TP
chiffrement polyalphabétique

Christophe Viroulaud

Terminale - NSI

Algo 21

Un chiffrement polyalphabétique consiste à :

- utiliser une clé de chiffrement composée de plusieurs lettres,
- recopier la clé de façon à obtenir une chaîne de la longueur du message.

B	R	A	V	O
N	S	I	N	S

Construire un algorithme de chiffrement polyalphabétique.

Remarque

Dans un souci de simplification, nous nous limiterons à des lettres majuscules. Chaque caractère sera donc :

- équivalent à une valeur ASCII comprise entre 65 (A) et 90 (Z),
- codé sur 7 bits.

Sommaire

Conversions

La première étape consiste à convertir une chaîne de caractère en sa correspondance binaire.

B	R	A	V	O
66	82	65	86	79
1000010	1010010	1000001	1010110	1001111
N	S	I	N	S
78	83	73	78	83
1001110	1010011	1001001	1001110	1010011

Tableau 1 – Conversion en binaire

Activité 1 :

1. Télécharger et extraire le dossier compressé **`TP-annexe-chiffrement.zip`** sur le site https://cviroulaud.github.io
2. Créer un le fichier **`tp_chiffrement.py`**
3. Écrire la fonction **`renverser(tab: list)`** $\rightarrow$ **`list`** qui renvoie un nouveau tableau miroir de **`tab`**
4. Écrire la fonction **`int_en_bin(nb: int)`** $\rightarrow$ **`list`** qui renvoie le tableau des bits de l'entier **`nb`**
5. Écrire la fonction **`bin_en_int(paquet: list)`** $\rightarrow$ **`int`** qui renvoie l'entier correspondant à la représentation binaire **`paquet`**.
6. Exécuter le fichier **`test_conversion.py`** pour tester les trois fonctions.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1  def renverser(tab: list) -> list:
2      """
3      renverse un tableau
4      """
5      l = len(tab)
6      res = [0 for _ in range(l)]
7      for i in range(l):
8          res[l-1-i] = tab[i]
9      return res
```

```
1  def int_en_bin(nb: int) -> list:
2      """
3      Convertit un entier en sa représentation binaire
4      """
5      q = nb
6      r = []
7      while q > 0:
8          r.append(q % 2)
9          q = q//2
10
11     return renverser(r)
```

```
1 def bin_en_int(paquet: list) -> int:
2     """
3     Convertit un paquet de bits en entier
4     """
5     entier = 0
6     for i in range(len(paquet)):
7         entier += paquet[i]*2**(len(paquet)-1-i)
8     return entier
```

Sommaire

Convertir la clé

Afin de pouvoir chiffrer le message lettre par lettre, il faut allonger la clé de la taille du message :

B	R	A	V	O
N	S	I	N	S

Activité 2 : Écrire la fonction `creer_cle_bin(cle: str, taille: int)` → `list` qui construit dans un tableau la version binaire de la clé, à la `taille` du message.

N	S	I	N	S
78	83	73	78	83
1001110	1010011	1001001	1001110	1010011

```
>>> creer_cle_bin("NSI", 5)
[[1, 0, 0, 1, 1, 1, 0],
 [1, 0, 1, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 0, 1],
 [1, 0, 0, 1, 1, 1, 0],
 [1, 0, 1, 0, 0, 1, 1]]
```

Code 1 – Conversion en binaire

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1  def creer_cle_bin(cle: str, taille: int) -> list:
2      """
3      crée la version binaire de la clé, de la taille
4      du message à chiffrer
5      """
6      cle_bin = []
7      for i in range(taille):
8          # si dépasse taille de la clé, revient à 0
9          lettre = cle[i % len(cle)]
10         cle_bin.append(int_en_bin(ord(lettre)))
11     return cle_bin
```

Sommaire

Appliquer le chiffrement XOR

La porte `xor` n'existe pas par défaut en Python. Il faut donc la créer.

