TP de cryptographie symétrique

Exercice 1:

Le chiffrement de César

1. **Chiffrement:**

   (a) Ecrivez en python une fonction de chiffrement de César **ChiffrementCesar(k,m)** acceptant en entrée un entier $k \in \{1, 2, \ldots, 25\}$ et une chaîne m de caractères alphabétiques majuscules (caractère$\in \{A, B, C, \ldots, Z\}$). La sortie sera une chaîne de caractère c .
   indications:
   Utilisez les indications des points 5 et 6 du petit récapitulatif python à la fin de ce TD
   La commande Python **n%m** est le reste de la division euclidienne de l'entier **n** par l'entier **m**.

   (b) **Applications:**

      i. **ChiffrementCesar**(10,'MATHS')
      ii. **ChiffrementCesar**(5,'CRYPTO')

2. **Déchiffrement:**

   (a) Ecrivez en python une fonction de déchiffrement de César **DechiffrementCesar(k',c)** acceptant en entrée la clé **k'** est le message chiffré **c**. La sortie sera le message déchiffré **m**.

   (b) Application: **DechiffrementCesar**(7,'HSNVYPAOTL')

3. **Cryptanalyse:**
   c=OEMVSNO
   Déterminez le message en clair m sachant que l'emetteur de ce message utilise un chiffrement de César.

Exercice 2:

Le chiffrement de Vigénère

1. **Chiffrement:**

   (a) Ecrivez en python un programme de chiffrement de Vigénère **ChiffrementVigenere(k,m)** acceptant en entrée 2 chaînes de caractères alphabétiques majuscules (**k** est la clé de chiffrement et **m** est le message en clair) . La sortie sera une chaîne de caractères majuscules **c** correspondant au chiffrement du message **m** suivant la méthode de Vigénère en utilisant la clé **k**.

   (b) Applications:

      i. **ChiffrementVigenere**('CRYPTO','MATHEMATIQUES')
      ii. **ChiffrementVigenere**('SECRET','CRYPTOGRAPHIE')

2. **Déchiffrement:**

   (a) Ecrivez en python une fonction de déchiffrement de Vigénère **DechiffrementVigenere(k',c)** acceptant en entrée 2 chaînes de caractères alphabétiques majuscules (**k'** est la clé de déchiffrement et **c** est le message chiffré) . La sortie sera une chaîne de caractères majuscules **m** correspondant au déchiffrement du message **c** suivant la méthode de Vigénère en utilisant la clé **k**'.

   (b) Application: **DechiffrementVigenere**('SECRET','EEVYIFSXKHYXK')

3. **Cryptanalyse:**
   On connaît la longueur de la clé ($|k| = 4$) et le message chiffré (**c=XOZHDIMOYEL**)

   (a) Combien y a-t-il de clés **k** possibles?

   (b) On sait que le message en clair **m** est soit **EXPONENTIEL** soit **LOGARITHMES**. Quel est le message envoyé? Quelle est la clé **k**?

Exercice 3:

Permutations

Le chiffrement de César vu dans l'exercice 1 est trop facile à décrypter. On peut rendre plus difficile ce décryptage en utilisant une permutation quelconque de l'alphabet plutôt qu'une permutation circulaire comme c'est le cas pour le chiffrement de César. Une permutation de l'alphabet est une bijection **f** de l'alphabet dans lui-meme qui à toute lettre majuscule **x** de l'alphabet attribue une lettre majuscule notée **f(x)**. Le tableau suivant donne un exemple d'une telle permutation:

x	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
f(x)	C	W	H	O	R	L	F	M	U	Z	X	A	E	I	Y	J	K	D	V	N	P	Q	S	T	G	B

Cette permutation est entièrement définie par la chaîne de caractères suivante: 'CWHORLFMUZXAEIYJKDVNPQSTGB'
Avec une telle permutation **f**, chiffrer un message **m** consistera à remplacer chaque lettre de **m** par son image par **f**.

1. Combien y a-t-il de permutations de l'alphabet?
2. **Chiffrement:**
   (a) Ecrivez en python un programme de chiffrement utilisant une permutation de l'alphabet **ChiffrementPermutation(k,m)** acceptant en entrée une permutation **k** de l'alphabet sous la forme d'une chaîne de 26 caractères (cette permutation k est la clé de chiffrement) et une chaîne **m** de caractères alphabétiques majuscules (**m** est le message en clair). La sortie sera une chaîne de caractères majuscules **c** correspondant au chiffrement du message **m** suivant la permutation **k**.
   (b) Applications: On note **f** la permutation donnée en exemple au début de l'exercice.
      i. **ChiffrementPermutation(f,'PERMUTATION')**
      ii. **ChiffrementPermutation(f,'ARRANGEMENT')**
3. **Déchiffrement:**
   (a) Ecrivez en python un programme de déchiffrement **DechiffrementPermutation(k,m)** acceptant en entrée une permutation **k'** de l'alphabet sous la forme d'une chaîne de 26 caractères (cette permutation **k'** est la clé de déchiffrement) et une chaîne **c** de caractères alphabétiques majuscules (**c** est le message chiffré). La sortie sera une chaîne de caractères majuscules **m** correspondant au déchiffrement du message **c** suivant la permutation **k**'.
   (b) Application: **DechiffrementPermutation(f,'ORIYEWDRERIN')**
4. **Cryptanalyse**:
   Un texte de langue française a été chiffré en utilisant une permutation inconnue. Dans ce texte les espaces, la ponctuation, les apostrophes, les tirets et les accents ont été enlevés. On ne fait pas non plus la différence entre les minuscules et les majuscules. On sait que ce texte contient le mot MATHEMATIQUES. Saurez-vous décrypter ce message? (vous pouvez utiliser les commandes décrites dans le point 6b du récapitulatif python)

