

APPLICATIONS DES MATHEMATIQUES



Cryptographie Partie 1

Programmation en Python 3

AVANT-PROPOS

- Ce document a été conçu pour l'enseignement de l'applications des mathématiques dispensé au *Collège de Genève*. Cela dit, il peut servir de support de cours pour d'autres filières d'enseignement.
- Vous trouverez dans ce chapitre de la **théorie** (définitions, théorèmes, démonstrations, etc.) et des **exercices** qui vous permettront progressivement de vous familiariser et de maîtriser les diverses notations et concepts en lien avec : les mathématiques, l'informatique et la physique. À la fin du chapitre se trouvent les **corrections des exercices**.
- Des **QR CODES** apparaissent à certains endroits du cours. Une fois scannés avec vos smartphones, ils donnent (aux personnes ayant un compte EDUGE) accès à la lecture de vidéos dont le contenu est en lien avec certains sujets du cours.

Vous pouvez télécharger ce document au format PDF à l'adresse suivante :

<https://www.sismondi.ch/disciplines/applications-des-mathematiques/cours-eleves>



Table des matières

1.3 Cryptographie et cryptanalyse	1
1.3.1 Introduction et terminologie	1
1.3.2 Fonctions de chiffrement, déchiffrement et clefs	3
1.3.3 Chiffrement par décalage (de César)	5
1.3.4 Chiffrement affine	11
1.3.5 Chiffrement par substitution	18
1.3.6 Chiffrement de Vigenère	29
1.3.7 Concept de chiffrement symétrique	37
1.3.8 Concept de chiffrement asymétrique	39
1.4 Corrections des exercices et activités	41

