Chiffrement de type AES, un moyen efficace pour sécuriser un message ?

41620-Dubus Antoine

Sommaire :

1) Principe de l'AES développé par Rijndael

Réalise des opérations sur des chaînes de caractères de 128 bits = 16 octets avec utilisation d'une clé

Développé en 2000

Utilisé pour protéger des données

2) Méthode de chiffrement d'un texte

Passage en hexadécimal

Hexadécimal	Langage courant
['00', '01', … , '0a', … , 'ff']	Code ASCII
'30'	'0'
'41'	'A'
'61'	'a'
'7a'	'z'

2) Méthode de chiffrement d’un texte

Passage en hexadécimal

'exemple de texte' → [['e', 'x', 'e', 'm'], ['p', 'l', 'e', ' '], ['d', 'e', ' ', 't'], ['e', 'x', 't', 'e']] → [['65', '78', '65', '6d'], ['70', '6c', '65', '20'], ['64', '65', '20', '74'], ['65', '78', '74', '65']]

2) Méthode de chiffrement d'un texte

subBytes

S = ['63', '7c', '77', '7b', 'f2', … , 'bb', '16']

SubBytes : ['00', … , 'ff'] ⟶ ['00', … , 'ff']

'00'	⟼	'63'
'01'	⟼	'7c'
'02'	⟼	'77'
'03'	⟼	'7b'
…		…
'fe'	⟼	'bb'
'ff'	⟼	'16'

2) Méthode de chiffrement d'un texte

subBytes

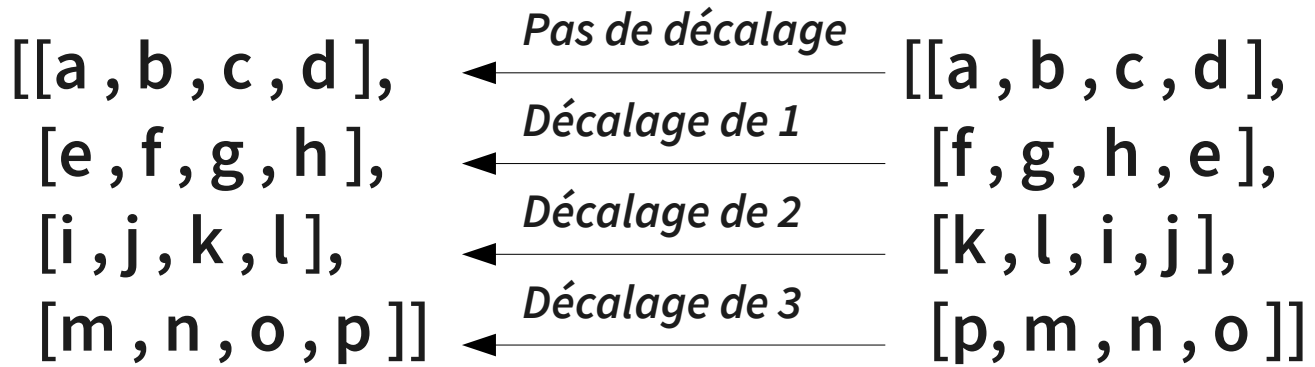
```
[['65', '78', '65', '6d'],
 ['70', '6c', '65', '20'],
 ['64', '65', '20', '74'],
 ['65', '78', '74', '65']]
```


```
[['4d', 'bc', '4d', '3c'],
 ['51', '50', '4d', 'b7'],
 ['43', '4d', 'b7', '92'],
 ['4d', 'bc', '92', '4d']]
```

2) Méthode de chiffrement d'un texte
shiftRows

ShiftRows :

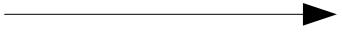
Décalage vers la gauche du numéro de ligne



2) Méthode de chiffrement d'un texte

shiftRows

```
[['4d', 'bc', '4d', '3c'],
 ['51', '50', '4d', 'b7'],
 ['43', '4d', 'b7', '92'],
 ['4d', 'bc', '92', '4d']]
```




```
[['4d', 'bc', '4d', '3c'],
 ['50', '4d', 'b7', '51'],
 ['b7', '92', '43', '4d'],
 ['4d', '4d', 'bc', '92']]
```

2) Méthode de chiffrement d'un texte
mixColumns

MixColumns :

Décalage vers le haut du numéro de colonne + numéro du tour

numéro du tour = 3

[[a , b , c , d],
[e , f , g , h],
[i , j , k , l],
[m , n , o , p]]

(1) (2) (3) (4)

[[m , b , g , l],
[a , f , k , p],
[e , j , o , d],
[i , n , c , h]]

(1) Décalage de 3

(2) Décalage de 4

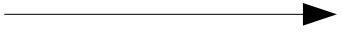
(3) Décalage de 5

(4) Décalage de 6

2) Méthode de chiffrement d'un texte

mixColumns

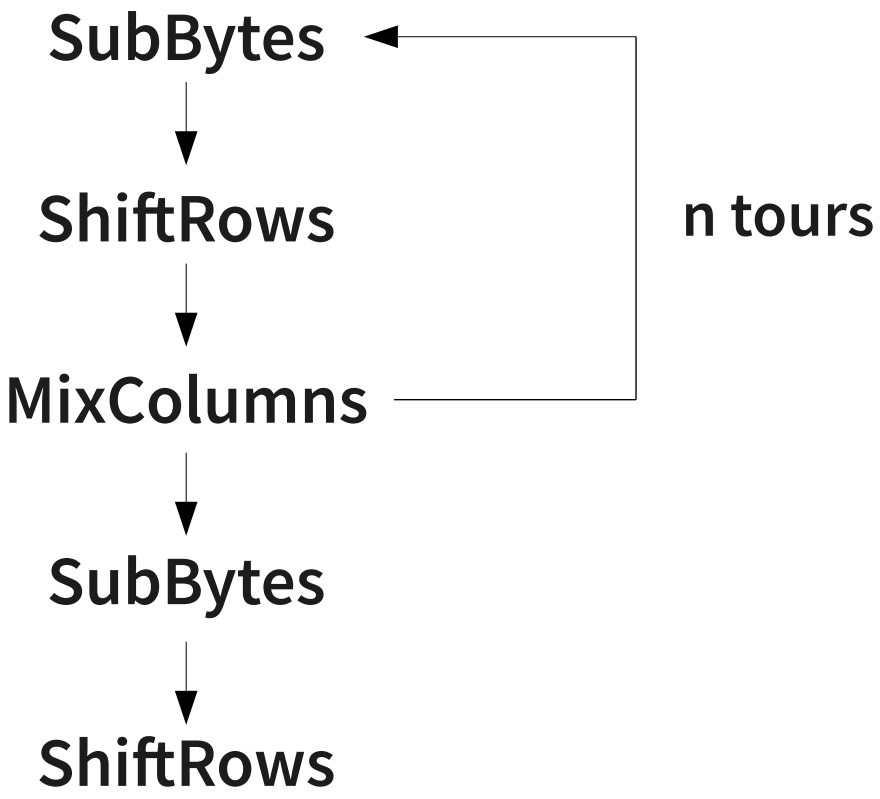
```
[['4d', 'bc', '4d', '3c'],
 ['50', '4d', 'b7', '51'],
 ['b7', '92', '43', '4d'],
 ['4d', '4d', 'bc', '92']]
```




```
[['4d', 'bc', 'b7', '4d'],
 ['4d', '4d', '43', '92'],
 ['50', '92', 'bc', '3c'],
 ['b7', '4d', '4d', '51']]
```

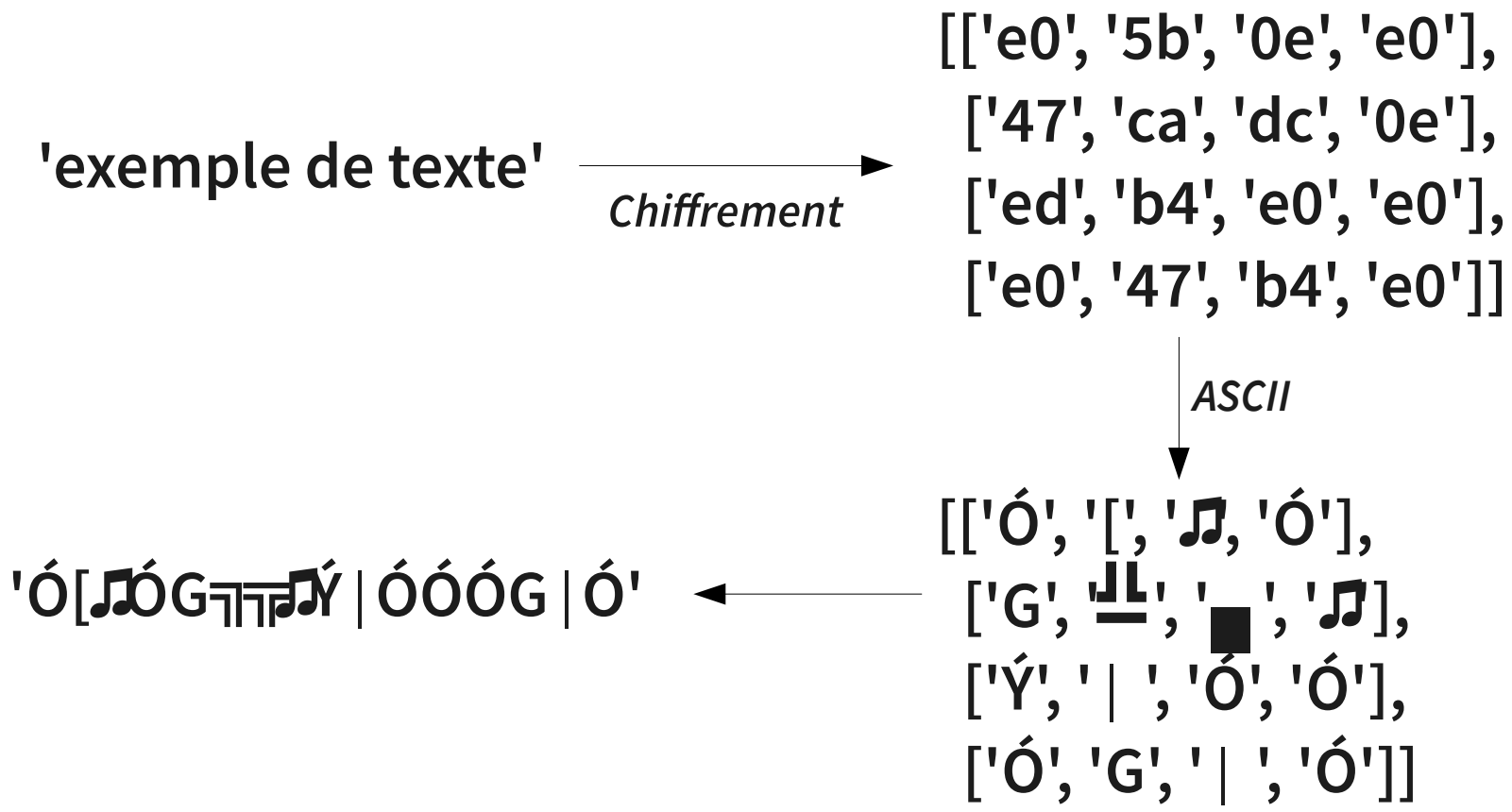
2) Méthode de chiffrement d'un texte

Chiffrement :



2) Méthode de chiffrement d'un texte

n = 10 tours dans le chiffrement



3) Méthode de déchiffrement

subBytes_réciproque

S_inv = ['52', '09', '6a', 'd5', '30', … , '0c', '7d']

SubBytes_réciproque : ['00', … , 'ff'] ⟶ ['00', … , 'ff']

'00'	⟼	'52'
'01'	⟼	'09'
'02'	⟼	'6a'
'03'	⟼	'd5'
…		…
'fe'	⟼	'0c'
'ff'	⟼	'7d'

3) Méthode de déchiffrement

subBytes_réciproque

```
[['65', '78', '65', '6d'],
 ['70', '6c', '65', '20'],
 ['64', '65', '20', '74'],
 ['65', '78', '74', '65']]
```

SubBytes →

```
[['4d', 'bc', '4d', '3c'],
 ['51', '50', '4d', 'b7'],
 ['43', '4d', 'b7', '92'],
 ['4d', 'bc', '92', '4d']]
```

```
[['4d', 'bc', '4d', '3c'],
 ['51', '50', '4d', 'b7'],
 ['43', '4d', 'b7', '92'],
 ['4d', 'bc', '92', '4d']]
```

SubBytes_réciproque →

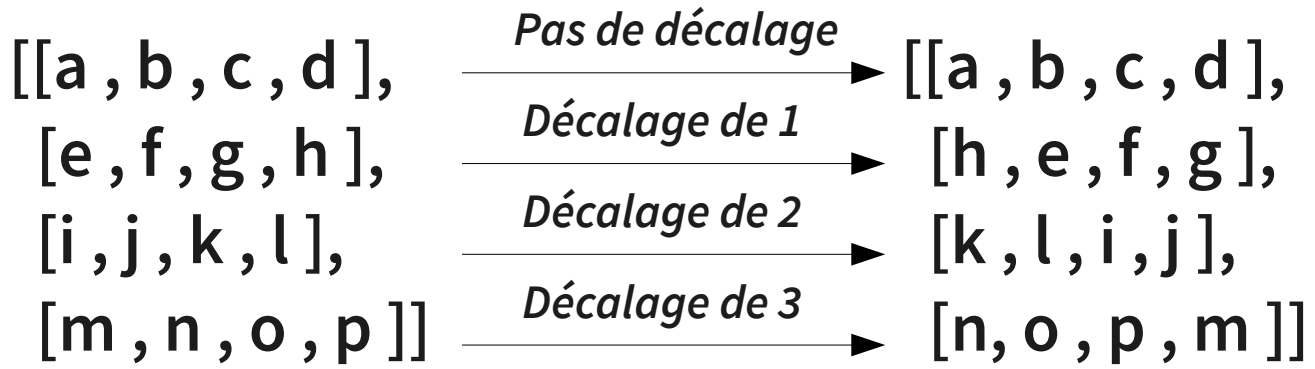
```
[['65', '78', '65', '6d'],
 ['70', '6c', '65', '20'],
 ['64', '65', '20', '74'],
 ['65', '78', '74', '65']]
```

