Chapitre 1 - Cryptologie : Partie 1

1- UN PEU DE VOCABULAIRE POUR COMMENCER :

La cryptologie est la science du secret (du grec *kruptos* qui signifie "caché"). Elle se compose de 2 disciplines :

- La **cryptographie,** qui comprend l'ensemble des méthodes de protection d'une information. Elle sert à garantir la confidentialité d'une information lors de communications ou de son stockage, en utilisant le chiffrement.
- L'autre discipline, la **cryptanalyse,** correspond aux méthodes utilisées pour analyser les messages chiffrés et "casser" la protection cryptographique de ces messages, afin de retrouver l'information claire sans connaître la clé de déchiffrement.

Autres points de vocabulaire :

- Le **chiffrement** est la transformation d'une information en clair en une information chiffrée, incompréhensible, mais que l'on peut déchiffrer avec une clé pour obtenir l'information en clair originale.
- Un **système de chiffrement** (ou *cryptosystème,* ou encore *chiffre*) est composé d'algorithmes de chiffrement et déchiffrement et d'une clé de chiffrement.

2- LES ORIGINES DE LA CRYPTOGRAPHIE :

a. LE CHIFFREMENT DE CESAR :

La cryptologie est utilisée depuis l'Antiquité, principalement dans le domaine militaire, pour éviter que les informations sur une armée ou sur un plan d'attaque ne tombent dans les mains de l'ennemi. La plus célèbre méthode de chiffrement de l'Antiquité est le chiffre de César, utilisé par Jules César. Cette méthode, aussi appelée **chiffrement par décalage**, consiste à décaler chaque lettre d'un message par la lettre de l'alphabet située à une distance fixée. Par exemple, si la distance est 3, la lettre A est remplacée par la lettre D, la lettre B par E, etc. ; et la lettre Z par C.

Par exemple, le message clair : ATTAQUEZ A L AUBE

est transformé, en utilisant le chiffre de César avec une distance de 3, en :

DWWDTXHC D O DXEH

La **distance** est un nombre compris entre 0 et 25**.** Si on choisit la distance 0, on ne change aucune lettre et le message chiffré reste identique au message clair, et donc lisible. L'alphabet latin étant composé de 26 lettres, si on choisit une distance de 26, on ne change aucune lettre. Au-delà de 26, le chiffrement ne sera pas différent de celui de la distance modulo 26. Par exemple, une distance de 29 donnera un message chiffré identique à une distance de 29 modulo 26 = 3.

- alphabet = "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"
- codage à la main :

A	T	T	A	Q	U	E	Z		A		L		A	U	B	E

Exercice 1. : Compléter le script de la fonction *chiffrementCesar()* donnée ci-contre. L'exécution de ce code donnera dans la console :

```
>>> (executing file "chriffrementDeCesar.py")
dwwdtxhc d o dxeh
attaquez a l aube
```

```
# script des fonctions
def chiffrementCesar(texte,cle,code) :
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    newTexte = ""
    ⇨ Script à compléter
    return newTexte

# programme principal
texteEnClair = "attaquez a l aube"

texteChiffre = chiffrementCesar(texteEnClair,3,True)
print(texteChiffre)

texteDeChiffre = chiffrementCesar(texteChiffre,3,False)
print(texteDeChiffre)
```

On rappelle ci-dessous quelques fonctions natives de pythons, utiles ici :

Point Cours :

- Méthode *index()* : retourne l'index de l'élément mis en argument …

  Si `mot = "cesaretcleopatre"`

  Alors `mot.index("a")` renvoie la valeur `3`

- Opérateur *modulo* `%` : renvoie le reste de la division euclidienne

4%26 =	28%26 =
26%26 =	0%26 =

⇨ Ecrire le code dans un fichier nommé *chiffrementDeCesar.py* .

CORRIGE

```
def chiffrementCesar(texte,cle,code) :
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    newTexte = ""
    if not code : cle = -cle
    for c in texte :
        if c != " ":
            i = alphabet.index(c)
            I = (i+cle)%26
            newC = alphabet[I]
        else : newC = c
        newTexte += newC
    return newTexte
```

b. ATTAQUE PAR FORCE BRUTE SUR UN TEXTE CHIFFRE EN « CESAR »:

La clé a seulement 26 valeurs possibles. Comme chaque valeur de clé donne un texte chiffré différent, il y a 26 messages chiffrés possibles pour un texte clair, selon la valeur de la clé.

Exercice 2. : Dans le même fichier *chiffrementDeCesar.py* , écrire le script d'une autre fonction nommée *forceBruteCesar()* . Cette fonction prend en argument un texte crypté « en César ». En l'exécutant, le texte décrypté pour les 26 valeurs différentes de clés, s'affiche dans la console.