Activité 3 :

1. Écrire la fonction `xor(x: int, y: int)` $\rightarrow$ `int`
2. Écrire alors la table de vérité de la porte logique `XOR`.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1  def xor(x: int, y: int) -> int:
2      if x == 0:
3          if y == 0:
4              return 0
5          else:
6              return 1
7      else:
8          if y == 0:
9              return 1
10         else:
11             return 0
```

```
1  print(f"0 xor 0 -> {xor(0, 0)}")
2  print(f"0 xor 1 -> {xor(0, 1)}")
3  print(f"1 xor 0 -> {xor(1, 0)}")
4  print(f"1 xor 1 -> {xor(1, 1)}")
```

La fonction créée permet de chiffrer la version binaire du message.

Message	1000010	1010010	1000001	1010110	1001111
$\oplus$	1001110	1010011	1001001	1001110	1010011
Chiffré	0001100	0000001	0001000	0011000	0011100

Tableau 2 – Application de la porte XOR

Activité 4 : Écrire la fonction `appliquer_xor(entree_bin: list, cle_bin: list)` $\rightarrow$ `list` qui renvoie un tableau de tableau représentant la version binaire chiffrée du message.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction


```
1  def appliquer_xor(entree_bin: list, cle_bin: list) -> list:
2      # préparation du tableau de tableau de sortie
3      sortie_bin = [[0 for _ in range(len(entree_bin[0]))]
4                    for _ in range(len(entree_bin))]
5
6      # application de xor sur chaque bit
7      for i in range(len(entree_bin)):
8          for j in range(len(entree_bin[0])):
9              sortie_bin[i][j] = xor(entree_bin[i][j],
10                                    cle_bin[i][j])
11
12     return sortie_bin
```

Sommaire

Chiffrement, déchiffrement

L'algorithme de chiffrement s'écrit alors :

- Convertir le message en binaire.
- Convertir la clé en binaire.
- Chiffrer le message en binaire.

Remarque

Il est inutile de convertir le message en caractères. C'est la version binaire qui transite jusqu'au destinataire.

Activité 5 : Écrire la fonction
`chiffrement(message: str, cle: str)` $\rightarrow$ `list`
qui applique l'algorithme précédent.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1  def chiffrement(message: str, cle: str) -> list:
2      # convertit le message en binaire
3      message_bin = []
4      for lettre in message:
5          message_bin.append(int_en_bin(ord(lettre)))
6
7      # convertit la clé en binaire
8      cle_bin = creer_cle_bin(cle, len(message))
9
10     # chiffre le message
11     secret_bin = appliquer_xor(message_bin, cle_bin)
12     return secret_bin
```

Pour le déchiffrement, le destinataire applique l'algorithme suivant :

- Convertir la clé en binaire.
- Déchiffrer le message en binaire.
- Convertir le message en caractères.

Activité 6 : Écrire la fonction `dechiffrement(secret_bin: list, cle: str)` $\rightarrow$ `str` qui applique l'algorithme précédent.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction


```
1  def dechiffrement(secret_bin: list, cle: str) -> str:
2      # convertit la clé en binaire
3      cle_bin = creer_cle_bin(cle, len(secret_bin))
4
5      # déchiffre le secret binaire
6      message_bin = appliquer_xor(secret_bin, cle_bin)
7
8      # convertit le message binaire en caractères
9      message = ""
10     for lettre_bin in message_bin:
11         message = message+chr(bin_en_int(lettre_bin))
12     return message
```

```
1  cle = "NSI"
2  s = chiffrement("BRAVO", cle)
3  print(dechiffrement(s, cle))
```

Sommaire

Opérateurs binaires

En Python, il est possible d'effectuer des opérations directement sur les bits des entiers.

$$\begin{array}{rcccccc} & 10_{10} = & 1 & 0 & 1 & 0 \\ \& & 8_{10} = & 1 & 0 & 0 & 0 \\ \hline & 8_{10} = & 1 & 0 & 0 & 0 \end{array}$$

```
1 >>> 10 & 8
2 8
```