3) Méthode de déchiffrement

shiftRows_réciproque

ShiftRows_réciproque :

Décalage vers la droite du numéro de ligne



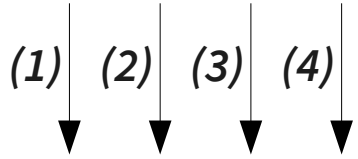
3) Méthode de déchiffrement
mixColumns_réciproque

MixColumns_réciproque :

Décalage vers le bas du numéro de colonne + numéro du tour

numéro du tour = 3

[[a , b , c , d],
[e , f , g , h],
[i , j , k , l],
[m , n , o , p]]



[[e , b , o , l],
[i , f , c , p],
[m , j , g , d],
[a , n , k , h]]

(1) Décalage de 3

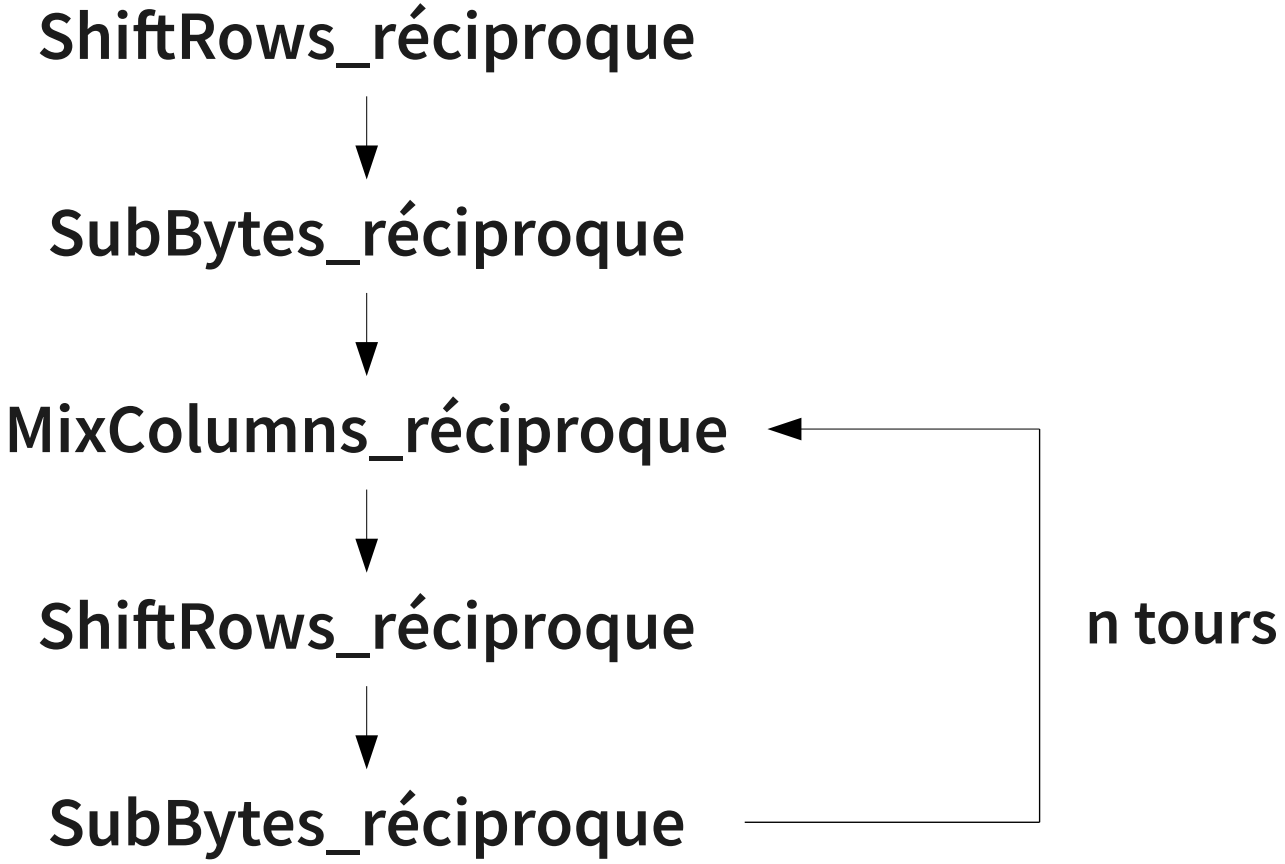
(2) Décalage de 4

(3) Décalage de 5

(4) Décalage de 6

3) Méthode de déchiffrement

Déchiffrement :



3) Méthode de déchiffrement

n = 10 tours dans le déchiffrement

Déchiffrement

[['e0', '5b', '0e', 'e0'],
['47', 'ca', 'dc', '0e'],
['ed', 'b4', 'e0', 'e0'],
['e0', '47', 'b4', 'e0']]

ASCII

[['e', 'x', 'e', 'm'],
['p', 'l', 'e', ' '],
['d', 'e', ' ', 't'],
['e', 'x', 't', 'e']]

'exemple de texte'

4) Efficacité de cette méthode

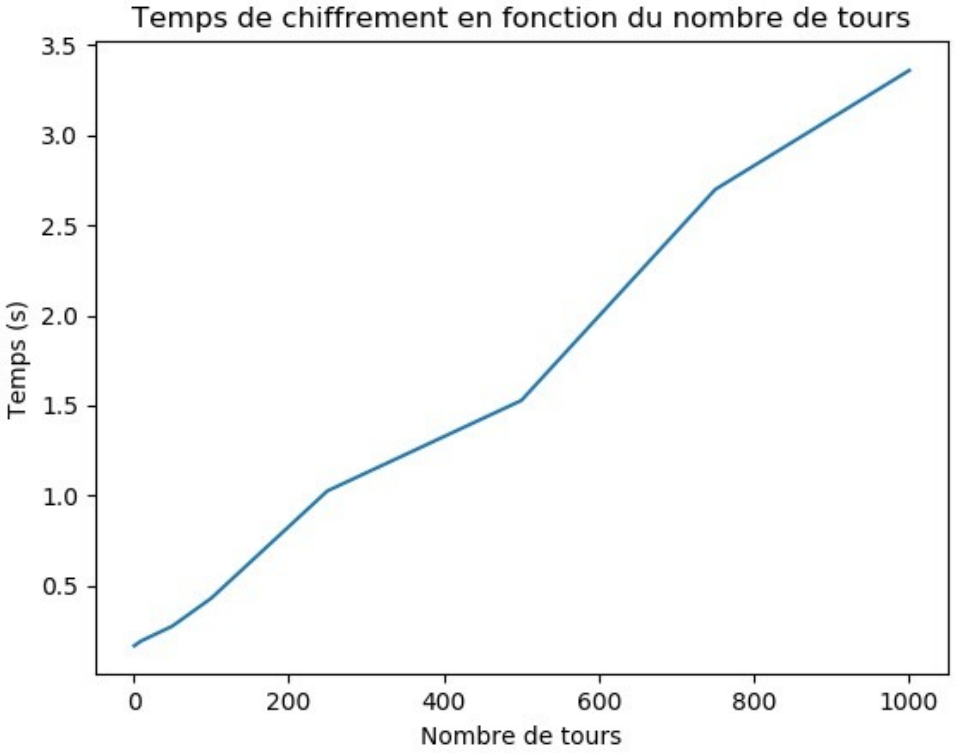
Complexité

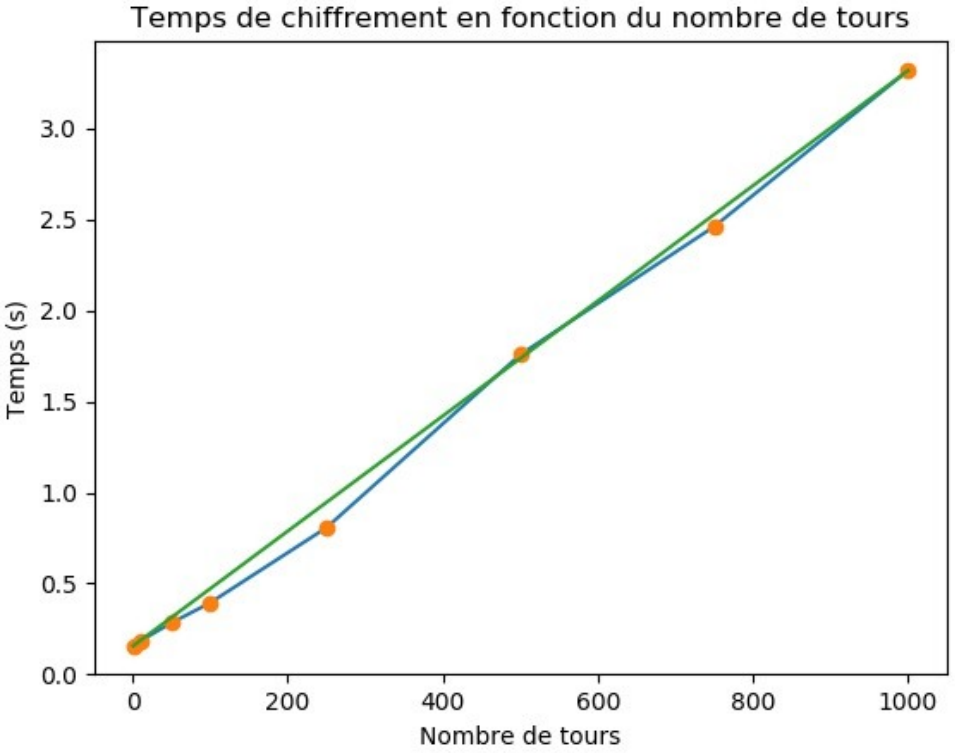
Complexités :

C(chiffrement) = O(nombre_tours*taille_texte)

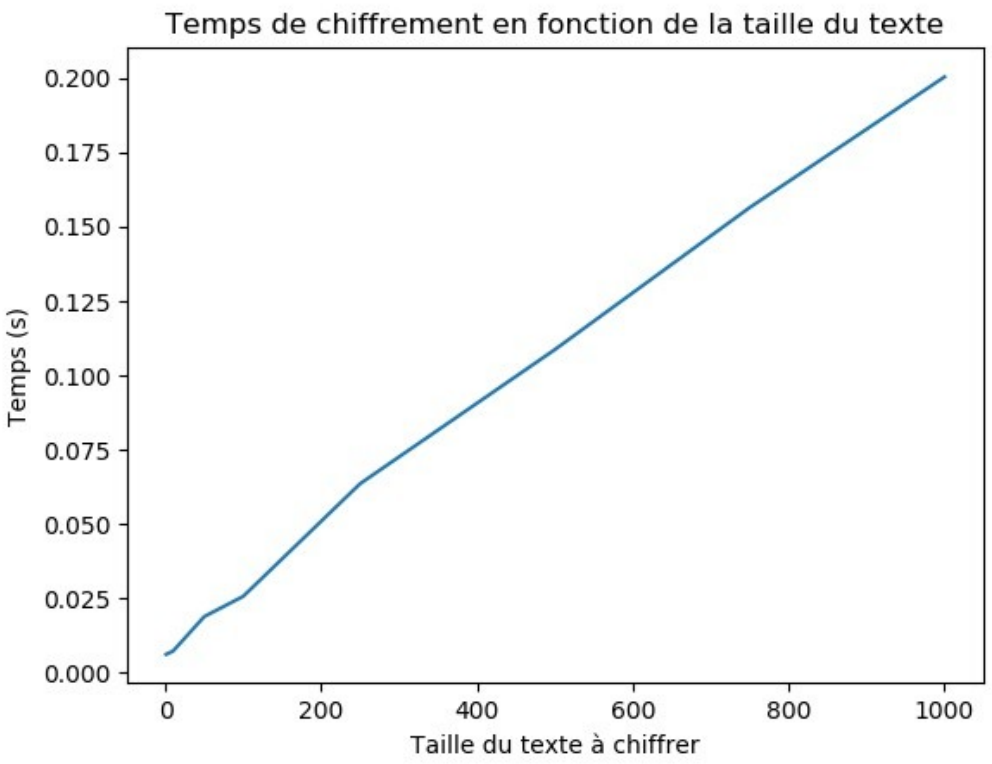
C(déchiffrement) = O(nombre_tours*taille_texte)

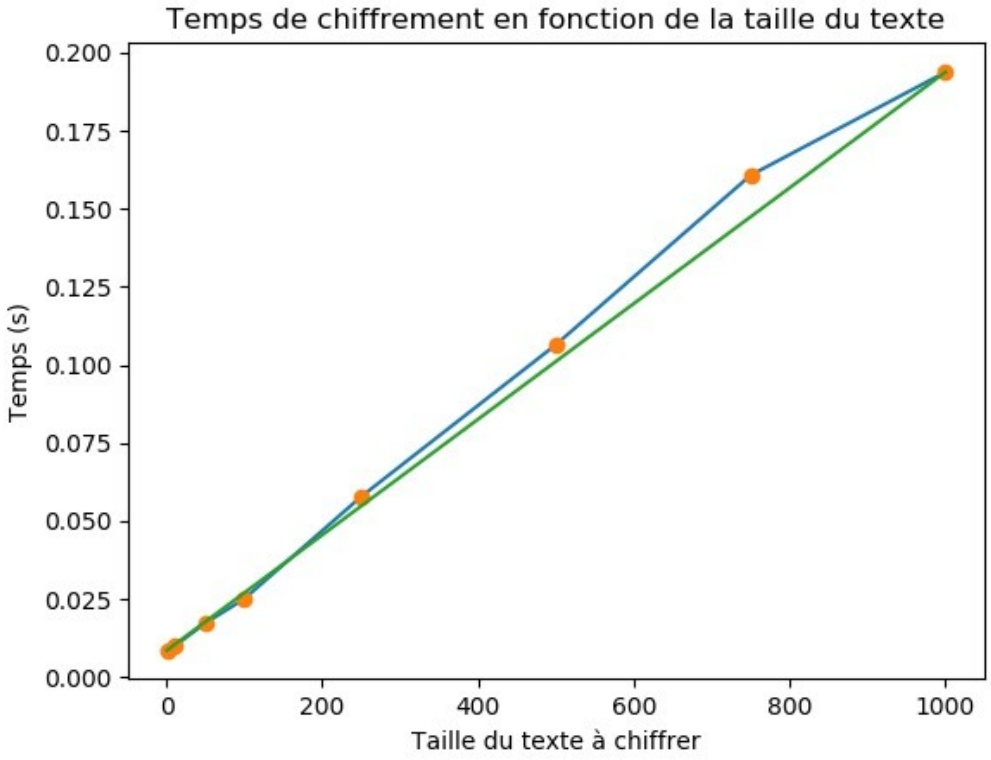
4) Efficacité de cette méthode





4) Efficacité de cette méthode





4) Efficacité de cette méthode

Répétition de séquence :

n = 10 tours

'exemple**exemple**exemple**exemple**exemple'