Par exemple en exécutant dans la console

```
>>> forceBruteCesar("dwwdtxhc d o dxeh")
```

On obtient :

```
avec la cle 1 : cvvcswgb c n cwdg
avec la cle 2 : buubrvfa b m bvcf
avec la cle 3 : attaquez a l aube
avec la cle 4 : zsszptdy z k ztad
avec la cle 5 : yrryoscx y j yszc
avec la cle 6 : xqqxnrbw x i xryb
avec la cle 7 : wppwmqav w h wqxa
avec la cle 8 : voovlpzu v g vpwz
avec la cle 9 : unnukoyt u f uovy
avec la cle 10 : tmmtjnxs t e tnux
avec la cle 11 : sllsimwr s d smtw
avec la cle 12 : rkkrhlvq r c rlsv
avec la cle 13 : qjjqgkup q b qkru
avec la cle 14 : piipfjto p a pjqt
avec la cle 15 : ohhoeisn o z oips
avec la cle 16 : nggndhrm n y nhor
avec la cle 17 : mffmcgql m x mgnq
avec la cle 18 : leelbfpk l w lfmp
avec la cle 19 : kddkaeoj k v kelo
avec la cle 20 : jccjzdni j u jdkn
avec la cle 21 : ibbiycmh i t icjm
avec la cle 22 : haahxblg h s hbil
avec la cle 23 : gzzgwakf g r gahk
avec la cle 24 : fyyfvzje f q fzgj
avec la cle 25 : exxeuyid e p eyfi
```

En lisant les 26 textes, on constate que le seul texte qui a du sens est celui avec une valeur de clé égale à 3.

CORRIGE

```
def forceBruteCesar(texteChiffre) :
    for i in range(1,26):
        texteDeChiffre = chiffrementCesar(texteChiffre,i,False)
        print(f"avec la cle {i} : {texteDeChiffre}")
```

Exercice 3. : On a retrouvé dans de vieilles armoires du lycée, la phrase qui suit. Apparemment, elle est cryptée « en César » : « *jz cv evq uv tcvfgrkiv vlk vkv gclj tflik kflkv cr wrtv ul dfeuv rlirzk tyrexv*»

⇨ Décrypter ce texte et rechercher sur google son auteur, et le sens à lui donner (réponse attendue sous forme de commentaires dans le fichier *chiffrementDeCesar.py*

CORRIGE

```
>>> (executing file "chriffrementDeCesar.py")
jz cv evq uv tcvfgrkiv vlk vkv gclj tflik kflkv cr wrtv ul
dfeuv rlirzk tyrexv
si le nez de cleopatre eut ete plus court toute la face du
monde aurait change
```

⇨ Uploader ce fichier *chiffrementDeCesar.py* sur *nsibranly.fr* .

C. LE CHIFFREMENT PAR SUBSTITUTION :

Avec le chiffrement de César, les lettres sont toute décalées de la même valeur. Il devient ainsi facile de mener une attaque en force brute. Une variante de cet algorithme consiste à substituer chaque lettre par une autre lettre, définie au hasard. Par exemple la lettre A est remplacée par la lettre G, la lettre B par la lettre D, la lettre Z par la lettre M, etc.

On appelle cette méthode le **chiffrement par substitution,** et dans ce cas la clé correspond au tableau de substitution de chaque lettre de l'alphabet.

Cette clé est une permutation de 26 éléments, il y a donc "26!" (factoriel de 26) clés possibles, soit 2 puissance 88, ou encore 10 puissance 27 (1 suivi de 27 zéros). Ce nombre gigantesque de clés possibles rend **l'attaque par force brute impossible à réaliser** dans la pratique sans moyens informatiques.

Exercice 4. : Compléter le script de la fonction *genCle()* donnée ci-dessous :

Cette fonction n'a pas de paramètres, mais retourne un string de 26 caractères, constitué des 26 lettres de l'alphabet écrites dans un ordre aléatoire.

```
def genCle() :
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    liste = []
    cle = ""
    ⇨ Script à compléter
    return cle
```

Par exemple, une première exécution de *genCle()* pourrait renvoyer :

```
>>> genCle()
'kiseworzfctmjydqlubnhgxvap'
```

une seconde exécution de *genCle()* pourrait renvoyer :

```
>>> genCle()
'ruatvzhjxindlsgwbmqcykfeop'
```

CORRIGE

```
def genCle() :
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    liste = []
    cle = ""
    for c in alphabet :
        liste.append(c)
    while liste != [] :
        i = randint(0,len(liste)-1)
        c = liste.pop(i)
        cle = cle + c
    return cle
```

On rappelle ci-dessous quelques fonctions natives de python, utiles ici :

Point Cours :

- La fonction *randint()* de la bibliothèque random , permet de générer un nombre aléatoire entier :

```
from random import randint  :
randint(0,26)  :
```

- La méthode *pop()* native de python, permet d'enlever un élément d'une liste et de le renvoyer en retour :

```
>>> liste = ['a','b','c']

>>> liste.pop(1)
'b'

>>> liste
['a', 'c']
```

⇨ Ecrire le code de la fonction *genCle()* dans un fichier nommé *chiffrementParSubstitution.py* .