UXPFFPVTXFKGFSULKGHHPNXFTUGNLXUPDBUXSLPLUFLHPUXZLPNBFVDPYBXV

VPLUDPYUGVBXVHPNPDBMXPGUNXFBXUPNLTUBMBXDNPELUPBLNPVUPYBV

NPDWISXPFPNPVKGFMPUVBTXGFVNPEXVTUGTZLBFNDPVHBTWPHBTXZLPVV

GFTXVVLPVNLFPBEVTUBKTXGFPVVPFTXPDDPTGLTPKWGVPVPUBHPFPBVPVYU

GYUXPTPVSPGHPTUXZLPVGLBDSPEUXZLPVVBKWBFTZLPDPVLFPVDBVTUGFGH

XPDBFBMXSBTXGFDBUKWXTPKTLUPDPVRBXTVVGKXBLOKGHHPDBNPHGSUBYW

XPDPKGHHPUKPBLDGXFDXHYGTBYYPDDPFTDPKBDKLDVIYUPTPFTTBFNXVZLPD

POXSLXTPDXFVXSFXRXBFKPDBHGFGTGFXPNPDBMXPXFNXMXNLPDDPVBKKGHH

GNPFTVYGFTBFPHPFTNLFPBYYUPWPFVXGFXFTLXTXMPBYYUGOXHBTXMPTGLTP

YUBTXZLP

Python:

1. **fonction**:

```
def Somme(a,b):
    return a+b
```

2. **branchement conditionnel** (if ...):

```
def maxi(a,b):
    if a>=b:
        m=a
    else:
        m=b
    return m
```

3. **boucle for**:

```
def fact(n):# n est un entier positif
    f=1
    if n>0:
        for i in range(n): #i va de 0 à n-1
            f=f*(i+1)
    return f
```

4. **boucle while**:

```
def fact2(n):# n est un entier positif
    f=1
    i=1
    while i<=n:
        f=f*i
        i=i+1
    return f
```

5. **caractères**:
   En Python la commande **ord('A')** donne un entier correspondant au code ASCII de la lettre A (c'est l'entier 65).
   Inversement la commande **chr(90)** donne le caractère correspondant au code ASCII 90 (c'est le caractère Z)

6. **Les chaînes de caractères**:

   (a) Les commandes suivantes montrent comment définir une chaîne de caractères (string), comment obtenir la longueur d'une chaîne de caractères (len(ch)), comment accéder à un caractère particulier de la châine et comment concaténer deux chaînes.

```
In [21]: ch='ABCDE'

In [22]: len(ch)
Out[22]: 5

In [23]: ch[2]
Out[23]: 'C'

In [24]: nch=ch+'Z'

In [25]: nch
Out[25]: 'ABCDEZ'
```

(b) Les commandes suivantes vous permettont de programmer la fonction de décryptage du dernier exercice plus simplement:

```
In [3]: ch='AABBCCDDEEEEEEEF'

In [4]: ch.count('E')
Out[4]: 7

In [5]: ch.count('A')
Out[5]: 2

In [6]: ch.replace('E','F')
Out[6]: 'AABBCCDDFFFFFFF'

In [7]: ch.index('C')
Out[7]: 4
```

7. **Les entiers**:
En Python les entiers sont stockés sur la machine sous la forme de nombres en base 2. La valeur d'un entier n'est pas limitée par un nombre de bits fixé à l'avance (ce n'est pas le cas pour les réels) et peut s'étendre jusqu'à la limite de la mémoire disponible. En plus de l'addition (+), de la soustraction (-), et de la multiplication (*) Python dispose des opérations sur les nombres entiers suivantes:

(a) Le quotient euclidien de la division euclidienne de a par b: **a//b** (le résultat est un nombre entier).

(b) Le reste euclidien de la division euclidienne de a par b: **a%b**

(c) l'affichage de la représentation binaire d'un nombre entier en base 10: **bin**(x) . Le résultat est une chaîne de caractère précédé du préfixe 0b:

```
In [5]: int(0b110)
Out[5]: 6

In [6]: ch=bin(22)

In [7]: ch
Out[7]: '0b10110'

In [8]: len(ch)
Out[8]: 7

In [9]: ch[4]
Out[9]: '1'

In [10]: ch[1]
Out[10]: 'b'
```

On voit dans cette suite de commandes que la chaîne de caractère ch peut être directement utilisée comme une liste pour accéder aux chiffres de la représentations binaire du nombre 22. La longueur de l'entier x en base 2 est donnée par la commande **int.bit_length(x)**. La longueur d'une chaîne de caractères ch est donnée par **len(ch)**.

(d) L'opération inverse est **int(x)**: l'exécution de la commande int(0b110) donne le nombre entier 6.