'Ó╥ÝÓÓÓÓGÓÓÝÝÓÝÝÓGÓ╖ G ¸ Ó╥╖╖ Ó╥╖╖ ╖ GÓÓGÓ╥╥╖╖ '

4) Efficacité de cette méthode

Changement d’un caractère :

'exemple de texte' → 'Ó[♫ÓG╗ ╥♫Ý | ÓÓÓG | Ó'

'axemple de texte' → 'Ó[♫¸ G╗ ╥♫Ý | ÓÓÓG | Ó'

'aexemple de texte' → '¸ÓGÓ╗ ╥Ó | [Ý♫ÓGÓ | ¸¸♫Ó¸¸¸¸¸¸'

4) Efficacité de cette méthode

Influence du nombre de tours de chiffrement :

'exemple' —*9 tours*→ '&áôSáá■►&'

'exemple' —*10 tours*→ '¸Ó¸╗╤ÝÓGÓ'

'exemple' —*11 tours*→ 'táhßUßßhå'

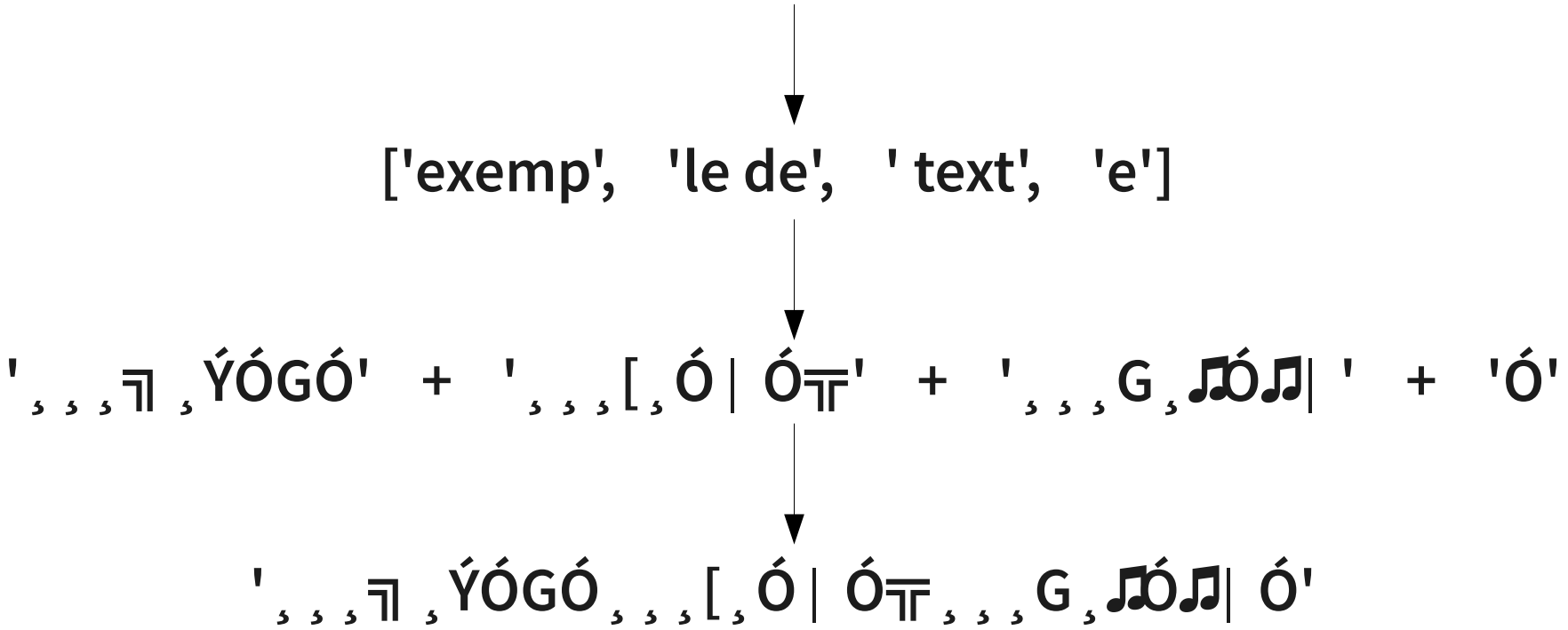
'exemple' —*12 tours*→ '°Ó°³DÆE°E'

5) Améliorations possibles

Chiffrement avec taille de chiffrement

n = 10 tours taille de chiffrement = 5

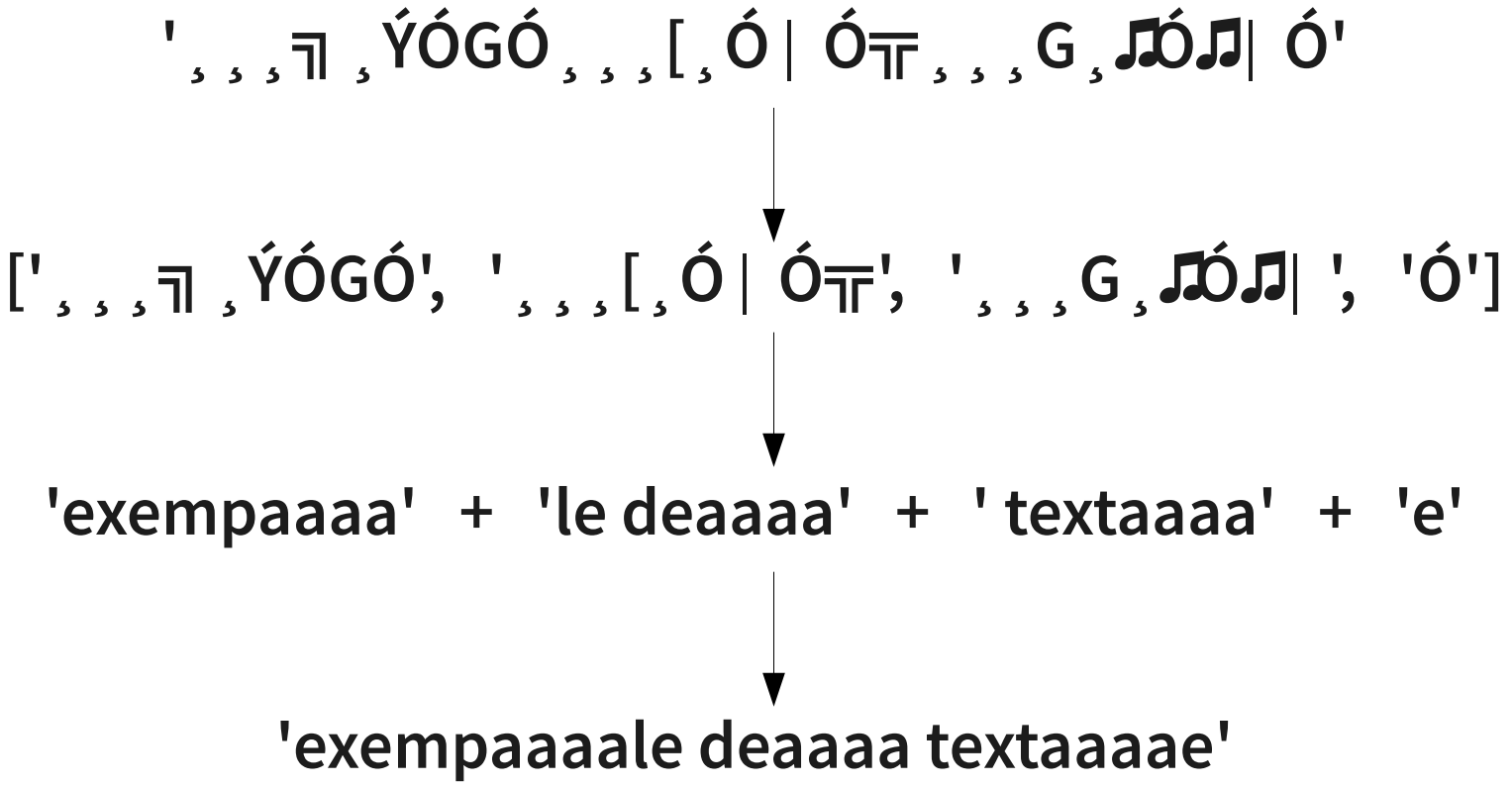
'exemple de texte'



5) Améliorations possibles

Déchiffrement avec taille de chiffrement

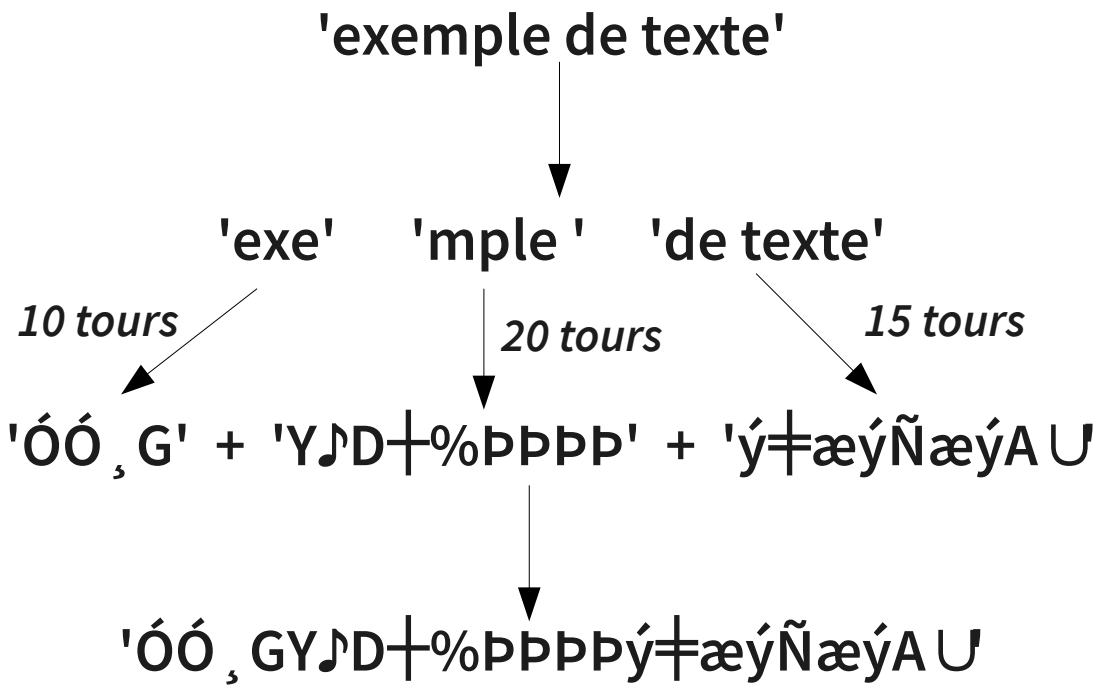
n = 10 tours taille_chiff = 5 Carré parfait suivant 5 : 9



5) Améliorations possibles

Chiffrement avec clé

cle = '0310052015' 03 caractères : 10 tours 05 caractères : 20 tours
caractères restants : 15 tours



Robustesse : plus de relation statistique entre texte clair et chiffré (confusion)

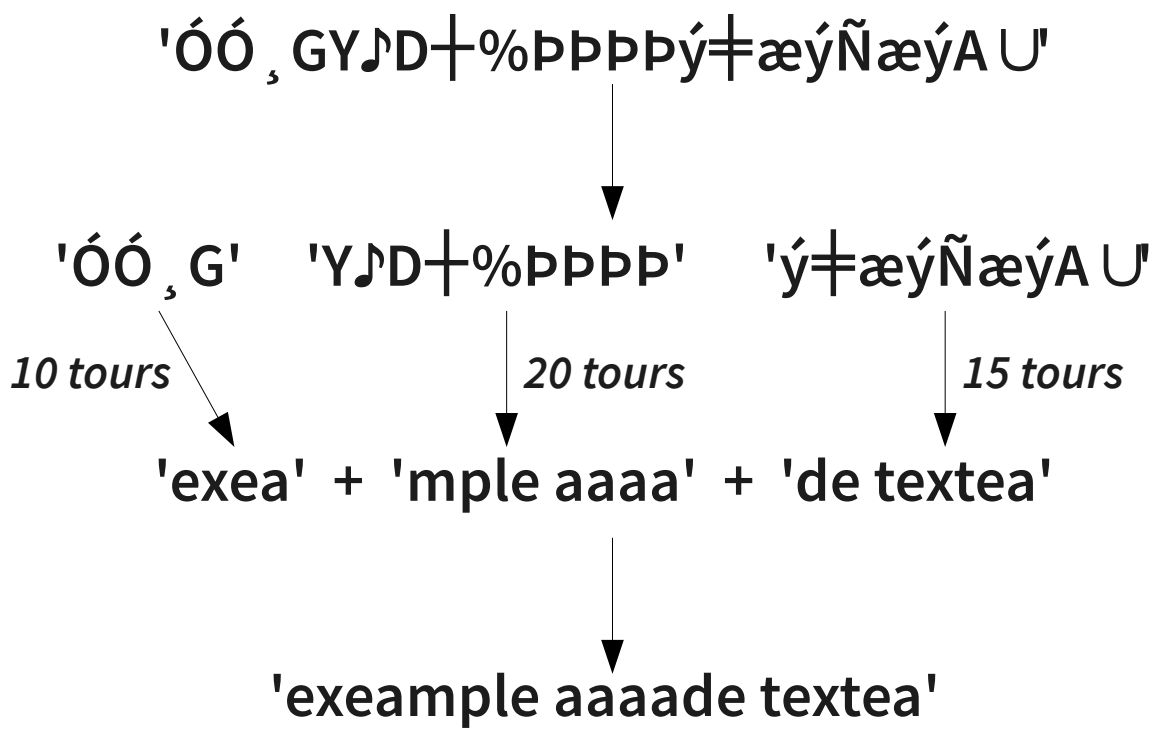
5) Améliorations possibles

Déchiffrement avec clé

cle = '0310052015' ['03', '10', '05', '20', '15'] → ['4', '10', '9', '20', '15']

