

# 15.2

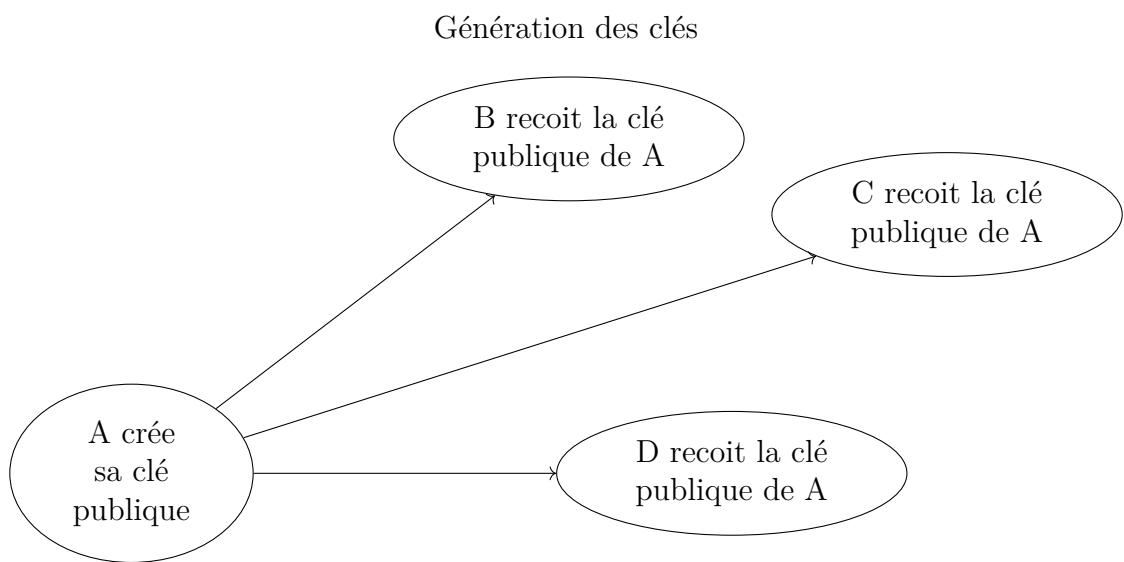
## Le chiffrement symétrique

NSI TERMINALE - JB DUTHOIT

### 15.2.1 Principe du chiffrement symétrique

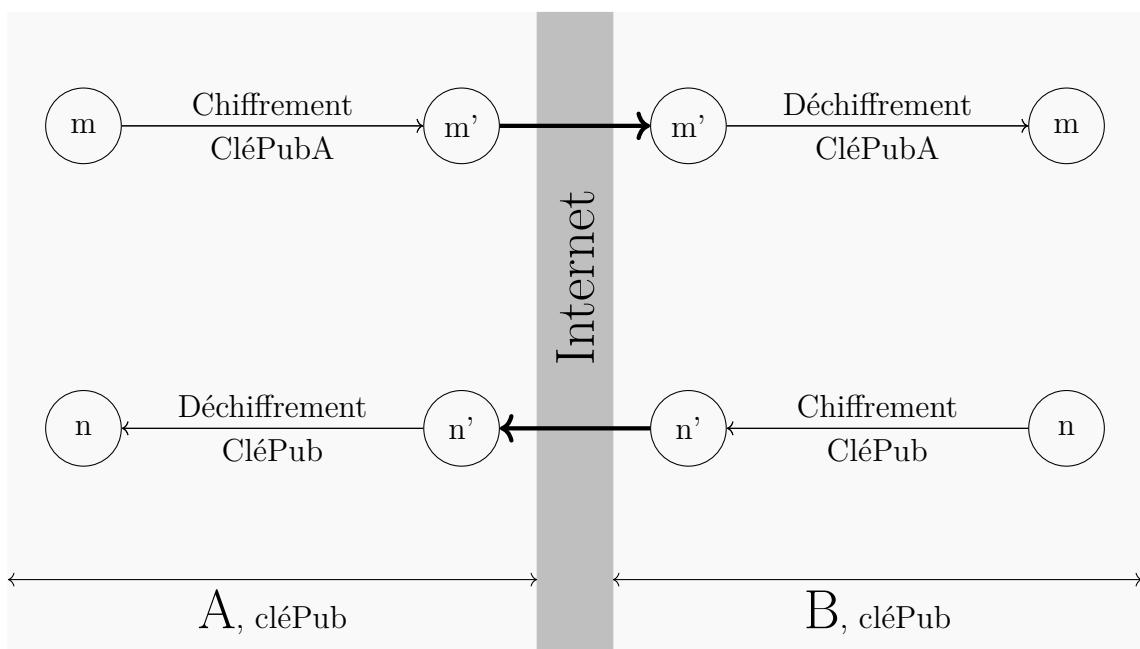
Supposons que  $A$  ait l'envie d'envoyer un message à  $B$ ,  $C$  et  $D$  de façon sécurisée.

$A$  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



## 15.2.2 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 `chiffrement_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 `dechiffrement_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.

