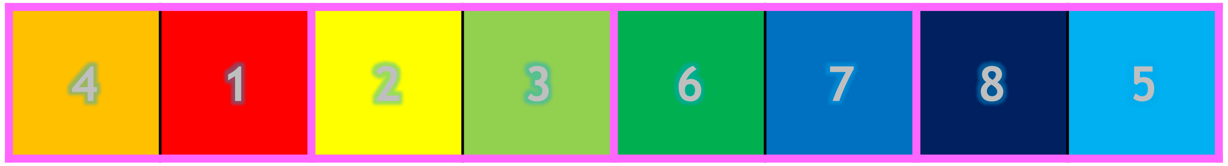
Solutions contre le craquage intelligent :

- Méthode récursive (sur les lignes d'une image ou X = 2k) :

Exemple : Sur une image comportant 8 pixels sur chaque lignes et Y = 5 lignes:

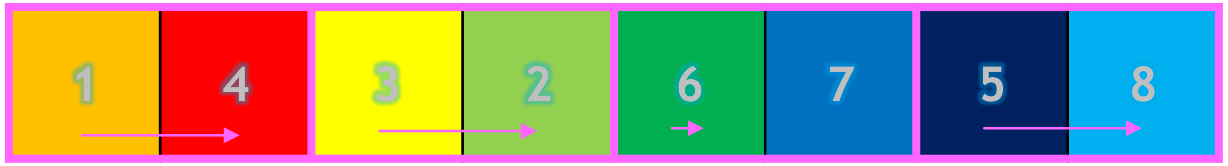


On répète cette opération (/2) jusqu'à arriver a 1 pixel (comme ça même si il y a un reste le programme tourne car il est récursif) :



Sur la même ligne on reprend : (1, 3) puis on rajoute le premier coefficient + le dernier :

$\Rightarrow$ (1, 3, 1+3, 1+3+1) = (1, 3, 4, 5)



Solutions contre le craquage intelligent :

- Méthode récursive (sur les lignes de l’image) :

Exemple : Sur une image comportant 8 pixels sur chaque lignes et Y = 5 lignes:

On arrive à : 2/2 = 1 donc les rotations sont terminés !

1	4	3	2	6	7	5	8

Donc :

1	2	3	4	5	6	7	8

1	4	3	2	6	7	5	8

Solutions contre le craquage intelligent :

- Méthode récursive + Rotation RGB (Prochaine diapo):

Image source

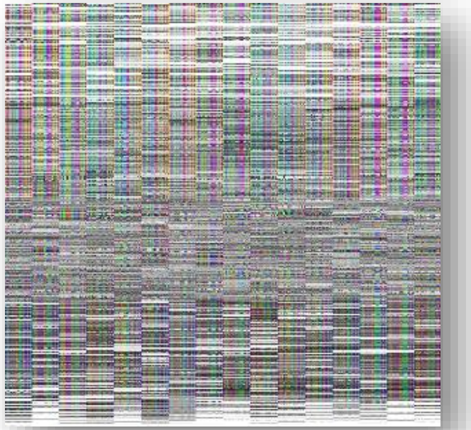


Image cryptée

Image décryptée

Complexité : O(X/2 * Y * Y) car on effectue Y fois X/2 fois (dans le pire cas : avant 1 pixel) Y rotations (dans le pire cas). Soit : O(X * Y^2) donc plus long que le code précédent.

Remarque : Clef cassable facilement par force brute car les opérations récursive sur les lignes sont facile a faire tourner par force brute.

Solutions contre le craquage intelligent :

- Méthode rotation RGB :

Exemple : On utilise les 4 premiers coefficients non utilisés de la clef :

Clef = [1, 8, 7, 9, 4, 6, 1, 9, 7, 5, 3, 6, 4, 8, 7, 9, 11, 03, 20, 21, 12, 58, 24, 25]

a b c d

Tel que :

On pose des conditions sur la parité des lignes et colonnes et on effectue le nombre de rotation de RGB (à l'aide de img.getpixel) qui vaut : a ou b ou c ou d.