4 caractères : 10 tours 9 caractères : 20 tours

caractères restants : 15 tours



6) Conclusion

Objectif du TIPE atteint

Annexe

```
# S-box (substitution box)

S = [['63', '7c', '77', '7b', 'f2', '6b', '6f', 'c5', '30', '01', '67', '2b', 'fe', 'd7', 'ab', '76'],
     ['ca', '82', 'c9', '7d', 'fa', '59', '47', 'f0', 'ad', 'd4', 'a2', 'af', '9c', 'a4', '72', 'c0'],
     ['b7', 'fd', '93', '26', '36', '3f', 'f7', 'cc', '34', 'a5', 'e5', 'f1', '71', 'd8', '31', '15'],
     ['04', 'c7', '23', 'c3', '18', '96', '05', '9a', '07', '12', '80', 'e2', 'eb', '27', 'b2', '75'],
     ['09', '83', '2c', '1a', '1b', '6e', '5a', 'a0', '52', '3b', 'd6', 'b3', '29', 'e3', '2f', '84'],
     ['53', 'd1', '00', 'ed', '20', 'fc', 'b1', '5b', '6a', 'cb', 'be', '39', '4a', '4c', '58', 'cf'],
     ['d0', 'ef', 'aa', 'fb', '43', '4d', '33', '85', '45', 'f9', '02', '7f', '50', '3c', '9f', 'a8'],
     ['51', 'a3', '40', '8f', '92', '9d', '38', 'f5', 'bc', 'b6', 'da', '21', '10', 'ff', 'f3', 'd2'],
     ['cd', '0c', '13', 'ec', '5f', '97', '44', '17', 'c4', 'a7', '7e', '3d', '64', '5d', '19', '73'],
     ['60', '81', '4f', 'dc', '22', '2a', '90', '88', '46', 'ee', 'b8', '14', 'de', '5e', '0b', 'db'],
     ['e0', '32', '3a', '0a', '49', '06', '24', '5c', 'c2', 'd3', 'ac', '62', '91', '95', 'e4', '79'],
     ['e7', 'c8', '37', '6d', '8d', 'd5', '4e', 'a9', '6c', '56', 'f4', 'ea', '65', '7a', 'ae', '08'],
     ['ba', '78', '25', '2e', '1c', 'a6', 'b4', 'c6', 'e8', 'dd', '74', '1f', '4b', 'bd', '8b', '8a'],
     ['70', '3e', 'b5', '66', '48', '03', 'f6', '0e', '61', '35', '57', 'b9', '86', 'c1', '1d', '9e'],
     ['e1', 'f8', '98', '11', '69', 'd9', '8e', '94', '9b', '1e', '87', 'e9', 'ce', '55', '28', 'df'],
     ['8c', 'a1', '89', '0d', 'bf', 'e6', '42', '68', '41', '99', '2d', '0f', 'b0', '54', 'bb', '16']]

S_inv = [['52', '09', '6a', 'd5', '30', '36', 'a5', '38', 'bf', '40', 'a3', '9e', '81', 'f3', 'd7', 'fb'],
         ['7c', 'e3', '39', '82', '9b', '2f', 'ff', '87', '34', '8e', '43', '44', 'c4', 'de', 'e9', 'cb'],
         ['54', '7b', '94', '32', 'a6', 'c2', '23', '3d', 'ee', '4c', '95', '0b', '42', 'fa', 'c3', '4e'],
         ['08', '2e', 'a1', '66', '28', 'd9', '24', 'b2', '76', '5b', 'a2', '49', '6d', '8b', 'd1', '25'],
         ['72', 'f8', 'f6', '64', '86', '68', '98', '16', 'd4', 'a4', '5c', 'cc', '5d', '65', 'b6', '92'],
         ['6c', '70', '48', '50', 'fd', 'ed', 'b9', 'da', '5e', '15', '46', '57', 'a7', '8d', '9d', '84'],
         ['90', '8d', 'ab', '00', '8c', 'bc', 'd3', '0a', 'f7', 'e4', '58', '05', 'b8', 'b3', '45', '06'],
         ['d0', '2c', '1e', '8f', 'ca', '3f', '0f', '02', 'c1', 'af', 'bd', '03', '01', '13', '8a', '6b'],
         ['3a', '91', '11', '41', '4f', '67', 'dc', 'ea', '97', 'f2', 'cf', 'ce', 'f0', 'b4', 'e6', '73'],
         ['96', 'ac', '74', '22', 'e7', 'ad', '35', '85', 'e2', 'f9', '37', 'e8', '1c', '75', 'df', '6e'],
         ['47', 'f1', '1a', '71', '1d', '29', 'c5', '89', '6f', 'b7', '62', '0e', 'aa', '18', 'be', '1b'],
         ['fc', '56', '3e', '4b', 'c6', 'd2', '79', '20', '9a', 'db', 'c0', 'fe', '78', 'cd', '5a', 'f4'],
         ['1f', 'dd', 'a8', '33', '88', '07', 'c7', '31', 'b1', '12', '10', '59', '27', '80', 'ec', '5f'],
         ['60', '51', '7f', 'a9', '19', 'b5', '4a', '0d', '2d', 'e5', '7a', '9f', '93', 'c9', '9c', 'ef'],
         ['a0', 'e0', '3b', '4d', 'ae', '2a', 'f5', 'b0', 'c8', 'eb', 'bb', '3c', '83', '53', '99', '61'],
         ['17', '2b', '04', '7e', 'ba', '77', 'd6', '26', 'e1', '69', '14', '63', '55', '21', '0c', '7d']]

# S[i][j] donne le terme de la i-ème ligne, j-ème colonne
```

Annexe

```
liste_hexadecimaux = ['00', '01', '02', '03', '04', '05', '06', '07', '08', '09', '0a', '0b', '0c', '0d', '0e',
'0f', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '1a', '1b', '1c', '1d', '1e', '1f', '20', '21',
'22', '23', '24', '25', '26', '27', '28', '29', '2a', '2b', '2c', '2d', '2e', '2f', '30', '31', '32', '33', '34',
'35', '36', '37', '38', '39', '3a', '3b', '3c', '3d', '3e', '3f', '40', '41', '42', '43', '44', '45', '46', '47',
'48', '49', '4a', '4b', '4c', '4d', '4e', '4f', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '5a',
'5b', '5c', '5d', '5e', '5f', '60', '61', '62', '63', '64', '65', '66', '67', '68', '69', '6a', '6b', '6c', '6d',
'6e', '6f', '70', '71', '72', '73', '74', '75', '76', '77', '78', '79', '7a', '7b', '7c', '7d', '7e', '7f', '80',
'81', '82', '83', '84', '85', '86', '87', '88', '89', '8a', '8b', '8c', '8d', '8e', '8f', '90', '91', '92', '93',
'94', '95', '96', '97', '98', '99', '9a', '9b', '9c', '9d', '9e', '9f', 'a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6',
'a7', 'a8', 'a9', 'aa', 'ab', 'ac', 'ad', 'ae', 'af', 'b0', 'b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8', 'b9',
'ba', 'bb', 'bc', 'bd', 'be', 'bf', 'c0', 'c1', 'c2', 'c3', 'c4', 'c5', 'c6', 'c7', 'c8', 'c9', 'ca', 'cb', 'cc',
'cd', 'ce', 'cf', 'd0', 'd1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8', 'd9', 'da', 'db', 'dc', 'dd', 'de', 'df',
'e0', 'e1', 'e2', 'e3', 'e4', 'e5', 'e6', 'e7', 'e8', 'e9', 'ea', 'eb', 'ec', 'ed', 'ee', 'ef', 'f0', 'f1', 'f2',
'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9', 'fa', 'fb', 'fc', 'fd', 'fe', 'ff']

liste_carac_acceptes = ['€', '☺', '☻', '♥', '♦', '♣', '♠', '•', '◘', '○', '◙', '♂', '♀', '♪', '♫', '☼', '►', '◄',
'↕', '‼', '¶', '§', '▬', '↨', '↑', '↓', '→', '←', '└', '↔', '▲', '▼', ' ', '!', '"', '#', '$', '%', '&', "'",
'(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=',
'>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '⌂',
'Ç', 'ü', 'é', 'â', 'ä', 'à', 'å', 'ç', 'ê', 'ë', 'è', 'ï', 'î', 'ì', 'Ä', 'Å', 'É', 'æ', 'Æ', 'ô', 'ö', 'ò',
'û', 'ù', 'ÿ', 'Ö', 'Ü', 'ø', '£', 'Ø', '×', 'ƒ', 'á', 'í', 'ó', 'ú', 'ñ', 'Ñ', 'ª', 'º', '¿', '®', '¬', '½',
'¼', '¡', '«', '»', '░', '▒', '▓', '│', '┤', 'Á', 'Â', 'À', '©', '╣', '║', '╗', '╝', '¢', '¥', '┐', '└', '┴',
'┬', '├', '─', '┼', 'ã', 'Ã', '╚', '╔', '╩', '╦', '╠', '═', '╬', '¤', 'ð', 'Ð', 'Ê', 'Ë', 'È', 'ı', 'Í', 'Î',
'Ï', '┘', '┌', '█', '▄', '¦', 'Ì', '▀', 'Ó', 'ß', 'Ô', 'Ò', 'õ', 'Õ', 'µ', 'þ', 'Þ', 'Ú', 'Û', 'Ù', 'ý', 'Ý',
'¯', '´', '', '±', '‗', '¾', '¶', 'œ', '÷', '¸', '°', '¨', '·', '¹', '³', '²', '■', ' ']

# liste_carac_acceptes[0] = '€' , liste_carac_acceptes[244] = '‰' , liste_carac_acceptes[245] = 'œ' (car sinon on
avait plusieurs fois les mêmes caractères) et pour tout i de 1 à 256 (i différent de 0, 244, 245),
liste_carac_acceptes[i] = 'alt + i'

def verification_unicite_caracteres_liste_carac_acceptes():
    n = len(liste_carac_acceptes)  # n = 256
    for i in range(n):
        for j in range(n):
            if i!=j and liste_carac_acceptes[i]==liste_carac_acceptes[j]:
                return False, (i,j)
    return True
verif = verification_unicite_caracteres_liste_carac_acceptes()  # renvoie True

##########################################################################################
```

Annexe

```
# import os

# import Antoine_Dubus_AES_TIPE as aes    # pour utiliser les fonctions du fichier 'Antoine_Dubus_AES_TIPE'
# cliquer sur Exécuter Démarrer le script (puis F5) pour éxécuter

from time import time
import numpy as np
import matplotlib.pyplot as plt
import math as m

# Si on veut importer un texte à chiffrer depuis un autre fichier
# nom='texteclair.txt'
# os.chdir('D:\\textetipe')
# fichier=open(nom, 'r')
# texteBrut=fichier.read() # importe le fichier comme une grande chaîne de caractères
# fichier.close()


# fonction qui enlève les caractères spéciaux présents dans le texte et renvoie une liste contenant les caractères de ce texte sans caractères spéciaux
def liste_sans_carac_spec(texte):
    #liste_carac_acceptes = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
    # liste_carac_acceptes est une variable globale de 256 caractères différents
    n = len(texte)
    liste_sans_carac_spec = []
    for i in range(n):
        if texte[i] in liste_carac_acceptes:
            liste_sans_carac_spec.append(texte[i])
    return liste_sans_carac_spec


# hex convertit un entier en chaîne hexadécimale préfixée de 0x
# int convertit une chaîne de caractère (ou un nombre) en entier  int('a', base=36)  (base va de 2 à 36)
# de chaîne de caractère à hexadécimal :
# int('essai', base=36) donne un entier à partir d'une chaîne de caractère
# hex(int('essai', base=36)) donne une chaîne hexadécimale préfixée par 0x à partir d'une chaîne de caractère
# hex(int('essai', base=36)).lstrip('0x') donne une chaîne hexadécimale sans le 0x à partir d'une chaîne de caractère

def texte2int(texte):
    return int(texte, base=36)

def texte2hex(texte):
    return hex(int(texte, base=36))
```