1.3 Cryptographie et cryptanalyse

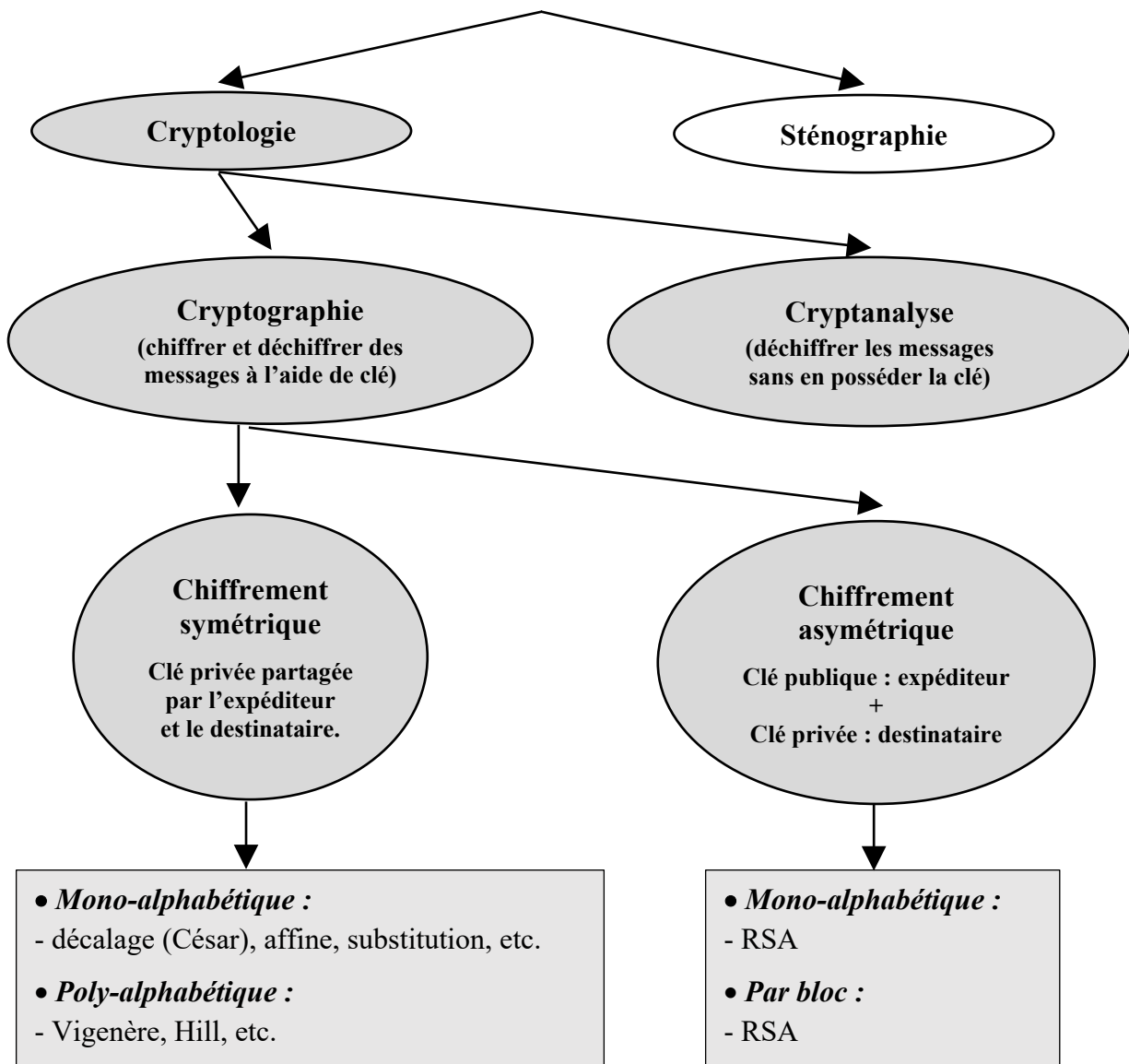
1.3.1 Introduction et terminologie

- La **cryptologie** est une science mathématique comportant deux branches : *la cryptographie* et *la cryptanalyse*.
- Le mot **cryptographie** vient du grec ancien *kryptô* – je cache – et *graphô* – j'écris. Lorsque l'on chiffre un message, on le modifie pour le rendre incompréhensible. Par exemple, un message top secret disant « Le code secret pour ouvrir un coffre-fort est X65byGdt8Dt. » ne peut pas être envoyé tel quel par la poste ou par courriel, c'est trop dangereux, car n'importe qui peut l'intercepter. On préférera l'envoyer sous une forme moins compréhensible :
« Ngu]eqfg]ugetgvu]rqwt]ncpegt]nc]dqodg]pwenéctg]uqp v]Z87d0IfvAFv ».
Quand les données d'un message sont chiffrées, on ne se cache pas de les avoir envoyées. N'importe qui peut intercepter le message. Mais bien malin celui qui intercepte un tel message, car il est incapable de le comprendre sans une « clé » pour le déchiffrer. C'est l'idée sous-jacente de la cryptographie : envoyer un message inintelligible que seul le destinataire saura déchiffrer mais que n'importe qui pourra intercepter. ***La cryptographie ne cache pas le message mais cache le sens du message.***
- La **cryptanalyse** est la technique qui consiste à déduire un *message en clair* d'un *message chiffré* sans posséder la « clé » de chiffrement. Le processus par lequel on tente de comprendre un message en particulier est appelé ***une attaque***. Une attaque est souvent caractérisée par les données qu'elle nécessite :

Cas 1 : le cryptanalyste possède des exemplaires chiffrés des messages, il peut faire des hypothèses sur les messages originaux qu'il ne possède pas. La cryptanalyse est plus ardue de par le manque d'informations à disposition.

Cas 2 : le cryptanalyste possède des messages ou des parties de messages en clair ainsi que les versions chiffrées.
- La **stéganographie**, du grec *stéganô* – je couvre – et *graphô* – j'écris –, fonctionne à l'inverse. ***Le sens du message reste clair mais le message lui-même est caché.*** En effet, le message sera présenté sous une forme telle que l'utilisateur ne se rendra même pas compte qu'il y a un message qui circule. L'encre invisible, aussi appelée encre sympathique, est un procédé de stéganographie. Quelqu'un écrit un message anodin avec une encre normale puis écrit le vrai message entre les lignes avec de l'encre invisible. Cette technique est toujours d'actualité et vous avez certainement passé du temps, enfant, à jouer avec de l'encre à base de citron qui n'apparaît que quand elle est chauffée. Il existe d'autres procédés de stéganographie plus ou moins subtils. Écrire un message micro-minuscule dans le point des i et des j d'un texte normal. La série Prison break présente un très bel exemple de stéganographie. Le héros cache dans son tatouage le plan d'évasion de la prison sous forme d'images incluse dans une image plus grande recouvrant tout son corps.
Il est à noter que si le message est caché, le support du message quant à lui peut être intercepté. Mais alors qu'un texte ou une image banale n'attirent pas l'attention et peuvent passer les lignes ennemies facilement, un texte chiffré donne envie de savoir ce qui se cache dedans. C'est là que réside une des forces de la stéganographie.
- Pour résumer, ***la cryptographie*** et ***la stéganographie*** sont deux procédés visant à protéger des messages afin d'assurer notamment la confidentialité des échanges ou des données. Cependant, les deux techniques diffèrent par leur approche et leur philosophie. Pour prendre une métaphore, la *stéganographie* consisterait à enterrer son argent dans son jardin là où *la cryptographie* consisterait à l'enfermer dans un coffre-fort. Retenez que pour rendre vos communications vraiment sécurisées, la panacée est encore de stéganographier des données chiffrées.

La cryptologie et la stéganographie sont deux procédés visant à **protéger des messages** afin d'assurer notamment la confidentialité des échanges ou des données.



