Chiffrement symétrique

1 - Code de César

1.1 Principe de César

Exemple ci-dessous avec une clé valant 3(pour 3 lettres de décalage) :

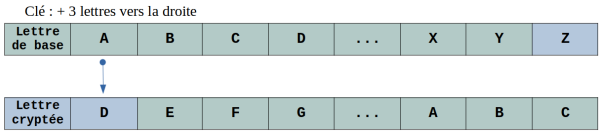

Figure 14.1 – César

Le code de César consiste tout simplement à décaler une lettre d'un certain nombre de cases vers la droite avant l'envoi. Pour décoder, il suffit de décaler de ce même nombre mais vers la gauche.

PRECONDITIONS : nous considérons ici qu'on doit fournir à nos fonctions des strings ne comportant que des MAJUSCULES, des espaces et des PONCTUATIONS autorisées. Le cours ne comporte donc pas la fonction de filtrage contrairement au TP.

```python
ALPHABET = ['A', 'B', 'C', 'D', 'E', ...  ... ... 'X', 'Y', 'Z']
PONCTUATION = [' ', '.', ' !', ' ?', ',', "'", ' ']

def trouve_indice(element :str, sequence :'str|list|tuple') -> 'int|None' :
    """Renvoie l'indice de element dans l'itérable sequence"""

    for i in range(len(sequence)) :  # Pour chaque indice possible
        if sequence[i] == element :  # Si la case i contient l'élément voulu
            return i                 # renvoie l'indice i

def decale(element :str, cle :int, sequence :list=ALPHABET) :
    """Renvoie la lettre chiffrant la lettre element en décalant de cle case vers la droite"""
    indice = trouve_indice(element, sequence)
    indice_decale = (indice + cle) % len(sequence)
    return sequence[indice_decale]

def chiffrement(message, cle, sequence=ALPHABET, ponctuation=PONCTUATION, debug=False) :
    """Renvoie le string en version chiffrée"""

    reponse = ''
    for caractere in message :
        if caractere in ponctuation :
            reponse = reponse + caractere
        elif caractere in sequence :
            crypte = decale(caractere, cle)
            reponse = reponse + crypte
            if debug : print(f"{caractere} devient {crypte}")
    return reponse

def dechiffrement(message, cle, sequence=ALPHABET, ponctuation=PONCTUATION) :
    """Renvoie le string en version déchiffrée"""

    return chiffrement(message, -cle, sequence, ponctuation)
```

1.2 Chiffrement symétrique

Un chiffrement est **symétrique** si on utilise une même clé pour chiffrer et déchiffrer.

1.3 Défauts de César

1) Pour communiquer, il faut se transmettre la clé en clair.

2) Il n'y a que 25 permutations possibles, facile de les faire toutes tester par un système informatique. C'est un problème FACILE.

2 - XOR bit à bit

2.1 Table de vérité du et/and

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

A RETENIR : le ET est VRAI uniquement lorsque toutes les entrées sont à VRAI.

2.2 Table de vérité du ou/or

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

A RETENIR : le OU est FAUX uniquement lorsque toutes les entrées sont à FAUX.

2.3 Table de vérité du ou exclusif/xor

a	b	a xor b
False	False	False
False	True	True
True	False	True
True	True	False

A RETENIR : le **xor** logique n'existe pas nativement en Python, si vous voulez l'utiliser, il faudra utiliser `not(a and b) and (a or b)`.

2.4 ASCII

ASCII permet d'associer 128 caractères à un nombre allant de 0 à 127. Exemple : `'A'` correspond à 65, le passage à la ligne correspond à 10.

2.5 Méthode du XOR

La méthode du XOR consiste à travailler directement sur les bits du message et les bits de la clé.

On notera qu'avec **une clé de X bits**, nous aurons 2^X possibilités de valeurs différentes. C'est un coût exponentiel, ce qui rend ce problème DIFFICILE à résoudre.

2.6 Exemple

— m caractérise les bits du message qu'on veut émettre.
— c caractérise les bits de la clé de chiffrement / déchiffrement. Si la clé c ne permet pas de couvrir tous les bits de m.
— mc caractérise les bits du message chiffré.
— md caractérise les bits du message déchiffré. Normalement, si tout se passe bien, nous devrions avoir $m = md$.

Imaginons qu'on veuille envoyer le message "BAC!" qui est encodé par l'ensemble d'octets qu'on peut obtenir en Python de cette façon :

```
>>> octets = [ord(c) for c in "BAC !"]
>>> octets
[66, 65, 67, 33]

>>> bits = [f"{bin(o)[2 :] :0>8}" for o in octets]
>>> bits
['01000010', '01000001', '01000011', '00100001']

>>> m = "".join(bits)
>>> m
'0100001001000001010000110010000 1'
```

Tentons de chiffrer ce texte en utilisant une clé à 5 bits 11001.

texte	B	A	C	!
valeur	66	65	67	33
m	0 1 0 0 0 0 1 0	0 1 0 0 0 0 0 1	0 1 0 0 0 0 1 1	0 0 1 0 0 0 0 1
c	0 1 1 0 0 1 . .	. . . . .	. . . . .	. . . . .
mc	. . . . .	. . . . .	. . . . .	. . . . .
c	. . . . .	. . . . .	. . . . .	. . . . .
md	. . . . .	. . . . .	. . . . .	. . . . .
valeur	.	.	.	.
texte	.	.	.	.