Annexe

```
 99  def texte2hexSans0x(texte):
100      return hex(int(texte, base=36))[2::]
101  # on utilisera plutôt la correspondance entre liste_carac_acceptes et liste_hexadecimaux pour passer du langage
     courant à l'hexadécimal et inversement
102
103
104  # convertit chaque élément de la liste en hexadécimal
105  # def liste_hexa(liste):
106  #     n = len(liste)
107  #     liste_hexa = []
108  #     for i in range(n):
109  #         liste_hexa.append(texte2hexSans0x(liste[i]))
110  #     for i in range(len(liste_hexa)):
111  #         if len(liste_hexa[i]) == 1:
112  #             liste_hexa[i] = '0' + liste_hexa[i]
113  #     return liste_hexa
114  def lang_courant2hexa_caractere(carac):
115      n = len(liste_hexadecimaux)  # n = 256
116      for i in range(n):
117          if carac == liste_carac_acceptes[i]:
118              return liste_hexadecimaux[i]
119
120  # convertit chaque élément de la liste de caractères du langage courant en hexadécimal
121  def liste_hexa(liste):
122      n = len(liste)
123      liste_hexa = []
124      for i in range(n):
125          liste_hexa.append(lang_courant2hexa_caractere(liste[i]))
126      for i in range(len(liste_hexa)):
127          if len(liste_hexa[i]) == 1:
128              liste_hexa[i] = '0' + liste_hexa[i]
129      return liste_hexa
130  # liste_hexa(['e', 'x', 'e', 'm', 'p', 'l', 'e']) renvoie ['65', '78', '65', '6d', '70', '6c', '65']
131
132  # regroupe les caractères en hexadécimal de la liste en une seule chaîne de caractère
133  def regroupe_liste(liste):
134      n = len(liste)
135      texteHexa = ''
136      for i in range(n):
137          texteHexa += liste[i]
138      return texteHexa
139  # regroupe_liste(['65', '78', '65', '6d', '70', '6c', '65']) renvoie '6578656d706c65'
```

Annexe

```
142 # fonction qui donne la liste des hexadécimaux pour quelques caractères
143 def essai_hex():
144     liste =
['0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r',
's','t','u','v','w','x','y','z']
145     liste_hex = []
146     for i in range(len(liste)):
147         liste_hex.append(hex(int(liste[i], base = 36)))
148     return liste_hex
149 
150 print(essai_hex())  # renvoie ['0x0', '0x1', '0x2', '0x3', '0x4', '0x5', '0x6', '0x7', '0x8', '0x9', '0xa',
'0xb', '0xc', '0xd', '0xe', '0xf', '0x10', '0x11', '0x12', '0x13', '0x14', '0x15', '0x16', '0x17', '0x18',
'0x19', '0x1a', '0x1b', '0x1c', '0x1d', '0x1e', '0x1f', '0x20', '0x21', '0x22', '0x23']
151 
152 # fonction qui donne la liste des hexadécimaux sans préfixe 0x pour quelques caractères
153 def essai_hex_sans_0x():
154     liste =
['0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r',
's','t','u','v','w','x','y','z']
155     liste_hex = []
156     for i in range(len(liste)):
157         liste_hex.append(hex(int(liste[i], base = 36))[2::])
158     return liste_hex
159 
160 print(essai_hex_sans_0x())  # renvoie ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e',
'f', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '1a', '1b', '1c', '1d', '1e', '1f', '20', '21',
'22', '23']
161 
162 ### exemples
163 texte_exemple_1 = 'exemple de texte'
164 liste_exemple_1 = liste_sans_carac_spec(texte_exemple_1)  # renvoie ['e', 'x', 'e', 'm', 'p', 'l', 'e', ' ', 'd',
'e', ' ', 't', 'e', 'x', 't', 'e']
165 liste_hex_exemple_1 = liste_hexa(liste_exemple_1)  # renvoie ['65', '78', '65', '6d', '70', '6c', '65', '20',
'64', '65', '20', '74', '65', '78', '74', '65']
166 
167 texte_exemple_2 = 'exemple de texte pour le chiffrement du tipe'
168 liste_exemple_2 = liste_sans_carac_spec(texte_exemple_2)  # renvoie ['e', 'x', 'e', 'm', 'p', 'l', 'e', ' ', 'd',
'e', ' ', 't', 'e', 'x', 't', 'e', ' ', 'p', 'o', 'u', 'r', ' ', 'l', 'e', ' ', 'c', 'h', 'i', 'f', 'f', 'r',
'e', 'm', 'e', 'n', 't', ' ', 'd', 'u', ' ', 't', 'i', 'p', 'e']
169 liste_hex_exemple_2 = liste_hexa(liste_exemple_2)   # renvoie ['65', '78', '65', '6d', '70', '6c', '65', '20',
'64', '65', '20', '74', '65', '78', '74', '65', '20', '70', '6f', '75', '72', '20', '6c', '65', '20', '63', '68',
'69', '66', '66', '72', '65', '6d', '65', '6e', '74', '20', '64', '75', '20', '74', '69', '70', '65']
```

Annexe

```
171 # exemple avec virgules, tirets, caractères accentués et majuscules
172 texte_exemple_3 = "Pour donner un aspect professionnel à votre document, Word offre des conceptions d'en-tête, de
    pied de page, de page de garde et de zone de texte qui se complètent mutuellement. Vous pouvez par exemple
    ajouter une page de garde, un en-tête et une barre latérale identiques. Cliquez sur Insérer et sélectionnez les
    éléments de votre choix dans les différentes galeries."
173
174 # exemple plus long
175 # Le "Lorem ipsum" est écrit en latin fictif et est utilisé en imprimerie pour calibrer une mise en page, il est
    remplacé par le vrai texte dès que possible.
176 texte_exemple_lorem_ipsum = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse
    lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas
    ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est
    eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat
    in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum
    augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit
    sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh.
    Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit
    mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin.
    Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula.
    Pellentesque rhoncus nunc et augue. Integer id felis. Curabitur aliquet pellentesque diam. Integer quis metus
    vitae elit lobortis egestas. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi vel erat non mauris
    convallis vehicula. Nulla et sapien. Integer tortor tellus, aliquam faucibus, convallis id, congue eu, quam.
    Mauris ullamcorper felis vitae erat. Proin feugiat, augue non elementum posuere, metus purus iaculis lectus, et
    tristique ligula justo vitae magna. Aliquam convallis sollicitudin purus. Praesent aliquam, enim at fermentum
    mollis, ligula massa adipiscing nisl, ac euismod nibh nisl eu lectus. Fusce vulputate sem at sapien. Vivamus leo.
    Aliquam euismod libero eu enim. Nulla nec felis sed leo placerat imperdiet. Aenean suscipit nulla in justo.
    Suspendisse cursus rutrum augue. Nulla tincidunt tincidunt mi. Curabitur iaculis, lorem vel rhoncus faucibus,
    felis magna fermentum augue, et ultricies lacus lorem varius purus. Curabitur eu amet."
177
178
179 # fonction qui transforme la liste de caractères en matrice carrée
180 def liste2mat(liste):
181     n = len(liste)
182     taille_mat = m.ceil(n**0.5)  # partie entière supérieure
183     mat = [[0 for i in range(taille_mat)] for i in range(taille_mat)]
184     for _ in range(taille_mat**2-n):
185         liste.append('a')  # il y aura autant d'éléments dans liste et dans mat (on ajoute le caractère 'a'
    autant de fois qu'il le faut pour avoir une matrice carrée)
186     for i in range(taille_mat):
187         for j in range(taille_mat):
188             mat[i][j] = liste[i*taille_mat + j]
189     return mat
190
191 mat_exemple_1 = liste2mat(liste_exemple_1)  # renvoie [['e', 'x', 'e', 'm'], ['p', 'l', 'e', ' '], ['d', 'e', '
    ', 't'], ['e', 'x', 't', 'e']]
```

Annexe

```
193 mat_exemple_2 = liste2mat(liste_exemple_2)  # renvoie [['e', 'x', 'e', 'm', 'p', 'l', 'e'], [' ', 'd', 'e', ' ',
    't', 'e', 'x'], ['t', 'e', ' ', 'p', 'o', 'u', 'r'], [' ', 'l', 'e', ' ', 'c', 'h', 'i'], ['f', 'f', 'r', 'e',
    'm', 'e', 'n'], ['t', ' ', 'd', 'u', ' ', 't', 'i'], ['p', 'e', 'a', 'a', 'a', 'a', 'a']]
194
195 # fonction qui transforme une matrice carrée de caractères en liste de caractères
196 def mat2liste(mat):
197     n = len(mat)
198     liste = []
199     for i in range(n):
200         for j in range(n):
201             liste.append(mat[i][j])
202     return liste
203
204 # mat2liste(mat_exemple_1) renvoie bien ['e', 'x', 'e', 'm', 'p', 'l', 'e', ' ', 'd', 'e', ' ', 't', 'e', 'x',
    't', 'e']
205
206 # fonction qui transforme une matrice de caractères du langage courant en matrice d'hexadécimaux
207 def lang_courant2hexa_matrice(mat):
208     N = len(mat)
209     mat_hexa = [[0 for i in range(N)] for i in range(N)]
210     for i in range(N):
211         for j in range(N):
212             mat_hexa[i][j] = lang_courant2hexa_caractere(mat[i][j])
213             if len(mat_hexa[i][j]) == 1:
214                 mat_hexa[i][j] = '0' + mat_hexa[i][j]
215     return mat_hexa
216
217 matrice_hexa_exemple_1 = lang_courant2hexa_matrice(mat_exemple_1)  # renvoie [['65', '78', '65', '6d'], ['70',
    '6c', '65', '20'], ['64', '65', '20', '74'], ['65', '78', '74', '65']]
218
219 matrice_hexa_exemple_2 = lang_courant2hexa_matrice(mat_exemple_2)  # renvoie [['65', '78', '65', '6d', '70',
    '6c', '65'], ['20', '64', '65', '20', '74', '65', '78'], ['74', '65', '20', '70', '6f', '75', '72'], ['20', '6c',
    '65', '20', '63', '68', '69'], ['66', '66', '72', '65', '6d', '65', '6e'], ['74', '20', '64', '75', '20', '74',
    '69'], ['70', '65', '61', '61', '61', '61', '61']]
220
221 # fonction qui transforme un caractère hexadécimal en caractère du langage courant
222 def hexa2lang_courant_caractere(carac):
223     # liste_carac = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
    'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
224     # liste_hexadecimaux et liste_carac_acceptes sont des variables globales
225     n = len(liste_hexadecimaux)  # n = 256
226     for i in range(n):
227         if carac == liste_hexadecimaux[i]:
228             return liste_carac_acceptes[i]
```

Annexe

```
229 # problème pour revenir en langage courant, il faut utiliser les 256 caractères
230 # pas de problèmes pour les guillemets et caractère  d'échappement, par exemple :
231 # "ab\de'ef\''"[1] renvoie 'b'    "ab\de'ef\''"[2] renvoie '\\'    "ab\de'ef\''"[5] renvoie "'"
232 # "ab\de'ef\''"[7] renvoie 'f'    "ab\de'ef\''"[8] renvoie "'"     "ab\de'ef\''"[9] renvoie "'"
233
234 # fonction qui transforme une matrice d'hexadécimaux en matrice de caractères du langage courant
235 def hexa2lang_courant_matrice(mat):
236     N = len(mat)
237     res = [[0 for i in range(N)] for i in range(N)]
238     for i in range(N):
239         for j in range(N):
240             res[i][j] = hexa2lang_courant_caractere(mat[i][j])
241     return res
242
243 mat_carac_lang_courant_exemple_1 = hexa2lang_courant_matrice(matrice_hexa_exemple_1)  # renvoie bien [['e', 'x',
    'e', 'm'], ['p', 'l', 'e', ' '], ['d', 'e', ' ', 't'], ['e', 'x', 't', 'e']]
244
245 # fonction qui regroupe les caractères d'une matrice en une seule chaîne de caractère
246 def regroupe_matrice(mat):
247     N = len(mat)
248     texte = ''
249     for i in range(N):
250         for j in range(N):
251             texte += mat[i][j]
252     return texte
253
254 texte_carac_exemple_1 = regroupe_matrice(mat_carac_lang_courant_exemple_1)  # renvoie bien 'exemple de texte'
255
256
257
258 ##############################################################################################
259 ### fonctions utiles au chiffrement : subBytes, shiftRows, mixColumns
260
261 ### subBytes
262 # subBytes : fonction qui utilise S (dépendante de S) et qui est bijective, cette opération est une permutation
    sur [|0, 255|]
263 def subBytes(mat, N):
264     res = [[0 for i in range(N)] for i in range(N)]   # matrice à N lignes et N colonnes
265     for i in range(N):
266         for j in range(N):
267             res[i][j] = S[int(mat[i][j][0], 16)][int(mat[i][j][1], 16)]   # dans la matrice M(i,j), i est
    remplacé par nouveau_i = l'entier correspondant au premier caractère de M(i,j) et j par nouveau_j = l'entier
    correspondant au deuxième puis M(i,j) = S(nouveau_i, nouveau_j) et cette opération est bijective
268     return res
```

Annexe

```
print(subBytes(S,len(S)))
print(subBytes(matrice_hexa_exemple_1,len(matrice_hexa_exemple_1)))
matrice_sub_exemple_1 = subBytes(matrice_hexa_exemple_1,len(matrice_hexa_exemple_1))  # renvoie [['4d', 'bc', '4d', '3c'], ['51', '50', '4d', 'b7'], ['43', '4d', 'b7', '92'], ['4d', 'bc', '92', '4d']]
matrice_sub_exemple_2 = subBytes(matrice_hexa_exemple_2,len(matrice_hexa_exemple_2))

# fonction réciproque
def subBytes_reciproque(mat, N):
    res = [[0 for i in range(N)] for i in range(N)]
    for i in range(N):
        for j in range(N):
            res[i][j] = S_inv[int(mat[i][j][0], 16)][int(mat[i][j][1], 16)]
    return res

print(subBytes_reciproque(S,len(S)))
print(subBytes_reciproque(subBytes(S,len(S)),len(S)))      # renvoie bien S
print(subBytes_reciproque(matrice_sub_exemple_1,len(matrice_sub_exemple_1)))      # renvoie [['65', '78', '65', '6d'], ['70', '6c', '65', '20'], ['64', '65', '20', '74'], ['65', '78', '74', '65']] donc renvoie bien matrice_hexa_exemple_1
print(subBytes_reciproque(subBytes(matrice_hexa_exemple_1,len(matrice_hexa_exemple_1)),len(matrice_hexa_exemple_1)))      # renvoie bien matrice_hexa_exemple_1

### shiftRows
def shiftRows(mat, N):
    res = [[0 for i in range(N)] for i in range(N)]   # matrice à N lignes et N colonnes
    for i in range(N):
        for j in range(N):
            res[i][j] = mat[i][(j + i) % N]      # a % b donne le reste de la division euclidienne de a par b
    return res   # la ligne i est décalée de i cran vers la gauche pour tout i de 0 à (N-1) et cette opération est bijective donc tous les caractères hexadécimaux de la matrice de départ sont toujours tous présents une et une seule fois dans la matrice modifiée

print(shiftRows(S,len(S)))
print(shiftRows(matrice_sub_exemple_1,len(matrice_sub_exemple_1)))
matrice_shift_exemple_1 = shiftRows(matrice_sub_exemple_1,len(matrice_sub_exemple_1))  # renvoie [['4d', 'bc', '4d', '3c'], ['50', '4d', 'b7', '51'], ['b7', '92', '43', '4d'], ['4d', '4d', 'bc', '92']]

# fonction réciproque
def shiftRows_reciproque(mat, N):
    res = [[0 for i in range(N)] for i in range(N)]   # matrice à N lignes et N colonnes
    for i in range(N):
        for j in range(N):
            res[i][j] = mat[i][(j - i) % N]
    return res
```