Exercice 5. : Dans le même fichier *chiffrementParSubstitution.py* , compléter le script de la fonction *chiffrementSubstitution()* donnée ci-dessous :

```
def chiffrementSubstitution(texte,cle,code) :
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    newTexte = ""
    ⇨ Script à compléter
    return newTexte

# Programme principal
cle = genCle()
print(f"clé utilisée : {cle}")
texteEnClair = "attaquez a l aube"

texteChiffre = chiffrementSubstitution(texteEnClair,cle,True)
print(texteChiffre)

texteDeChiffre = chiffrementSubstitution(texteChiffre,cle,False)
print(texteDeChiffre)
```

Cette fonction renvoie un texte crypté par substitution. La clé mise en argument aura été générée par la fonction *cleGen()* précédente.

L'exécution de ce code pourrait donner dans la console :

```
>>> (executing file "chiffrementParSubstitution.py")
clé utilisée : ruotvfiawsbpjxezlndchmqygk
rccrlhvk r p rhuv
attaquez a l aube
```

⇨ Uploader ce fichier *chiffrementDeCesar.py* sur *nsibranly.fr* .

CORRIGE

```
def chiffrementSubstitution(texte,cle,code) :
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    newTexte = ""
    for c in texte :
        if c != " ":
            if code :
                i = alphabet.index(c)
                newC = cle[i]
            else :
                i = cle.index(c)
                newC = alphabet[i]
        else : newC = c
        newTexte = newTexte + newC
    return newTexte
```

Pour information : Ce chiffrement est moins sujet à une attaque par force brute que celui de César. Il reste par contre sujet à un autre type d'attaque : **l'analyse fréquentielle des lettres.** Dans la langue française, comme dans toutes les langues, certaines lettres sont plus fréquentes que d'autres. Le E par exemple est la lettre la plus courante, suivie par le S, alors que le W et le K sont les moins courantes. Avec le chiffrement par décalage, chaque lettre est toujours remplacée par la même lettre, en appliquant le décalage par une distance fixée. Ainsi, la fréquence d'apparition des lettres dans un message chiffré reste la même que pour le message clair.

En comptant le nombre d'occurrences de chaque lettre dans le message chiffré, vous voyez quelles lettres sont les plus fréquentes et lesquelles sont les moins fréquentes. Vous pouvez ainsi déduire que les lettres les plus fréquentes correspondent aux lettres les plus fréquentes de la langue française. Par exemple si le V est la lettre la plus fréquente dans le texte chiffré, vous pouvez déduire que la lettre E est permutée par la lettre V, etc.

Vous pouvez faire la même analyse fréquentielle sur les **digrammes,** c'est à dire les occurrences de 2 lettres de suite. En français, le digramme le plus fréquent est "ES", suivi de "LE". Avec un message chiffré

suffisamment long, il y a forcément des motifs reconnaissables dans le texte chiffré qui permettent de le décrypter.

Ce type d'attaque s'appelle une **attaque par "texte chiffré seulement".** C'est la pire des attaques sur un système de chiffrement. Cela signifie qu'un attaquant peut retrouver le message en clair et la clé de déchiffrement simplement en observant les textes chiffrés, sans rien connaître de plus.

d. LE CHIFFREMENT DE VIGENERE :

Un autre algorithme célèbre est le chiffre de Vigenère, inventé au XVIe siècle. Il reprend en partie le principe de substitution, mais en variant la distance de décalage au cours du chiffrement en utilisant un mot ou une phrase comme clé. Chaque lettre de la clé correspond à sa position dans l'alphabet. Pour chiffrer un message, on écrit le texte clair et on écrit la clé en dessous, en répétant la clé autant de fois que nécessaire pour couvrir l'ensemble du message.

Par conséquent :

- le chiffrement consiste à additionner chaque lettre du message avec la lettre de la clé en dessous, modulo 26 ;
- le déchiffrement consiste à soustraire chaque lettre du message chiffré avec la lettre de la clé en dessous, modulo 26.

Par exemple, avec la clé "RABELAIS" et le message clair :

« SCIENCE SANS CONSCIENCE N EST QUE RUINE DE L AME »

on obtient : « JCJIYCM KRNT GZNAUZEOGP N MKK QVI CUQFV DF P LMM » .