### Exercice 15.2

Utiliser la force brute pour déchiffrer 'ZCWRLKKIRMZRCCVIGCLJ' message chiffré avec un chiffrement de César.

## 15.2.3 Le chiffrement de Vignère

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	B	O	N	J	O	U	R
clé	J	B	J	B	J	B	J

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 15.3

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 15.4

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

### Exercice 15.5

On utilise la méthode de chiffrement de Vignère 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 ?

## 15.2.4 Le chiffrement XOR

Analysez et testez ce programme :

```
def chiffrement(message,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 "`^`" permet de faire un ou exclusif bit à bit. Par exemple :  
66 `^` 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...
- Les caractéristiques de l'opérateur XOR, et le fait qu'il puisse être implémenté directement dans le processeur, font qu'il est souvent utilisé dans les algorithmes de chiffrement modernes, comme AES ou ChaCha20. Bien sûr ces algorithmes sont nettement plus complexes que la méthode naïve que nous avons utilisée, mais leurs principes reposent sur des fonctionnements similaires.
- En plus d'être relativement sûrs (voir ci-dessous), ces algorithmes sont très efficaces et permettent de chiffrer très rapidement. On peut ainsi chiffrer en direct des communications audio ou vidéo en temps réel.
- **⚠️** L'opérateur XOR étant réversible, la réitération de l'opération sur le message chiffré rendra le message original. Ainsi, la fonction qui permet de chiffrer permet également de déchiffrer !

### Exercice 15.6

Utilisez les fonctions précédentes pour chiffrer la phrase `Veuillez transférer 1000 euros sur mon compte off-shore` en utilisant la clef `ÿšGtÜkGyŽukWYüüwt`.

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 15.7**

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

`\x1b-z4 /&a)375;3z1$z#4?r-}=/z$$/&oz\x034;<%z=/z>$z! 3&mz;-z4 /&a;$.3 a67a9=4(3&?r%?r-?r%3 $zia+` 4`

On sait que le message contient le mot **courage**, et on sait aussi que la clef est composée de 3 lettres majuscules.

Déterminer le message en clair, ainsi que la clé utilisée.

\*\*\*

### 15.2.5 Un point sur les mots de passe

En informatique, la robustesse d'un mot de passe aléatoire est exprimée en terme d'entropie de Shannon, mesurée en bits.

D'après wikipedia, « au lieu de mesurer la robustesse par le nombre de combinaisons de caractères qu'il faut tester pour trouver le mot de passe avec certitude, on utilise le logarithme en base 2 de ce nombre. Cette mesure est appelée l'entropie du mot de passe. Un mot de passe avec une entropie de 42 bits calculée de la sorte serait aussi robuste qu'une chaîne de 42 bits choisie au hasard.

En d'autres termes, un mot de passe de 42 bits de robustesse ne serait brisé de façon certaine qu'après  $2^{42} = 439804651104$  tentatives lors d'une attaque par force brute. L'ajout d'un bit d'entropie à un mot de passe double le nombre de tentatives requises, ce qui rend la tâche de l'attaquant deux fois plus difficile.»

L'entropie d'un mot de passe de taille  $L$  utilisant des caractères parmi  $N$  possibilités est donné par la formule

$$H = L \cdot \log_2(N)$$

On constate donc qu'un mot de passe de 10 caractères latin majuscules est plus «résistant» qu'un mot de passe de 6 caractères utilisant n'importe quel symbole du clavier français... Cela signifie que le nombre de caractère est nettement plus important que la diversité de ceux-ci.

Nombre de caractères	Nombres seulement	Lettres minuscules	Lettres majuscules et minuscules	Nombres, lettres majuscules et minuscules	Nombres, lettres majuscules et minuscules, symboles
4	Immédiat	Immédiat	Immédiat	Immédiat	Immédiat
5	Immédiat	Immédiat	Immédiat	Immédiat	Immédiat
6	Immédiat	Immédiat	Immédiat	Immédiat	Immédiat
7	Immédiat	Immédiat	1 seconde	2 secondes	4 secondes
8	Immédiat	Immédiat	28 secondes	2 minutes	5 minutes
9	Immédiat	3 secondes	24 minutes	2 heures	6 heures
10	Immédiat	1 minute	21 heures	5 jours	2 semaines
11	Immédiat	32 minutes	1 mois	10 mois	3 ans
12	1 seconde	14 heures	6 ans	53 ans	226 ans
13	5 secondes	2 semaines	332 années	3 000 années	15 000 ans
14	52 secondes	1 an	17 000 ans	202 000 ans	1 million d'années
15	9 minutes	27 ans	898 000 ans	12 millions d'années	77 millions d'années
16	1 heure	713 ans	46 millions d'années	779 millions d'années	5 milliards d'années
17	14 heures	18 000 ans	2 milliards d'années	48 milliards d'années	380 milliards d'années
18	6 jours	481 000 ans	126 milliards d'années	1 trillion d'années	26 trillions d'années

Combien de temps votre mot de passe résiste-t-il à une attaque ?

### 15.2.6 l'inconvénient du chiffrement symétrique

☞ 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.