Annexe

```
print(shiftRows_reciproque(S, len(S)))
print(shiftRows_reciproque(shiftRows(S, len(S)),len(S)))  # renvoie bien S
print(shiftRows(shiftRows_reciproque(S, len(S)),len(S)))  # renvoie bien S
print(shiftRows_reciproque(matrice_shift_exemple_1,len(matrice_shift_exemple_1)))      # renvoie [['4d', 'bc', '4d', '3c'], ['51', '50', '4d', 'b7'], ['43', '4d', 'b7', '92'], ['4d', 'bc', '92', '4d']] donc renvoie bien matrice_sub_exemple_1
print(shiftRows_reciproque(shiftRows(matrice_sub_exemple_1, len(matrice_sub_exemple_1)),len(matrice_sub_exemple_1)))  # renvoie bien matrice_sub_exemple_1

### mixColumns
def mixColumns(mat, N, numero_tour):      # N est la taille de la matrice
    res = [[0 for i in range(N)] for i in range(N)]       # on va décaler vers le haut chaque colonne j de j+numero_tour et la fonction est bijective
    for i in range(N):
        for j in range(N):
            res[i][j] = mat[(i+j+numero_tour) % N][j]
    return res

print(mixColumns(S,len(S),3))
print(mixColumns(matrice_shift_exemple_1,len(matrice_shift_exemple_1),3))
matrice_mix_exemple_1 = mixColumns(matrice_shift_exemple_1,len(matrice_shift_exemple_1),3)  # renvoie ['4d', 'bc', 'b7', '4d'], ['4d', '4d', '43', '92'], ['50', '92', 'bc', '3c'], ['b7', '4d', '4d', '51']]

# fonction réciproque
def mixColumns_reciproque(mat, N, numero_tour):      # N est la taille de la matrice      # si la fonction chiffrement utilise le numéro du tour k, la fonction déchiffrement doit utiliser le numéro (nb_total_de_tours - k)  ou alors on utilise une boucle qui va en sens inverse (3e entrée du range = -1) et on peut garder k
    res = [[0 for i in range(N)] for i in range(N)]       # on va décaler vers le bas chaque colonne j de j+numero_tour
    for i in range(N):
        for j in range(N):
            res[i][j] = mat[(i-j-numero_tour) % N][j]
    return res

print(mixColumns_reciproque(S, len(S), 3))
print(mixColumns_reciproque(mixColumns(S, len(S), 3),len(S),3))  # renvoie bien S
print(mixColumns(mixColumns_reciproque(S, len(S), 3),len(S),3))  # renvoie bien S
print(mixColumns_reciproque(matrice_mix_exemple_1,len(matrice_mix_exemple_1),3))  # renvoie [['4d', 'bc', '4d', '3c'], ['50', '4d', 'b7', '51'], ['b7', '92', '43', '4d'], ['4d', '4d', 'bc', '92']] donc renvoie bien matrice_shift_exemple_1
print(mixColumns_reciproque(mixColumns(matrice_shift_exemple_1, len(matrice_shift_exemple_1), 3),len(matrice_shift_exemple_1),3))  # renvoie bien matrice_shift_exemple_1

# transformations matricielles pour lier tous les éléments de la matrice entre eux
```

Annexe

```
341 # comme shiftRows mais pour les colonnes pour avoir une opération bijective, en décalant un certain nombre de
    fois qui dépend du numéro du tour
342
343 # départ : texte clair qu'on transforme en hexadécimal puis dans une matrice et on fait des opérations
    (bijectives) entre cette matrice et la matrice S de référence
344
345
346 ### chiffrement
347 # donne une matrice contenant les caractères hexadécimaux du texte chiffré à partir de la matrice contenant les
    caractères en langage courant du texte clair
348 def chiffrement_mat_claire_hexa_vers_mat_chiffree_hexa(mat, cle, taille_chiff, nombre_tours):
349     N = len(mat)
350     for i in range(nombre_tours):
351         mat = subBytes(mat, N)
352         mat = shiftRows(mat, N)
353         mat = mixColumns(mat, N, i)
354     mat = subBytes(mat, N)
355     mat = shiftRows(mat, N)
356     return mat
357 # la boucle fait S M0 S M1 S M2 ... S M(nombre_tours-1)      # S : shiftRows   M : mixColumns
358
359 print(chiffrement_mat_claire_hexa_vers_mat_chiffree_hexa(S, 0, 0, 10))
360 print(chiffrement_mat_claire_hexa_vers_mat_chiffree_hexa(matrice_hexa_exemple_1,0,0,10))  # renvoie [['e0', '5b',
    '0e', 'e0'], ['47', 'ca', 'dc', '0e'], ['ed', 'b4', 'e0', 'e0'], ['e0', '47', 'b4', 'e0']]
361
362 # donne une matrice contenant les caractères hexadécimaux du texte chiffré à partir du texte clair en langage
    courant
363 def chiffrement_texte_clair_vers_mat_hexa_chiffree(texte_clair, cle, taille_chiff, nombre_tours):
364     liste_sans_carac_speciaux = liste_sans_carac_spec(texte_clair)
365     print('liste_sans_carac_speciaux', liste_sans_carac_speciaux)
366     matrice_claire = liste2mat(liste_sans_carac_speciaux)
367     print('matrice_claire', matrice_claire)
368     matrice_claire_hexa = lang_courant2hexa_matrice(matrice_claire)
369     print('matrice_claire_hexa', matrice_claire_hexa)
370     matrice_chiffree_hexa = chiffrement_mat_claire_hexa_vers_mat_chiffree_hexa(matrice_claire_hexa, cle,
    taille_chiff, nombre_tours)
371     print('matrice_chiffree_hexa', matrice_chiffree_hexa)
372     # matrice_chiffree_carac_lang_courant = hexa2lang_courant_matrice(matrice_chiffree_hexa)
373     # print('matrice_chiffree_carac_lang_courant', matrice_chiffree_carac_lang_courant)
374     # texte_chiffre_carac_lang_courant = regroupe_matrice(matrice_chiffree_carac_lang_courant)
375     # return texte_chiffre_carac_lang_courant
376     return matrice_chiffree_hexa
377
378 mat_chiffree_hexa_exemple_1 = chiffrement_texte_clair_vers_mat_hexa_chiffree(texte_exemple_1, 0, 0, 10)  #
    renvoie [['e0', '5b', '0e', 'e0'], ['47', 'ca', 'dc', '0e'], ['ed', 'b4', 'e0', 'e0'], ['e0', '47', 'b4', 'e0']]
```

Annexe

```
379 texte_chiffre_hexa_exemple_1 = regroupe_matrice(mat_chiffree_hexa_exemple_1)  # renvoie
    'e05b0ee047cadc0eedb4e0e0e047b4e0'
380 matrice_chiffree_carac_lang_courant_exemple_1 = hexa2lang_courant_matrice(mat_chiffree_hexa_exemple_1)  # renvoie
    [['Ò', '[', '♫', 'Ò'], ['G', '╨', '▄', '♫'], ['Ý', '┤', 'Ò', 'Ò'], ['Ò', 'G', '┤', 'Ò']]
381 texte_chiffre_carac_lang_courant_exemple_1 = regroupe_matrice(matrice_chiffree_carac_lang_courant_exemple_1)  #
    renvoie 'Ò[♫ÒG╨▄♫Ý┤ÒÒÒG┤Ò'
382 # on peut le laisser en texte hexadécimal pour plus de lisibilité
383
384 # donne le texte chiffré en hexadécimal à partir du texte clair en langage courant
385 def chiffrement_texte_clair_vers_texte_hexa_chiffre(texte_clair, cle, taille_chiff, nombre_tours):
386     matrice_chiffree_hexa = chiffrement_texte_clair_vers_mat_hexa_chiffree(texte_clair, cle, taille_chiff,
    nombre_tours)
387     texte_chiffre_hexa = regroupe_matrice(matrice_chiffree_hexa)
388     return texte_chiffre_hexa
389
390 # chiffrement_texte_clair_vers_texte_hexa_chiffre(texte_exemple_1, 0, 0, 10) renvoie bien
    'e05b0ee047cadc0eedb4e0e0e047b4e0'
391 # chiffrement_texte_clair_vers_texte_hexa_chiffre(texte_exemple_2, 0, 0, 10) renvoie
    'b447cae0b4f7ede0edd4d45b2dedb429e06bf70e0ef7b40ee0e047d9e0cadcf7775bb45fb4e029dcb4e0e014f7e0d90e5f'
392 # On remarque que malgré le fait que le début des deux textes clairs soit identique, les deux textes chiffrés
    sont complètement différents.
393
394 # donne le texte chiffré en langage courant à partir du texte clair en langage courant
395 def chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_clair, cle, taille_chiff, nombre_tours):
396     matrice_chiffree_hexa = chiffrement_texte_clair_vers_mat_hexa_chiffree(texte_clair, cle, taille_chiff,
    nombre_tours)
397     matrice_chiffree_carac_lang_courant = hexa2lang_courant_matrice(matrice_chiffree_hexa)
398     texte_chiffre_carac_lang_courant = regroupe_matrice(matrice_chiffree_carac_lang_courant)
399     return texte_chiffre_carac_lang_courant
400
401 texte_chiffre_exemple_1 = chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_exemple_1, 0, 0,
    10)  # renvoie bien 'Ò[♫ÒG╨▄♫Ý┤ÒÒÒG┤Ò'
402 texte_chiffre_exemple_2 = chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_exemple_2, 0, 0,
    10)  # renvoie '┤G╨Ò┤ÝÒÝÉÉ[-Ý┤)Òk♫♫┤♫ÒÒG┘Ò╨▄w[┤_┤Ò)▄┤ÒÒ¶Ò┘♫_'
403 texte_chiffre_exemple_3 = chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_exemple_3, 0, 0,
    10)  # renvoie 'Ò;♫-;Ò-ÉÒw▄Å┤â▒ÒÒÒ┤┤wÒ┤[┤┤[É┘┤♫-)┤♫[┘Ò┤Ò_ÒÒ┘-k┤Ò-Ý┤┘Ò♫┤Ò▬▬ÅÒÒÒ-┤[♫-┘ÅÉÅÒe[Ý┘┤Å┤À-Ý┤♫-┤[c┤ÒÒ┤k;-╨[k-[Ò┤w♫ÒÒ[wÒF┤Å-ÒA-Ò_ÒÉ▄;wÒ▄Ò╨♫[┤♫Òâ[♫♫É┘wo┘ÝÒ♫-╨Ò┘Ò9Ò╨ÝÅ-
    ┤ÉÒ┘Ò┤♫F┤▄┤╨É[♫ÅÒÉâÉ┤ÝÒwÝ┤w┤e┤┤[♫ÅÃw┤┤-[┘┤İ¶w┤┤╨wÒ┤Ïü-)G-♫↔┤┤[ÅâÒw-
    ┘w┤É)É┤[Ò↔k♫┤ow┘○eÒ♫â┘ÏÒÅ┘┤ÉÖ[▄wÏ┤-[┤O┘ÒÒ┤Ò-┤ÉÒÝÝ┤ÒeÒÒw┤;♫▄;ÒkÀÒ-GÝ┤#♫▄É)wÒ\xadÒw┤ÒÅ┤┤♫ÅÉk▄Å♫↔┘)♫o┘┤▄┘Ò┤-
    ÒÅk▀Ò┘┘♫┘#┘Å┘ÒG┤┘Ò▄┤┘ÒÝâ┘ÏÒ┤'
```

Annexe

```
texte_chiffre_exemple_lorem_ipsum =
chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_exemple_lorem_ipsum, 0, 0, 10)  # renvoie 'À-
[illegible]

texte_chiffre_exemple_lorem_ipsum == texte_exemple_lorem_ipsum  # renvoie False


### efficacité
# chiffrer un texte avec des séquences qui se répète et voir s'il apparaît des séquence identiques dans le texte
chiffré
texte_exemple_sequences_identiques = 'exempleexempleexempleexempleexemple'
texte_chiffre_exemple_sequences_identiques =
chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_exemple_sequences_identiques, 0, 0, 10)  #
renvoie 'Ò[illegible]'
# On voit qu'il n'y a aucune séquence de longueur strictement supérieure à 1 qui se répète dans le texte chiffré.

# influence de la modification d'un caractère du texte clair sur le texte chiffré
# chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('exemple de texte', 0, 0, 10)  # renvoie
'Ò[illegible]'
# chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('axemple de texte', 0, 0, 10)  # renvoie
'Ò[illegible]'
# Pour un caractère modifié dans le texte clair, un seul caractère est aussi modifié dans le texte chiffré, cela
est dû au fait que les caractères sont chiffrés un par un, mais le caractère chiffré et sa position dans le texte
chiffré changent en fonction de la taille du texte.
```