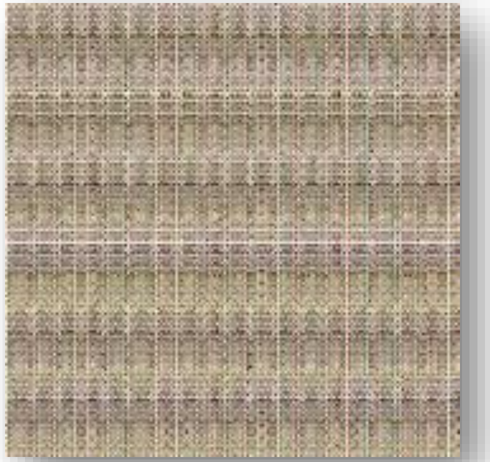


Image cryptée

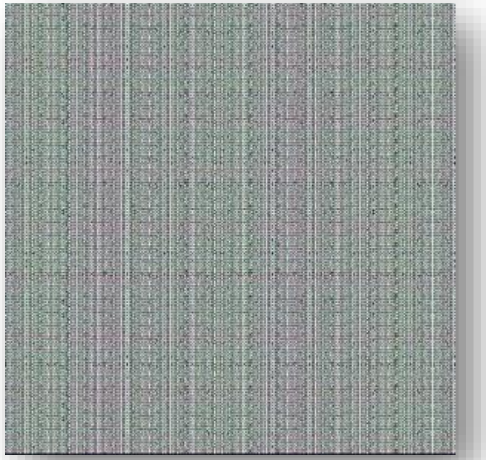


Image cryptée + RGB

Solutions contre le craquage intelligent :

- Méthode rotation RGB :

Conséquence (mathématiques) :

Méthode affine :

Nombre d'opérations à effectuer = $10^{24} * \binom{24}{8} * \binom{100}{4} * 10^{1920} = 3.10^{1956}$

$$<=> 9.10^{1924}\ milliards\ d'années$$

Equivalent à cette échelle

Méthode affine + Rotation RGB :

Nombre d'opérations à effectuer = $10^{24} * \binom{24}{12} * \binom{100}{4} * 10^{1920} = 12.10^{1956}$

$$<=> \boxed{36.10^{1924}\ milliards\ d'années}$$

Solutions contre le craquage intelligent :

- Méthode Rotation RGB :

Conséquence (visuelle):