Code 2 – **AND** binaire

$$\begin{array}{rcccc} 10_{10} = & 1 & 0 & 1 & 0 \\ \mid \quad 8_{10} = & 1 & 0 & 0 & 0 \\ \hline 10_{10} = & 1 & 0 & 1 & 0 \end{array}$$

```
1 >>> 10 | 8
2 10
```

Code 3 – `OR` binaire

$$\begin{array}{rcccc} 10_{10} = & 1 & 0 & 1 & 0 \\ \hat{} 8_{10} = & 1 & 0 & 0 & 0 \\ \hline 2_{10} = & 0 & 0 & 1 & 0 \end{array}$$

```
1 >>> 10 ^ 8
2 2
```

Code 4 – `XOR` binaire

B	R	A	V	O
66	82	65	86	79
N	S	I	N	S
78	83	73	78	83

Tableau 3 – Conversion en ASCII

Activité 7 : Écrire la fonction `str_en_int(texte: str, taille: int)` → `list` qui renvoie le tableau des valeurs ASCII d'un `texte`. La `taille` correspond à la longueur du message à chiffrer (5 dans notre exemple). Cette fonction sera utilisée pour convertir en entiers, le message mais également la clé.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction

```
1  def str_en_int(texte: str, taille: int) -> list:
2      """
3      renvoie le tableau des valeurs ASCII d'un texte
4      """
5      texte_ascii = [0 for _ in range(taille)]
6      for i in range(taille):
7          texte_ascii[i] = ord(texte[i % len(texte)])
8      return texte_ascii
```

```
1  >>> str_en_int("BRAVO",5)
2  [66, 82, 65, 86, 79]
3  >>> str_en_int("NSI", 5)
4  [78, 83, 73, 78, 83]
```

L'algorithme de chiffrement s'écrit alors :

- Convertir le message en entiers ASCII.
- Convertir la clé en entiers ASCII, de la même taille que le message.
- Chiffre les entiers ASCII en appliquant un `OU EXCLUSIF` binaire.

Activité 8 : Écrire la fonction `chiffrement(message: str, cle: str)` $\rightarrow$ `list` qui applique l'algorithme précédent.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

Correction


```
1  def chiffrement(message: str, cle: str) -> list:
2      # convertit le message en valeurs sASCII
3      message_ascii = str_en_int(message, len(message))
4
5      # convertit la clé en valeurs ASCII
6      cle_ascii = str_en_int(cle, len(message))
7
8      # chiffre les valeurs ASCII
9      secret_ascii = [0 for _ in range(len(message))]
10     for i in range(len(message_ascii)):
11         # opérateur binaire xor
12         secret_ascii[i] = message_ascii[i] ^ cle_ascii[i]
13     return secret_ascii
```

Activité 9 :

1. Écrire l'algorithme de déchiffrement.
2. Écrire la fonction `dechiffrement(secret_ascii: list, cle: str)` $\rightarrow$ `str` qui applique l'algorithme précédent.

Avant de regarder la correction

- Prendre le temps de réfléchir,
- Analyser les messages d'erreur,
- Demander au professeur.

- Convertir la clé en entier ASCII, de la taille du message.
- Déchiffrer les entiers ASCII du message secret.
- Convertir les entiers ASCII en lettres

```python
def dechiffrement(secret_ascii: list, cle: str) -> str:
    # convertit la clé en valeurs ASCII
    cle_ascii = str_en_int(cle, len(secret_ascii))

    # déchiffre les valeurs ASCII
    message_ascii = [0 for _ in range(len(secret_ascii))]
    for i in range(len(secret_ascii)):
        # opérateur binaire xor
        message_ascii[i] = secret_ascii[i] ^ cle_ascii[i]

    # convertit les valeurs ASCII en lettre
    message = ""
    for lettre_ascii in message_ascii:
        message = message+chr(lettre_ascii)
    return message
```

```python
cle = "NSI"
s = chiffrement("BRAVO", cle)
print(dechiffrement(s, cle))
```

Remarque

Le chiffrement par `OU EXCLUSIF` est faible si la clé n'est pas suffisamment longue et si, de plus, nous disposons de quelques informations sur le contenu du message.