Annexe

```
417 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('aexemple de texte', 0, 0, 10)  # renvoie
    '¸ÒGÒ╩▄Ò┤[Ý♫ÒGÒ┤¸¸♫Ò¸¸¸¸¸¸'
418 # Donc rajouter un caractère change tout au texte chiffré.
419
420 # influence du nombre de tours sur le chiffrement
421 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('exemple', 0, 0, 9)  # renvoie '&áôSáá▬&'
422 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('exemple', 0, 0, 10)  # renvoie '¸Ò¸╩▄ÝÒGÒ'
423 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('exemple', 0, 0, 11)  # renvoie 'táhßUßßhă'
424 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('exemple', 0, 0, 12)  # renvoie '°Ò°³DÆE°E'
425 # Donc à chaque tour dans le chiffrement, tout le texte est modifié
426 # On remarque aussi que la fréquence d'apparition des lettres est modifiée, par exemple le texte clair 'exemple'
    comporte trois 'e' et une seule fois tous les autres caractères alors que le texte chiffré '°Ò°³DÆE°E' comporte
    trois '°', deux 'E' et une seule fois tous les autres caractères
427
428 # évaluation de la compression : est-ce que les données occupent plus de place ?
429 # La taille d'un texte chiffré est le premier carré parfait suivant la taille du texte clair donc les tailles des
    textes clair et chiffré sont du même ordre de grandeur.
430
431 # On remarque que chaque caractères identiques d'un texte chiffré correspondent à des caractères identiques d'un
    texte clair donc il y a une relation statistique entre texte clair et texte chiffré ce qui est un problème pour
    la sécurité.
432
433 ### déchiffrement
434 def dechiffrement_mat_chiffree_hexa_vers_mat_claire_hexa(mat, cle, taille_chiff, nombre_tours):
435     N = len(mat)
436     mat = shiftRows_reciproque(mat, N)
437     mat = subBytes_reciproque(mat, N)
438     for i in range(nombre_tours-1, -1, -1):
439         mat = mixColumns_reciproque(mat, N, i)
440         mat = shiftRows_reciproque(mat, N)
441         mat = subBytes_reciproque(mat, N)
442     return mat
443 # la boucle fait Mr(nombre_tours-1) Sr Mr(nombre_tours-2) Sr ... Mr1 Sr Mr0 Sr          # S : shiftRows   M :
    mixColumns   r : reciproque
444
445 print(chiffrement_mat_claire_hexa_vers_mat_chiffree_hexa(S, 0, 0, 10))
446 print(dechiffrement_mat_chiffree_hexa_vers_mat_claire_hexa(chiffrement_mat_claire_hexa_vers_mat_chiffree_hexa(S,
    0, 0, 1), 0, 0, 1))      # redonne bien S
447 print(dechiffrement_mat_chiffree_hexa_vers_mat_claire_hexa(chiffrement_mat_claire_hexa_vers_mat_chiffree_hexa(S,
    0, 0, 10), 0, 0, 10))    # redonne bien S
448 print(dechiffrement_mat_chiffree_hexa_vers_mat_claire_hexa(chiffrement_mat_claire_hexa_vers_mat_chiffree_hexa(mat
    rice_hexa_exemple_1,0,0,10),0,0,10))   # redonne bien matrice_hexa_exemple_1
```

Annexe

```
450 matrice_dechiffree_hexa_exemple_1 =
    dechiffrement_mat_chiffree_hexa_vers_mat_claire_hexa(mat_chiffree_hexa_exemple_1, 0, 0, 10)  # renvoie bien
    [['65', '78', '65', '6d'], ['70', '6c', '65', '20'], ['64', '65', '20', '74'], ['65', '78', '74', '65']]
451 matrice_dechiffree_carac_lang_courant_exemple_1 = hexa2lang_courant_matrice(matrice_dechiffree_hexa_exemple_1)  #
    renvoie bien [['e', 'x', 'e', 'm'], ['p', 'l', 'e', ' '], ['d', 'e', ' ', 't'], ['e', 'x', 't', 'e']]
452 texte_dechiffre_carac_lang_courant_exemple_1 = regroupe_matrice(matrice_dechiffree_carac_lang_courant_exemple_1)
    # renvoie bien 'exemple de texte'
453
454 def dechiffrement_mat_chiffree_hexa_vers_texte_clair(mat, cle, taille_chiff, nombre_tours):
455     matrice_claire_hexa = dechiffrement_mat_chiffree_hexa_vers_mat_claire_hexa(mat, cle, taille_chiff,
    nombre_tours)
456     matrice_claire_carac_lang_courant = hexa2lang_courant_matrice(matrice_claire_hexa)
457     texte_clair_carac_lang_courant = regroupe_matrice(matrice_claire_carac_lang_courant)
458     return texte_clair_carac_lang_courant
459
460 # dechiffrement_mat_chiffree_hexa_vers_texte_clair(mat_chiffree_hexa_exemple_1, 0, 0, 10) renvoie bien 'exemple
    de texte'
461
462
463 # si on a une liste d'hexadécimaux à déchiffrer
464 def dechiffrement_liste_chiffree_hexa_vers_texte_clair(liste, cle, taille_chiff, nombre_tours):
465     mat = liste2mat(liste)
466     return dechiffrement_mat_chiffree_hexa_vers_texte_clair(mat, cle, taille_chiff, nombre_tours)
467
468 liste_hexa_a_dechiffrer = mat2liste(mat_chiffree_hexa_exemple_1)  # renvoie ['e0', '5b', '0e', 'e0', '47', 'ca',
    'dc', '0e', 'ed', 'b4', 'e0', 'e0', 'e0', '47', 'b4', 'e0']
469 # dechiffrement_liste_chiffree_hexa_vers_texte_clair(liste_hexa_a_dechiffrer, 0, 0, 10) renvoie bien 'exemple de
    texte'
470
471
472 # si on a un texte d'hexadécimaux à déchiffrer (donc le nombre de caractères n du texte est pair)
473 # renvoie une liste de chaînes de caractères de longueur 2 constituée des caracères du texte
474 def texte_hexa2liste(texte):
475     n = len(texte)
476     liste = []
477     for i in range(0, n, 2): # le pas de la boucle est 2
478         liste.append(texte[i] + texte[i+1])
479     return liste
480
481 # texte_hexa2liste(texte_chiffre_hexa_exemple_1) renvoie bien ['e0', '5b', '0e', 'e0', '47', 'ca', 'dc', '0e',
    'ed', 'b4', 'e0', 'e0', 'e0', '47', 'b4', 'e0']  car texte_chiffre_hexa_exemple_1 renvoie
    'e05b0ee047cadc0eedb4e0e0e047b4e0'
```

Annexe

```
483 def dechiffrement_texte_chiffre_hexa_vers_texte_clair(texte, cle, taille_chiff, nombre_tours):
484     liste_chiffree_hexa = texte_hexa2liste(texte)
485     return dechiffrement_liste_chiffree_hexa_vers_texte_clair(liste_chiffree_hexa, cle, taille_chiff,
    nombre_tours)
486
487 texte_hexa_a_dechiffrer = texte_chiffre_hexa_exemple_1  # renvoie 'e05b0ee047cadc0eedb4e0e0e047b4e0'
488 # dechiffrement_texte_chiffre_hexa_vers_texte_clair(texte_hexa_a_dechiffrer, 0, 0, 10) renvoie bien 'exemple de
    texte'
489
490
491 # si on a un texte de caractères du langage courant à déchiffrer
492 # cette fonction renvoie une liste constituée des caracères du texte
493 def texte2liste(texte):
494     n = len(texte)
495     liste = []
496     for i in range(0, n):
497         liste.append(texte[i])
498     return liste
499
500 def dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair(texte, cle, taille_chiff, nombre_tours):
501     liste_chiffree_carac_lang_courant = texte2liste(texte)
502     mat_chiffree_carac_lang_courant = liste2mat(liste_chiffree_carac_lang_courant)
503     mat_chiffree_hexa = lang_courant2hexa_matrice(mat_chiffree_carac_lang_courant)
504     return dechiffrement_mat_chiffree_hexa_vers_texte_clair(mat_chiffree_hexa, cle, taille_chiff, nombre_tours)
505
506 texte_carac_langage_courant_a_dechiffrer_exemple_1 =
    chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_exemple_1, 0, 0, 10)  # renvoie
    'Ò[♫ÒG╬▄♫Ÿ┤ÒÒÒG┤Ò'
507 texte_dechiffre_exemple_1 =
    dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair(texte_carac_langage_courant_a_dechiffrer_exemple_
    1, 0, 0, 10)  # renvoie bien 'exemple de texte'
508
509 texte_dechiffre_exemple_2 =
    dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair(texte_chiffre_exemple_2, 0, 0, 10)  # renvoie
    bien 'exemple de texte pour le chiffrement du tipeaaaaa'  (Les 'a' ont été ajouté lors du passage dans une
    matrice carrée mais si celui qui donne le message à déchiffrer donne aussi le nombre de caractères du message
    clair, alors on peut savoir combien de 'a' supprimer pour retrouver le message clair de départ)
510
511 texte_dechiffre_exemple_3 =
    dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair(texte_chiffre_exemple_3, 0, 0, 10)  # renvoie
    "Pour donner un aspect professionnel à votre document, Word offre des conceptions d'en┤tête, de pied de page, de
    page de garde et de zone de texte qui se complètent mutuellement. fous pouvez par exemple ajouter une page de
    garde, un en┤tête et une barre latérale identiques. Cliquez sur Insérer et sélectionnez les éléments de votre
    choix dans les différentes galeries.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
512 # problème avec le tiret - déchiffré en ┤ et le V déchiffré en f
```

Annexe

```
texte_dechiffre_exemple_lorem_ipsum = dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair(texte_chiffre_exemple_lorem_ipsum, 0, 0, 10)  # renvoie bien 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. festibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis. Curabitur aliquet pellentesque diam. Integer quis metus vitae elit lobortis egestas. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi vel erat non mauris convallis vehicula. Nulla et sapien. Integer tortor tellus, aliquam faucibus, convallis id, congue eu, quam. Mauris ullamcorper felis vitae erat. Proin feugiat, augue non elementum posuere, metus purus iaculis lectus, et tristique ligula justo vitae magna. Aliquam convallis sollicitudin purus. Praesent aliquam, enim at fermentum mollis, ligula massa adipiscing nisl, ac euismod nibh nisl eu lectus. Fusce vulputate sem at sapien. fivamus leo. Aliquam euismod libero eu enim. Nulla nec felis sed leo placerat imperdiet. Aenean suscipit nulla in justo. Suspendisse cursus rutrum augue. Nulla tincidunt tincidunt mi. Curabitur iaculis, lorem vel rhoncus faucibus, felis magna fermentum augue, et ultricies lacus lorem varius purus. Curabitur eu amet.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'









##################################################################################################
### Etude du temps de calcul et comparaison entre texte clair et chiffré

def temps_chiffrement(texte, cle, taille_chiff, nombre_tours):
    t1 = time()
    texte_chiffre = chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte, cle, taille_chiff, nombre_tours)
    t2 = time()
    return t2-t1

temps_exemple_1_10tours = temps_chiffrement(texte_exemple_1, 0, 0, 10)  # renvoie 0.0 puis 0.0010008811950683594
temps_exemple_lorem_ipsum_10tours = temps_chiffrement(texte_exemple_lorem_ipsum, 0, 0, 10)  # renvoie 0.16628265380859375 puis 0.13842201232910156 puis 0.14545369148254395
```

Annexe

```
temps_exemple_1_100tours = temps_chiffrement(texte_exemple_1, 0, 0, 100)  # renvoie 0.0 puis 0.013342618942260742
temps_exemple_lorem_ipsum_100tours = temps_chiffrement(texte_exemple_lorem_ipsum, 0, 0, 100)  # renvoie 0.39096570014953613 puis 0.3974947929382324 puis 0.38344526290893555

temps_exemple_1_1000tours = temps_chiffrement(texte_exemple_1, 0, 0, 1000)  # renvoie 0.04687809944152832 puis 0.031130075454711914
temps_exemple_lorem_ipsum_1000tours = temps_chiffrement(texte_exemple_lorem_ipsum, 0, 0, 1000)  # renvoie 2.9498066902160645 puis 2.86474347114563 puis 3.0419678688049316


def temps_moyen_chiffrement(texte, cle, taille_chiff, nombre_tours, nombre_chiffrements):
    temps = 0
    for _ in range(nombre_chiffrements):
        temps += temps_chiffrement(texte, cle, taille_chiff, nombre_tours)
    return temps/nombre_chiffrements

temps_moyen_exemple_1_1000tours_10chiffrements = temps_moyen_chiffrement(texte_exemple_1, 0, 0, 1000, 10)  # renvoie 0.04153449535369873 puis 0.041576933860778806 puis 0.03749039173126221
# temps_moyen_exemple_lorem_ipsum_1000tours_10chiffrements = temps_moyen_chiffrement(texte_exemple_lorem_ipsum, 0, 0, 1000, 10)  # renvoie 3.482820653915405 puis 3.6372624397277833 puis 3.4334664344787598


### graphique du temps de chiffrement en fonction du nombre de tours
def graph_temps_en_fonction_de_nombre_tours(texte, cle, taille_chiff):
    X = np.array([1, 10, 50, 100, 250, 500, 750, 1000]) # liste des nombres de tours
    Y = np.array([temps_moyen_chiffrement(texte, cle, taille_chiff, nombre_tours, 10) for nombre_tours in X]) # temps moyen sur 10 chiffrements
    plt.plot(X,Y)
    plt.title("Temps de chiffrement en fonction du nombre de tours")
    plt.xlabel("Nombre de tours")
    plt.ylabel("Temps (s)")
    plt.show()
# avec régression linéaire (droite tracée entre le premier et le dernier point)
    plt.plot(X, Y, 'o')
    plt.plot([X[0], X[-1]], [Y[0], Y[-1]])
    plt.show()

print(graph_temps_en_fonction_de_nombre_tours(texte_exemple_lorem_ipsum, 0, 0))

# donne une droite passant par l'origine donc le temps est proportionnel au nombre de tours, c'était évident au vu du programme mais on peut ainsi le vérifier graphiquement
```