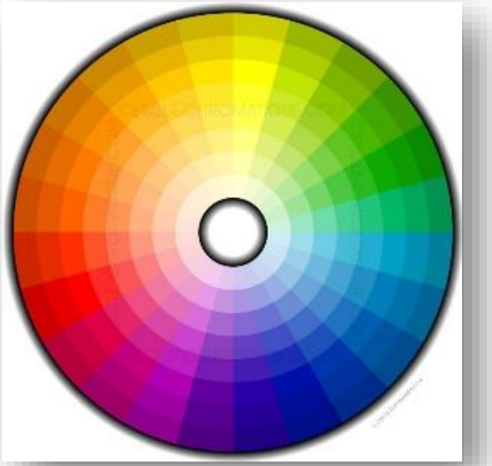


Image source

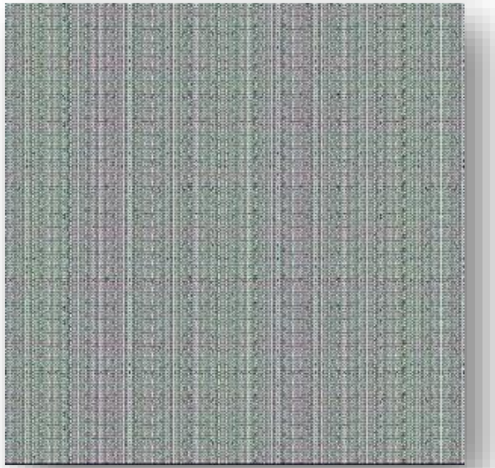


Image cryptée + rotation RGB

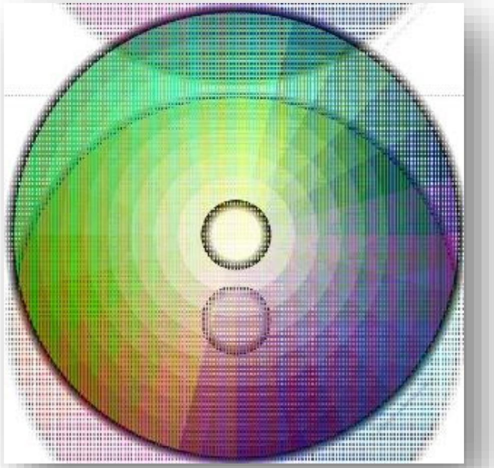


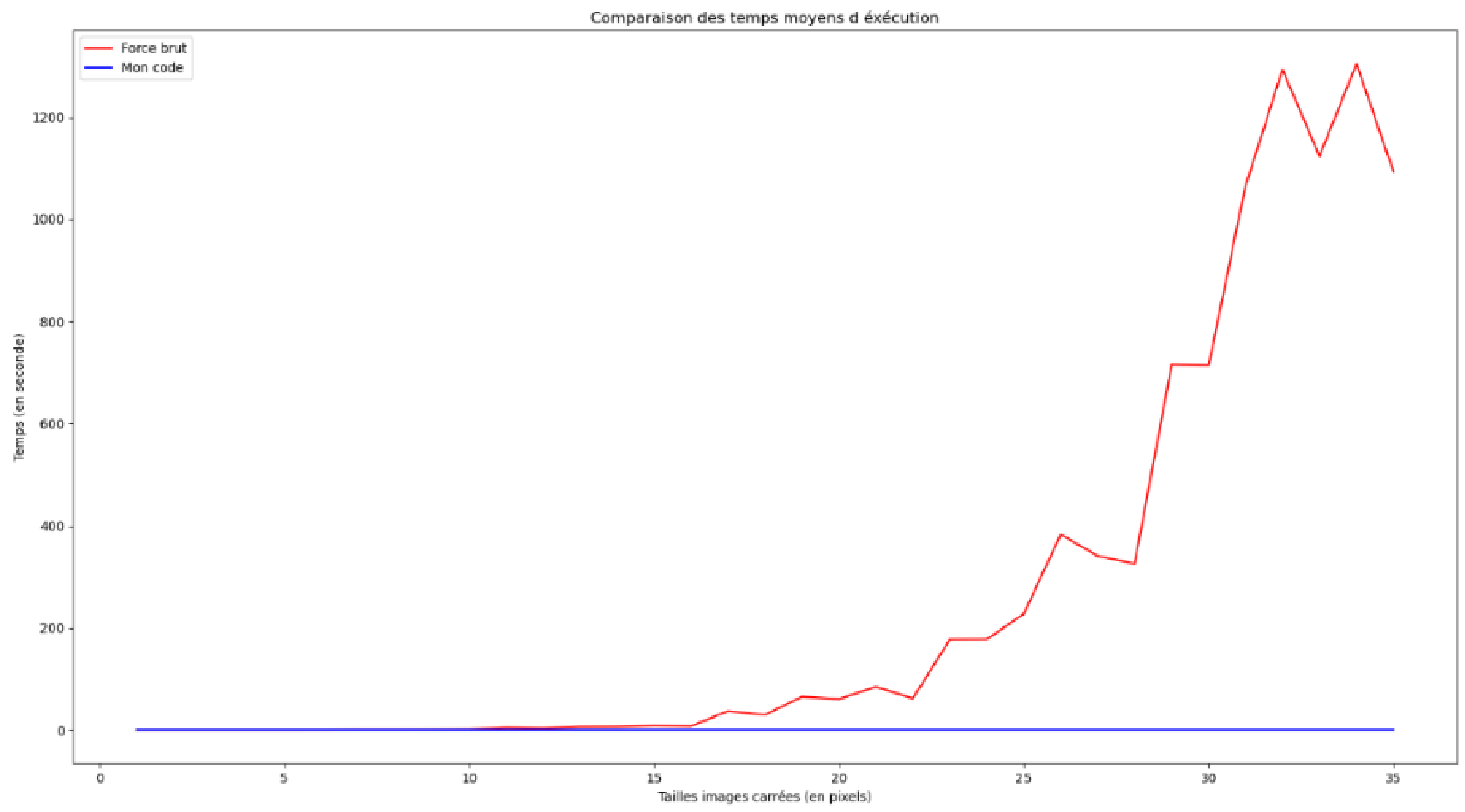
Image décryptée sans rotation RGB

Image décryptée avec rotation RGB

Remarque : Si c'est un texte, le fait de ne pas décrypter la rotation RGB est encore plus embêtante pour le « hacker » car l'image n'aura plus aucun sens.

