Sécurisation des communications sur Internet

Exercices

Exercice 1 : Le code césar

- Le chiffrement par décalage (appelé aussi codage de César, en référence à Jules César qui utilisait cette technique pour ses correspondances militaires) consiste à choisir un entier n et à décaler chaque lettre du message initial de n lettres dans l'alphabet (en recommançant à "A" si le décalage fait dépasser "Z"). C'est l'entier n qui constitue la clé de chiffrement. La méthode de déchiffrment revient à décaler les lettres du message chiffré de n positions dans l'autre sens.

1. Compléter la fonction `cesar(message,n)` qui prend en paramètres une chaîne de caractère et un entier n. Cette fonction renvoie la chaîne de caractères chiffrée avec un décalage de n lettres "vers la droite".

In [8]:
```
def cesar_1(message, n):
    alphabet='abcdefghijklmnopqrstuvwxyz'
    speciaux=[' ']
    res=''









    return res

message='ils sont fous ces romains'
assert(cesar_1(message,3)=='lov vrqw irxv fhv urpdlqv')
```

2. En utilisant judicieusement la fonction précédente, décoder ce message : `'zsg uoizcwg bs gcbh dog awsil'`

In [1]:
```
#Réponse :
```

Exercice 2 : Le code césar v2

Dans l'exercice précédent, il n'y a que 25 clés disponibles. Améliorons un peu le chiffrement, tout en simplifiant le code précédent, à l'aide de la norme Unicode. Pour cela, chaque caractère(y compris les caractères spéciaux et accentués) du message à chiffrer sera décalé de n rangs vers la droite, ou n est un entier compris entre 1 et potentiellement $+\infty$...

Voici des extraits de la documentation python concernant les fonctions `ord()` et `chr()` :

- La fonction `ord()` renvoie le nombre entier représentant le code Unicode du caractère représenté par la chaîne donnée. Par exemple, `ord('a')` renvoie le nombre entier 97 et `ord('€')` (symbole euro) renvoie 8364. Il s'agit de l'inverse de `chr()`.
- La fonction `chr(i)` renvoie la chaîne représentant un caractère dont le code de caractère Unicode est le nombre entier `i`. Par exemple, `chr(97)` renvoie la chaîne de caractères `'a'`, tandis que `chr(8364)` renvoie `'€'`. Il s'agit de l'inverse de `ord()`.

1. On souhaite chiffrer chaque caractère du message avec la clé 8400. En utilisant les fonctions `ord()` et `car()`, écrire l'instruction qui permet de chiffrer `'e'`.

In [ ]:
```

```

2. Compléter la fonction `cesar_2(message,n)` qui prend en paramètres une chaîne de caractère et un entier n. Cette fonction renvoie la chaîne de caractères chiffrée avec un décalage de n "vers la droite".

In [70]:
```
def cesar_2(message,n):
    m=''




    return m
```

In [71]:
```
message="c'est formidable l'informatique"
assert(cesar_2(message,214)=='ĹýĻĥŊöļŅňŃĿÍķĸŁĻöŁýĿńļŅňŃķŊĿÑŋĻ')
```

3. Décoder le message suivant : `'ØúmĆĈúmĉĈmþüăĄćúĈmċöĈĉmąöćûĄþĈmĂþúĊĉmĆĈúmøúmĆĈúmĉĈmĈöþĈÃ'` . *Indice : On sait que la clé est inférieure à 165*

In [ ]:
```

```

Exercice 3 : XOR

Le but de cet exercice est de chiffrer un message à l'aide de l'opérateur XOR (voir cours). L'un des intérêts de cet opérateur est qu'il est réversible. En python, cette opération est possible avec le symbole `^` qui applique cet opérateur bit à bit.

Exemple : `78^84` renvoie `26` (ce qui correspond en binaire à 01010100 ⊕ 01001110 = 00011010)

1. Vérifier que cet opérateur est réversible.

In [ ]:
```

```

2. Une chaîne de caractères Unicode est une séquence de points de code, qui sont des nombres de 0 à 0x10FFFF (1 114 111 en décimal). Cette séquence de points de code doit être stockée en mémoire sous la forme d'un ensemble de unités de code, et les unités de code sont ensuite transposées en octets de 8 bits. Les règles de traduction d'une chaîne Unicode en une séquence d'octets sont appelées un encodage de caractères ou simplement un encodage. Python permet de travailler avec ces séquences d'octets grâce aux méthodes `encode()` , `decode()` et `bytes()`

Exemple 1 :

In [19]:
```
octets='à'.encode()
print(octets, type(octets))
```

```
b'\xc3\xa0' <class 'bytes'>
```