Annexe

```
570 ### graphique du temps de chiffrement en fonction de la taille du texte
571
572 # fonction qui donne un texte de taille donnée à partir d'un texte
573 def texte_de_taille_donnee(texte, taille):
574     n = len(texte)
575     if taille <= n :
576         return texte[:taille]
577     return texte_de_taille_donnee(texte+texte, taille) # fonction récursive, on répète le texte s'il est trop
    court
578
579 # texte_exemple_1 renvoie 'exemple de texte'
580 # texte_de_taille_donnee(texte_exemple_1, 5) renvoie 'exemp' qui est bien de taille 5
581 # texte_de_taille_donnee(texte_exemple_1, 25) renvoie 'exemple de texteexemple d' qui est bien de taille 25
582 # texte_de_taille_donnee(texte_exemple_1, 50) renvoie 'exemple de texteexemple de texteexemple de texteex' qui
    est bien de taille 50
583
584 def graph_temps_en_fonction_de_taille_du_texte(texte, cle, taille_chiff, nombre_tours):
585     X = np.array([1, 10, 50, 100, 250, 500, 750, 1000]) # liste des tailles du texte à chiffrer
586     Y = np.array([temps_moyen_chiffrement(texte_de_taille_donnee(texte, taille), cle, taille_chiff, nombre_tours,
    10) for taille in X]) # temps moyen sur 10 chiffrements
587     plt.plot(X,Y)
588     plt.title("Temps de chiffrement en fonction de la taille du texte")
589     plt.xlabel("Taille du texte à chiffrer")
590     plt.ylabel("Temps (s)")
591     plt.show()
592 # avec régression linéaire (droite tracée entre le premier et le dernier point)
593     plt.plot(X, Y, 'o')
594     plt.plot([X[0], X[-1]], [Y[0], Y[-1]])
595     plt.show()
596
597 print(graph_temps_en_fonction_de_taille_du_texte(texte_exemple_lorem_ipsum, 0, 0, 100)) # 100 tours dans chaque
    chiffrement
598
599 # donne une droite passant par l'origine donc le temps de chiffrement est proportionnel à la taille du texte à
    chiffrer et la complexité est linéaire en la taille du texte à chiffrer donc cet algorithme de chiffrement est
    exploitable
600
```

Annexe

```
603 ### améliorations possibles ?
604 ## amélioration 1 : On pourrait chiffrer indépendamment des chaînes d'un certain nombre de caractères puis les
    concaténer, il faudrait donc ajouter une variable "taille_chiff" de la taille des chaînes à chiffrer qu'il serait
    indispensable de connaître pour le déchiffrement.
605 # par exemple pour texte_exemple_1 = 'exemple de texte' et taille_chiff = 5 , l'algorithme chiffre 'exemp' 'le
    de' ' text' puis 'e' et les concatène ce qui donne un résultat différent du chiffrement direct de 'exemple de
    texte'
606 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('exemp', 0, 0, 10)  renvoie '...╨.ŸÒGÒ'
607 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('le de', 0, 0, 10)  renvoie '...[.Ò┤Ò▄'
608 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(' text', 0, 0, 10)  renvoie '...G.♫Ò♫┤'
609 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('e', 0, 0, 10)  renvoie 'Ò'
610 # si on les concatène, on obtient '...╨.ŸÒGÒ...[.Ò┤Ò▄...G.♫Ò♫┤Ò'
611 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('exemple de texte', 0, 0, 10)  renvoie
    'Ò[♫ÒG╨▄♫Ÿ┤ÒÒÕG┤Ò' qui est bien différent des 4 résultats précédents concaténés
612 def decoupe(texte, taille_decoupe):
613     n = len(texte)
614     liste = []
615     taille_liste = n//taille_decoupe
616     for i in range(taille_liste):
617         liste.append(texte[i*taille_decoupe:(i+1)*taille_decoupe])
618     liste.append(texte[taille_liste*taille_decoupe:])
619     if liste[-1] == '' :
620         liste = liste[:-1]
621     return liste
622 # decoupe('exemple de texte', 5) renvoie ['exemp', 'le de', ' text', 'e']
623
624 def chiffrement_avec_taille_chiff(texte_clair, cle, taille_chiff, nombre_tours):
625     liste_claire = decoupe(texte_clair, taille_chiff)
626     texte_chiffre = ''
627     for i in range(len(liste_claire)):
628         texte_chiffre += chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(liste_claire[i], cle,
    taille_chiff, nombre_tours)
629     return texte_chiffre
630 # chiffrement_avec_taille_chiff('exemple de texte', 0, 5, 10) renvoie bien '...╨.ŸÒGÒ...[.Ò┤Ò▄...G.♫Ò♫┤Ò'
631 # Avec cette méthode, la robustesse est augmentée car la fréquence d'apparition des caractères n'est plus du tout
    la même entre texte clair et texte chiffré et le déchiffrement est plus compliqué car il faut connaître, en plus
    du nombre de tours, la taille de chiffrement utilisée pour pouvoir savoir exactement où séparer le texte chiffré
    avant de le déchiffrer avec la première méthode.
632
```

Annexe

```
633 def dechiffrement_avec_taille_chiff(texte_chiffre, cle, taille_chiff, nombre_tours):
634     taille_decoupe = m.ceil(taille_chiff**0.5)**2 # donne le carré parfait suivant taille_chiff car les textes
    sont mis dans une matrice carrée pour être chiffrés
635     liste_chiffree = decoupe(texte_chiffre, taille_decoupe)
636     texte_clair = ''
637     for i in range(len(liste_chiffree)):
638         texte_clair += dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair(liste_chiffree[i], cle,
    taille_chiff, nombre_tours)
639     return texte_clair
640 # Pour taille_chiff = 5 , on a taille_decoupe = 9
641 # decoupe('...ẗ.ẎÒGÒ...[.Ò┤Ò■...G.♫Ò♫┼Ò', 9) renvoie ['...ẗ.ẎÒGÒ', '...[.Ò┤Ò■', '...G.♫Ò♫┼', 'Ò']
642 # dechiffrement_avec_taille_chiff('...ẗ.ẎÒGÒ...[.Ò┤Ò■...G.♫Ò♫┼Ò', 0, 5, 10) renvoie 'exempaaaale deaaaa
    textaaaae'
643 # Les caractères 'a' ajoutés sont dus au passage du texte à chiffrer dans une matrice carrée, pour plus de
    lisibilité, on peut les remplacer par le caractère espace ' ' et il y aura seulement des espaces supplémentaires
    dans le texte déchiffré.
644 # Remarque : si on essaye de déchiffrer directement sans connaître taille_chiff, on obtient pas du tout le texte
    déchiffré, par exemple,
    dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair('...ẗ.ẎÒGÒ...[.Ò┤Ò■...G.♫Ò♫┼Ò', 0, 0, 10) renvoie
    'l® ®a®ee®e aaxpamte®d®e®eata®xaaaaaa'
645
646
647 ## amélioration 2 : ajouter une variable "cle" au chiffrement pour que le chiffrement dépende de cette clé de
    telle sorte que même si quelqu'un connaissait le programme de déchiffrement, il ne pourrait rien faire sans la
    clé
648 # L'ajout d'une clé pourrait donner des informations sur où couper le texte clair (en différentes parties de
    tailles différentes) et en combien de tours de chiffrement chiffrer chacune de ces parties (un nombre différent
    de tours pour chaque partie par exemple)
649
650 # cle = '0310052015' signifierait que les 03 premiers caractères sont chiffrés avec 10 tours, les 05 caractères
    suivants avec 20 tours et les caractères restants avec 15 tours.
651 # len(cle) = 10   cle[0:2] = '03'   cle[2:4] = '10'   cle[4:6] = '05'   cle[6:8] = '20'   cle[8:10] = '15'
652
653 def decoupe_par_2(texte):
654     return decoupe(texte, 2)
655 # decoupe_par_2('0310052015') renvoie ['03', '10', '05', '20', '15']
656
```

Annexe

```
657 def chiffrement_avec_cle(texte_clair, cle, taille_chiff, nombre_tours):
658     liste_cle = decoupe_par_2(cle) # cle doit être de taille paire et non multiple de 4
659     n = len(cle)  # donc len(liste_cle) = n//2
660     # les int(liste_cle[0]) premiers caractères sont chiffrés avec int(liste_cle[1]) tours, les int(liste_cle[2]) caractères suivants sont chiffrés avec int(liste_cle[3]) tours, ... jusque (n//2)-2 puis les caractères restants sont chiffrés avec int(liste_cle[-1]) tours
661     texte_chiffre = ''
662     indice = 0
663     for i in range(0, n//2 - 1, 2):
664         texte_chiffre += chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_clair[indice:indice+int(liste_cle[i])], cle, taille_chiff, int(liste_cle[i+1]))
665         indice += int(liste_cle[i])
666     texte_chiffre += chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre(texte_clair[indice:], cle, taille_chiff, int(liste_cle[-1]))
667     return texte_chiffre
668 # Pour cle = '0310052015' , on a liste_cle = ['03', '10', '05', '20', '15'] et il faut chiffrer texte_clair[0:03] avec 10 tours puis texte_clair[03:05] avec 20 tours puis le reste de texte_clair avec 15 tours
669 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('exe', 0, 0, 10)  renvoie 'ÒÒ¸G'
670 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('mple ', 0, 0, 20)  renvoie 'ŸJD╠%PPPP'
671 # chiffrement_texte_clair_vers_texte_carac_lang_courant_chiffre('de texte', 0, 0, 15)  renvoie 'ý▒æýÑæýA┴'
672 # chiffrement_avec_cle('exemple de texte', '0310052015', 0, 0) renvoie bien 'ÒÒ¸GYJD╠%PPPPý▒æýÑæýA┴'
673 ## Cette méthode est beaucoup plus robuste car il n'y a plus aucune relation statistique entre les caractères du texte clair et ceux du texte chiffré alors qu'il y en avait une sans utiliser de clé, c'est la confusion. Cela permet d'augmenter grandement la sécurité.
674 # confusion : pas de relation statistique entre texte clair et texte chiffré
675 # diffusion : le changement d'un caractère du texte clair a une influence sur tout le texte chiffré
676
677 # Pour le déchiffrement, il faut redécouper le texte chiffré avant de le déchiffrer mais on ne peut pas le découper aux mêmes endroits que lors du chiffrement, il faut découper aux carrés parfaits suivants les entiers de la clé
```

Annexe

```
678 def dechiffrement_avec_cle(texte_chiffre, cle, taille_chiff, nombre_tours):
679     liste_cle = decoupe_par_2(cle) # cle doit être de taille paire et non multiple de 4
680     n = len(cle)  # donc len(liste_cle) = n//2
681     # si on a liste_cle = ['03', '10', '05', '20', '15'] , il faut le changer en ['4', '10', '9', '20', '15'] car
    les morceaux de texte chiffrés l'ont été dans une matrice carrée
682     texte_clair = ''
683     indice = 0
684     for i in range(0, n//2 - 1, 2):
685         liste_cle[i] = str(m.ceil(int(liste_cle[i])**0.5)**2) # ceil est la partie entière entière, cette
    opération permet d'avoir le carré parfait suivant liste_cle[i] et en fait une chaîne de caractère comme les
    autres objets de la liste
686     for i in range(0, n//2 - 1, 2):
687         texte_clair +=
    dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair(texte_chiffre[indice:indice+int(liste_cle[i])],
    cle, taille_chiff, int(liste_cle[i+1]))
688         indice += int(liste_cle[i])
689     texte_clair += dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair(texte_chiffre[indice:], cle,
    taille_chiff, int(liste_cle[-1]))
690     return texte_clair
691 # dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair('ÒÒ¸G', 0, 0, 10)  renvoie 'exea'
692 # dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair('YJD╠%PPPP', 0, 0, 20)  renvoie 'mple aaaa'
693 # dechiffrement_texte_chiffre_carac_lang_courant_vers_texte_clair('ý▒æýÑæýA┴', 0, 0, 15)  renvoie 'de textea'
694 # dechiffrement_avec_cle('ÒÒ¸GYJD╠%PPPPý▒æýÑæýA┴', '0310052015', 0, 0) renvoie bien 'exeample aaaade textea'
695
696
697
698 ##############################################################################################
699 # si on veut regarder en moyenne la taille de la plus grande séquence identique entre le texte clair et le texte
    chiffré
700 # programme comparant les différences entre deux textes chiffrés par des méthodes différentes ou entre le texte
    clair et le chiffré (recherche des séquences identiques entre les deux textes) et qui donne la taille de la plus
    grande séquence identique entre texte_1 et texte_2
701 def comparaison_a_partir_de(texte_1, depart_1, texte_2, depart_2):
702     n1 = len(texte_1)
703     n2 = len(texte_2)
704     nb_carac_identiques = 0
705     i = depart_1
706     j = depart_2
707     while i<n1 and j<n2 and texte_1[i]==texte_2[j]:
708         nb_carac_identiques += 1
709         i += 1
710         j += 1
711     return nb_carac_identiques
```

Annexe

```
713 # comparaison_a_partir_de('abcdef', 0, 'abcdefghij', 0)  renvoie 6
714 # comparaison_a_partir_de('abcdef', 0, 'abcdefghij', 1)  renvoie 0
715 # comparaison_a_partir_de('abcdef', 2, 'abcdefghij', 2)  renvoie 4
716
717 def comparaison(texte_1, texte_2):
718     max_nb_carac_identiques = 0
719     for i in range(len(texte_1)):
720         for j in range(len(texte_2)):
721             nb_carac_identiques = comparaison_a_partir_de(texte_1, i, texte_2, j)
722             max_nb_carac_identiques = max(max_nb_carac_identiques, nb_carac_identiques)
723     return max_nb_carac_identiques
724
725 # comparaison('abcdef','abcdefghij')  renvoie 6
726 # comparaison('gfjabcdefghijku','jggfffffabcdefghihhhh')  renvoie 9
727 # comparaison('rtuabcdefghijk','abcdefghijkfgjf')  renvoie 11
```