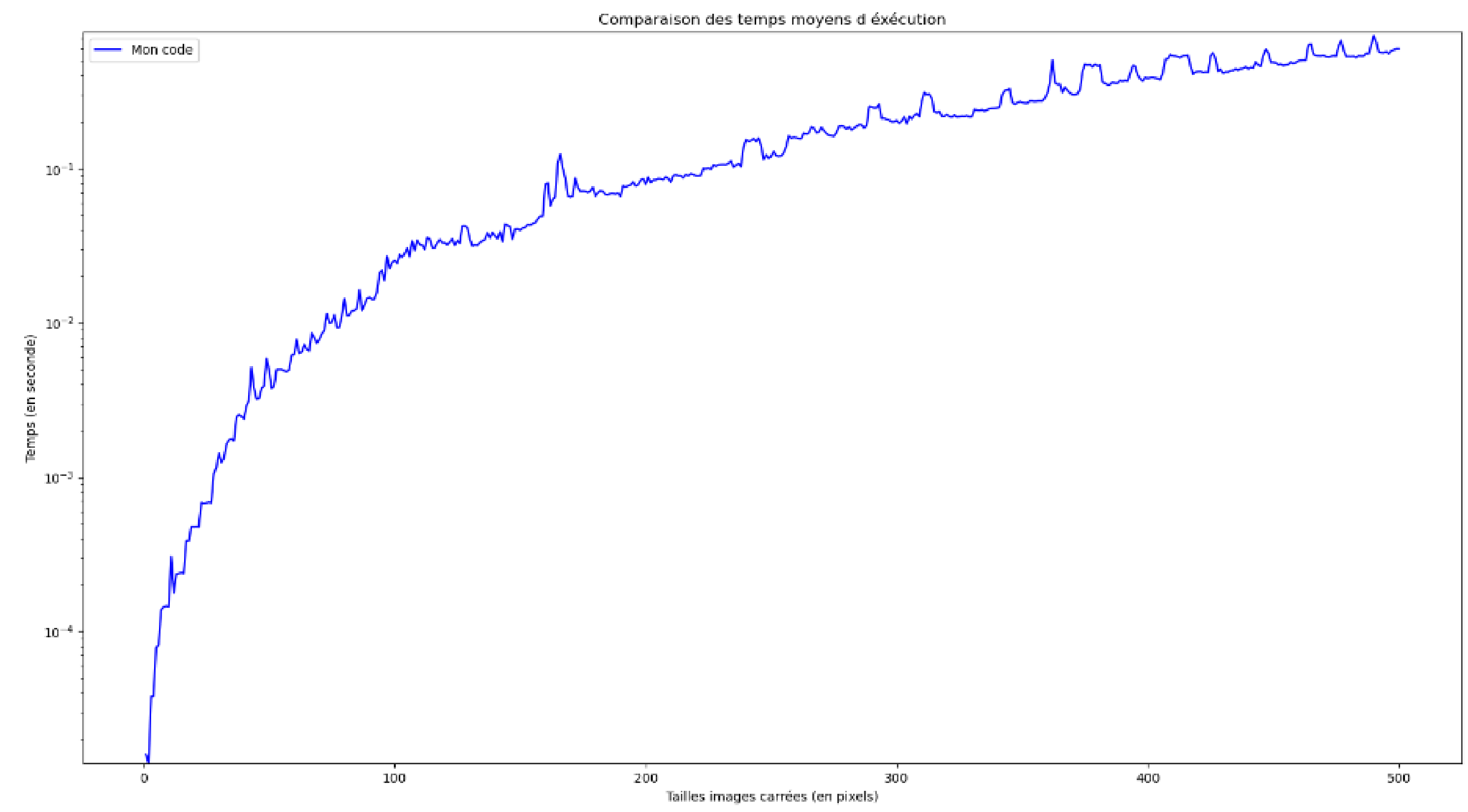
Comparaison temps moyens :