- alphabet = "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"
- codage à la main :

S	C	I	E	N	C	E		S	A	N	S		C	O	N	S	C	I	E	N	C	E		N		E	S	T		Q	U	E		R	U	I	N	E		D	E		L		A	M	E
R	A	B	E	L	A	I		S	R	A	B		E	L	A	I	S	R	A	B	E	L		A		I	S	R		A	B	E		L	A	I	S	R		A	B		E		L	A	I

Exercice 6. : Compléter le script de la fonction *chiffrementVigenere()* donnée ci-contre :

```
# script des fonctions
def chiffrementVigenere(texte,cle,code) :
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    newTexte = ""

    ⇨ Script à compléter

    return newTexte

# programme principal
texteEnClair = "science sans conscience n est que ruine de l ame"
cle = "rabelais"


texteChiffre = chiffrementVigenere(texteEnClair,cle,True)
print(texteChiffre)
texteDeChiffre = chiffrementVigenere(texteChiffre,cle,False)
print(texteDeChiffre)
```

L'exécution de ce code donnera dans la console :

```
>>> (executing file "chriffrementDeVigenere.py")
jcjiycm krnt gznauzeogp n mkk qvi cuqfv df p lmm
science sans conscience n est que ruine de l ame
```

⇨ Ecrire le code dans un fichier nommé *chiffrementParVigenere.py* .

CORRIGE

```
def chiffrementVigenere(texte,cle,code) :
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    newTexte = ""
    num = 0
    for c in texte :
        if c != " ":
            i = alphabet.index(c)
            j = alphabet.index(cle[num])
            if code : cCrypt = alphabet[(i+j)%26]
            else : cCrypt = alphabet[(i-j)%26]
            newTexte = newTexte + cCrypt
            num = num  +  1
            if num == len(cle) :
                num = num - len(cle)
        else :
            newTexte = newTexte + c
    return newTexte
```

⇨ Uploader ce fichier *chiffrementParVigenere.py* sur *nsibranly.fr* .

Un algorithme peu résistant : Cet algorithme a résisté à la cryptanalyse jusqu'au XVIIIe siècle, mais a fini par être cassé avec la méthode suivante :

1. Supposez d'abord que vous connaissez la longueur de la clé.
2. Découpez le texte en parties du même nombre de lettres que la clé. La première lettre de chaque partie est toujours chiffrée avec la même lettre, la première lettre de la clé.
3. Vous pouvez donc réaliser une analyse fréquentielle, comme pour le chiffre de César, en observant que la lettre la plus courante correspond à la lettre E suivie de la lettre S, etc. Cela vous donne la première lettre de la clé.
4. Faites de même avec la 2e lettre, et ainsi de suite pour retrouver toutes les lettres de la clé et déchiffrer entièrement le message.

Si vous ne connaissez pas la longueur de la clé, supposez d'abord que la clé a une longueur de 1, et réalisez l'analyse fréquentielle. Si celle-ci ne donne pas de résultat, augmentez la longueur de la clé de 1, recommencez l'analyse fréquentielle et ainsi de suite. Lorsque la longueur de clé testée est la bonne, l'analyse fréquentielle fonctionnera et vous donnera la clé et le texte clair.

À partir du XIXe siècle, on utilise des machines mécaniques à cylindres de plus en plus complexes basées sur le principe de **substitution polyalphabétique** comme le chiffre de Vigenère. La plus connue de ces machines est la **machine Enigma,** utilisée par les Allemands durant la 2de guerre mondiale, et cassée par les Alliés (voir bande annonce du film « Imitation Game »

3- LES PRINCIPES FONDATEURS DE LA CRYPTOGRAPHIE MODERNE :

À la fin du XIXe siècle, Auguste Kerckhoffs, un cryptologue militaire, énonce les principes fondamentaux de la cryptographie moderne, les **principes de Kerckhoffs** :

- la sécurité d'un algorithme de chiffrement ne doit pas être basée sur la non-connaissance par l'attaquant de l'algorithme de chiffrement ou du système utilisé. Elle doit uniquement être basée sur le fait que **l'attaquant ne connaît pas la clé** ;
- pour être sécurisés, les algorithmes de chiffrement doivent être **physiquement** ou **mathématiquement impossibles à résoudre pour l'attaquant** ;
- le système doit être **adapté à son utilisation pratique** et la **clé** doit facilement être **modifiable.**

Le principe le plus important est le premier. Il a été reformulé par Claude Shannon, le fondateur de la théorie de l'information et de la cryptographie scientifique, comme ceci : **"l'adversaire connaît le système"**.

Ce principe est non seulement vrai en cryptographie, mais aussi dans la sécurité informatique en général. Ainsi, la sécurité d'un algorithme ne doit jamais être basée sur le fait que l'attaquant ne connaît pas le code source de l'algorithme. Il est possible, par la rétro-ingénierie, de retrouver le code source d'un algorithme à partir de code compilé ou simplement en observant les résultats d'un programme.