

# 15.2

## Le chiffrement symétrique

NSI TERMINALE - JB DUTHOIT

### 15.2.1 Un exemple de chiffrement symétrique, le chiffrement de César

L'une des méthodes les plus anciennes pour chiffrer un texte est attribuée à Jules César. Elle consiste à remplacer chaque lettre du message par la lettre située trois places plus loin dans l'alphabet. Par extension, on continue à appeler cette technique un chiffrement de César même si le décalage vaut autre chose que 3. On appelle le premier message le clair et celui obtenu après décalage le chiffré. Pour simplifier, on considère dans toute cette activité qu'on ne chiffre que les lettres majuscules non accentuées, les autres caractères (espaces inclus) restant inchangés.

#### Exercice 15.1

On s'intéresse ici au chiffrement de César, avec uniquement des lettres majuscules.

1. En considérant un décalage de 3, chiffrer le mot TERMINALE.
2. Quelle opération faut-il effectuer pour déchiffrer un message ? Déchiffrer alors le mot LQIRUPDWLTXH.
3. Créer une fonction python codage\_cesar( $k$ ,mot) qui effectue un chiffrement de César avec un décalage de  $k$  lettres sur mot. La fonction renvoie le mot chiffré.
4. Créer une fonction python decodage\_cesar( $k$ ,mot\_chiffre) qui effectue un déchiffrement de mot\_chiffre, sachant que mot\_chiffre a été chiffré avec un chiffrement de César avec un décalage de  $k$  lettres . La fonction renvoie le mot déchiffré.

### 15.2.2 Un autre chiffrement, le chiffrement de Vignere

Ici, la clef n'est pas un entier ,mais un mot :

Choisissons donc maintenant un "mot" qui nous servira de clé de chiffrement. Je choisis par exemple JB :

On va ensuite "recopier" autant de fois que nécessaire la clé pour avoir exactement autant de lettres que le mot à coder :

mot à coder	BONJOUR
clé	JBJBJB

Ensuite, nous allons coder la lettre B avec un décalage de 9 (pour la lettre J, la lettre O avec un décalage de 1 (pour la lettre B) , la lettre N avec un décalage de 9 ...etc...

mot à coder	BONJOUR
clé	JBJBJB
mot chiffré	

### Exercice 15.2

Créer une fonction `chiffrement_vignere(mot, cle)` qui prend en argument une chaîne de caractères `mot` et une chaîne de caractère `cle`, et qui renvoie le mot `mot` codé avec le chiffrement de Vignère avec la clé `cle`.

### Exercice 15.3

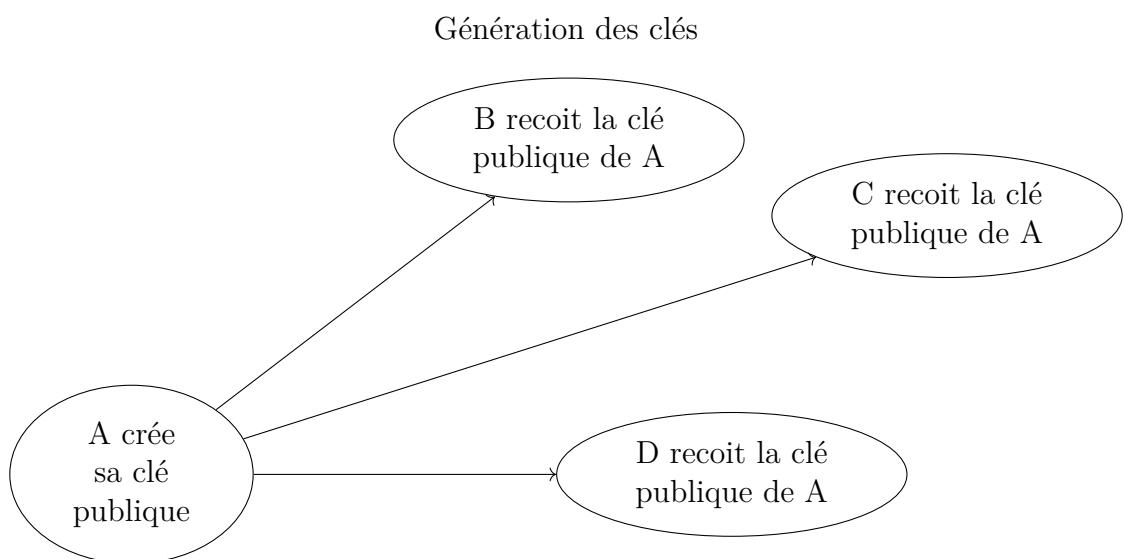
Créer une fonction `dechiffrement_vignere(mot, cle)` qui déchiffre un mot qui a été chiffrer avec le chiffrement de Vignère avec la clé `cle`

☞ Le gros problème avec le chiffrement symétrique, c'est qu'il est nécessaire pour A et B de se mettre d'accord à l'avance sur la clé qui sera utilisée lors des échanges (et aussi se mettre d'accord sur l'algorithme de chiffrement). Le chiffrement asymétrique permet d'éviter ce problème.