La méthode `.encode()` renvoie une version encodée(par défaut en utf-8) de la chaîne sous la forme d'un objet "bytes" (une "liste" d'octets, byte signifie octet en anglais). L'affichage de sortie est précédé d'un `b` (comme byte). On remarque ici que le caractère "à" est codé sur 2 octets : c3 et a0, ce qui en base 10 correspond aux entiers 195 et 160 :

In [20]:
```
print(octets[0], octets[1])
```

```
195 160
```

Exemple 2:

In [23]:
```
octets=bytes([195,160])
```

Out[23]:
```
b'\xc3\xa0'
```

Réciproquement, la fonction `bytes` convertit une liste d'entiers en octets. Il ne reste plus alors, qu'à utiliser la méthode `decode()` pour convertir ces octets en caractère(s) :

In [24]:
```
octets.decode()
```

Out[24]:
```
'à'
```

Compléter la foncton `xor(message,cle)` qui prend en paramètres deux listes d'octets, l'une représentant le message à chiffrer et l'autre la clé utilisée.Cette fonction renvoie le message chiffré sous forme d'octets.

Aide :

- Pour chaque octet du message, on applique l'opérateur `^` à l'octet de la clé que l'on aura étendue.
- On obtient une liste d'entiers que l'on renvoie sous forme d'octets.

In [3]:
```
def xor(message,cle):
    res=[]
    #parcourir le message et appliquer l'opérateur xor









    #renvoyer les octets obtenus
    return bytes(res)
```

In [4]:
```
#tests
M="L'INFORMATIQUE C'EST SUPER"
C="NSI"
secret= xor(M.encode(),C.encode())
assert (secret==b'\x02t\x00\x00\x15\x06\x1c\x1e\x08\x1a\x1a\x18\x1b\x16i\rt\x0c\x1d\x07i\x1d\x06\
x19\x0b\x01')
```

Out[4]:
```
b''
```

3. Quelle instruction, utilisant la fonction `xor` de la question précédente, permet de retrouver le message de départ ?

In [ ]:
```

```

Exercice 4 : Casser le chiffrement

En disposant de quelques informations pertinentes et à condition que la clé ne soit pas trop grande, il est possible de casser ce chiffrement en utilisant la force brute , c'est à dire tester toutes les possibilités.

Le message chiffré est `'\x0e6/+y;.< x-(7,,\x9b\xf0z48z:646<z*\x9a\xf3(64+<{'` . On sait aussi que :

- La clé utilisée contient 3 lettres majuscules de l'alphabet français.
- Le message décodé contient les octets `nse!` .

A l'aide de la fonction de l'exercice précédent, décrypter ce message et retrouver la clé utilisée :

In [1]:
```
secret=b'\x0e6/+y;.< x-(7,,\x9b\xf0z48z:646<z*\x9a\xf3(64+<{'
a='ABCDEFGHIJKLMNOPQRSTUVWXYZ'









#
```

Exercice 5 : Casser le chiffrement 😉👊👌

Même exercice que le précédent. Le message secret est

`\x00!A.\x14m=-A)\x00$ h\x11;\x12m<\x8b\xd8z\x158s>\x00)Mm!-\x155\x14?=-A>F"\x90\xf1A.\x14m%!\x044\x12c` .

On sait cette fois que le message en clair contient les octets `retourne` et que la clé est un mot de 5 lettres (des minuscules et des majuscules).

In [6]:
```
secret=b'\x00!A.\x14m=-A)\x00$ h\x11;\x12m<\x8b\xd8z\x158s>\x00)Mm!-\x155\x14?=-A>F"\x90\xf1A.\x14m%!\x044\x12c'
Aa='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'



#
```

Exercice 6: Diffie-Hellman

1. Compléter la fonction `cle(p,g,a)` qui prend en paramètres un nombre premier `p` , un nombre entier `g` inférieur à `p` et une clé privée entière `a` . Cette fonction renvoie la valeur d'une clé publique à l'aide du calcul utilisé dans le protocole de Diffie Hellman.

In [7]:
```
def cle(p,g,a):
    return
```

1. On suppose que le nombre premier p est égal à 23 et que g est égal à 5. Alice choisit 6 comme clé privée et Bob choisit 15. Calculer les clés publiques d'Alice et Bob et Vérifier que la clé secrète partagée est égale à 2.

In [2]:
```
P=23
G=5

Apriv=6
Bpriv=15
```

Exercice 7 : Diffie-Hellman

Alice et Bob choisissent le protocole de Diffie-Hellman pour s'envoyer une clé secrète qui leur servira ensuite à échanger des informations. Ils utilisent pour cela le nombre premier $P = 941$ et le nombre $G = 627$. Alice a choisi $a = 347$ comme clé privée et Bob a choisi $b = 781$. En vous aidant de la fonction précédente, déterminer la clé secrète qui sera partagée par Alice et Bob.

In [ ]:
```

```