Comparaison avec le code force brute (moyenne de 100) :



Comparaison temps moyens :

Comparaison avec le code force brute (moyenne de 500) :



Rendu final:

Si on combine les deux méthodes de cryptage alors on obtient :

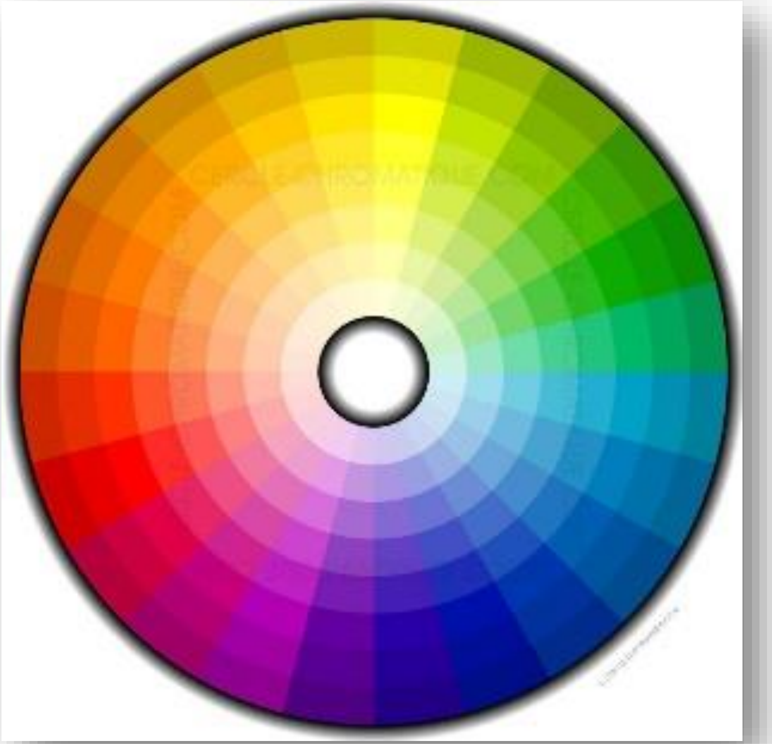


Image source

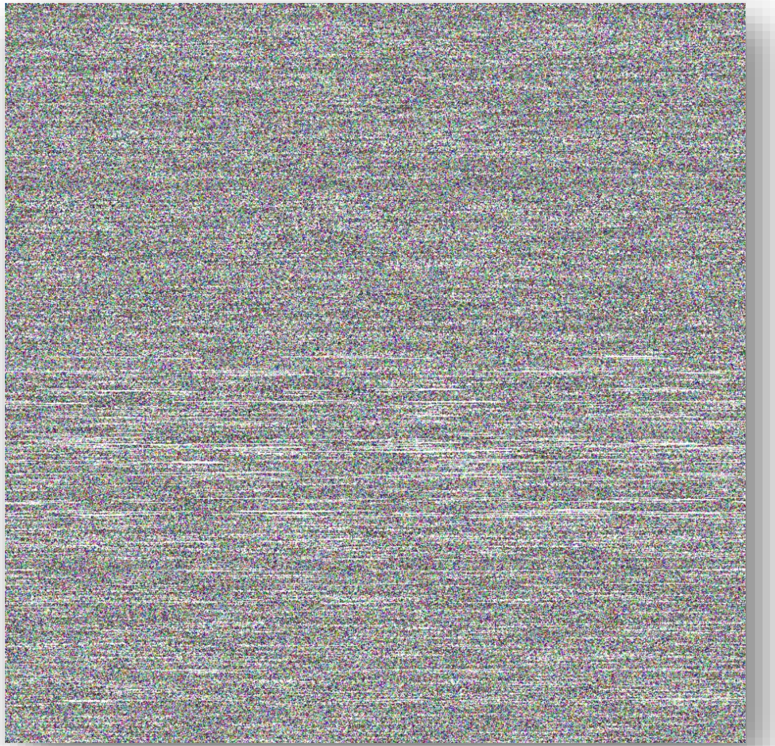


Image cryptée

Image décryptée