### 15.2.3 Principe du chiffrement symétrique

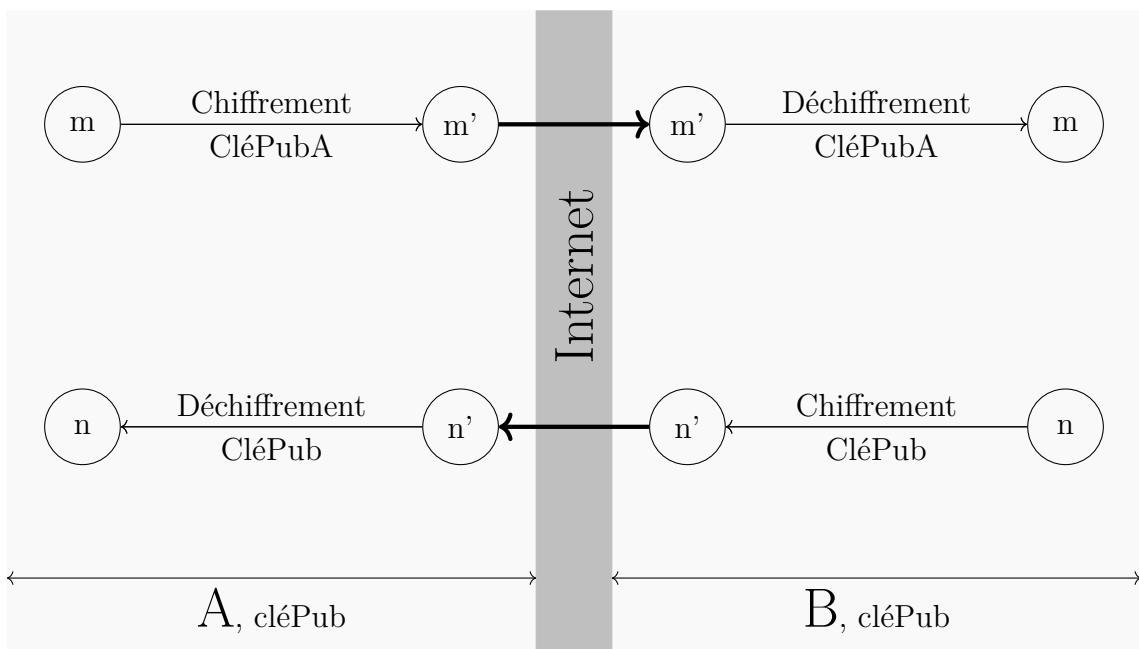
Supposons que A ait l'envie d'envoyer un message à B, C et D de façon sécurisée.

A doit créer une clé de chiffrement (clé publique) et la transmettre à B :



La clé va servir à chiffrer lors de l'envoi et à déchiffrer lors de la réception :

## Envoi et réception de message avec XOR



### 15.2.4 Un autre algorithme

Analysez et testez ce programme :

```
def chiffrement(mess ,cle):
    mess_code = ""
    position = 0
    for lettre in message:
        mess_code = mess_code + chr(ord(lettre) ^ cle[position])
        position = (position + 1) % len(cle)
    return mess_code
```

#### Remarque

Ici, la clé est exprimée sous la forme d'un tableau d'entiers et l'opérateur  $\wedge$  permet de faire un ou exclusif bit à bit. Par exemple :

66  $\wedge$  68 donne 6 car 66 s'écrit 1000010 et 68 1000100 donc l'"opération" ou exclusif bit à bit donne 0000110, soit le nombre 6...

#### Exercice 15.4

Utilisez les fonctions précédentes pour chiffrer la phrase "Veuillez transférer 1000 € sur mon compte offshore" en utilisant la clef "üšGtÜkGüZükWUüüwt".

Cette clef s'obtient en considérant les caractères dont l'unicode est [365, 353, 288, 355, 364, 489, 288, 371, 377, 365, 489, 372, 370, 361, 369, 373, 357].

#### Exercice 15.5

On considère le message suivant, que l'on va essayer de déchiffrer :

```
secret = '\x1a\x04$.\x19ix\x00*-\'\x05e9\x00 "V7±\x036+\x1fe;\x13e<\x9f&7\x12$?\x13ey'
```

On sait que les 4 derniers caractères du message en clair sont 'ge!'. On sait que la clef est composée de 3 lettres.

Déterminer le message en clair, ainsi que la clé utilisée.  
Chronométrier le temps mis pour trouver la réponse. \*\*\*

### Exercice 15.6

On utilise la méthode de chiffrement de Vigenère aux 95 caractères ASCII utilisables.

1. Vérifier qu'il existe environ cent million de clefs différentes comportant 4 caractères
2. Si l'on peut tester 1000 clefs par seconde, combien de temps est nécessaire pour déchiffrer un message par recherche exhaustive ?