Remarques

a) Un système cryptographique doit posséder **au minimum trois qualités** :

- 1) **la confidentialité** ; garantir que seul l'expéditeur et le destinataire pourront avoir une vision intelligible du message.
- 2) **l'authenticité** du message ; garantir l'identité de l'expéditeur.
- 3) **l'intégrité** du message ; garantir qu'il n'a pas été modifié par une tierce personne.

b) **Principe de Kerckhoffs** :

« La sécurité d'un système de chiffrement ne doit reposer **que sur la clé**. »

Autrement dit : « L'ennemi peut avoir connaissance du système de chiffrement (fonction de chiffrement et de déchiffrement) mais pas de la clé. »

c) **Grâce au chiffrement asymétrique**, la communication chiffrée est possible sur des réseaux informatiques non-sécurisés. Cela permet de développer le commerce en ligne (vente en ligne).

1.3.2 Fonctions de chiffrement, déchiffrement et clefs

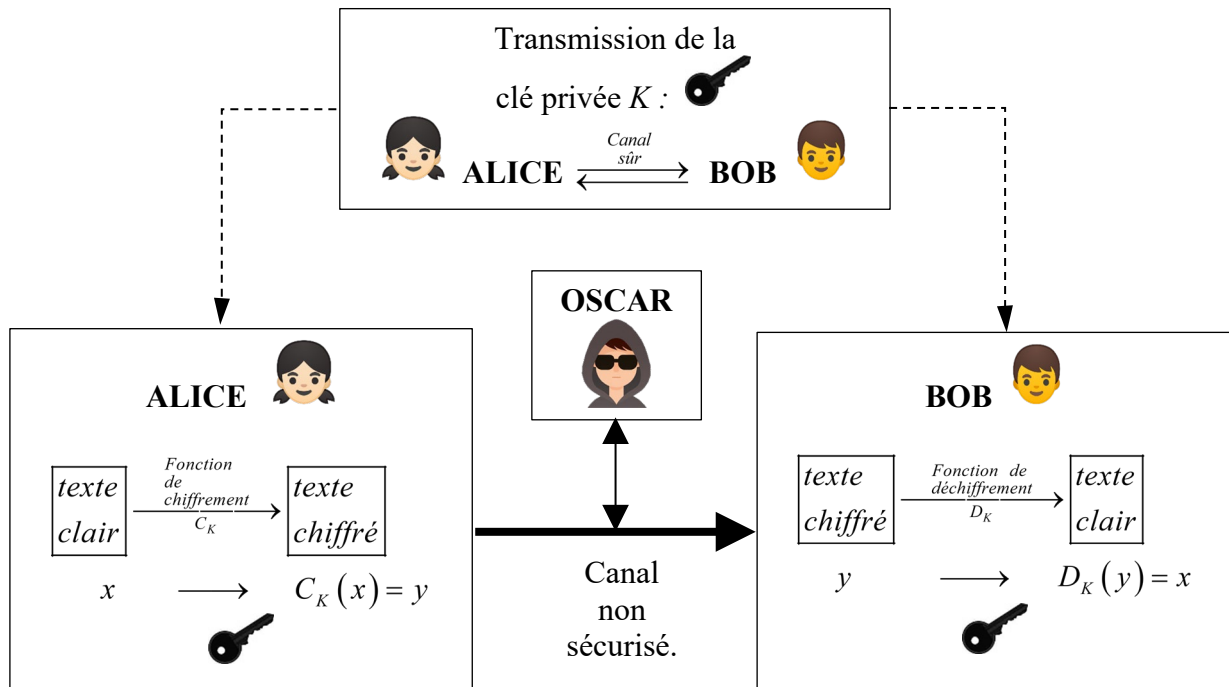
L'objectif fondamental de la **cryptographie** est de permettre à deux personnes, appelées traditionnellement **ALICE** et **BOB**, de communiquer au travers d'un canal non sécurisé de telle sorte qu'un opposant, **OSCAR** (le cryptanalyste), ne puisse pas comprendre ce qui est échangé. Le canal peut être par exemple une ligne de téléphone, une fibre optique ou tout autre réseau de communication.

L'information qu'ALICE souhaite transmettre à BOB, que l'on appelle **texte clair**, peut être un texte en français, une donnée numérique, ou n'importe quoi d'autre de structure arbitraire.

ALICE transforme le texte clair par un procédé de **chiffrement** en utilisant une **clef** prédéterminée et envoie le **texte chiffré** au travers du canal. OSCAR peut intercepter le texte chiffré en espionnant le canal non sécurisé mais ne peut retrouver le texte clair. BOB qui connaît la clef, peut **déchiffrer** le texte et le récupérer. Ces idées se formalisent ainsi.

Schéma de principe

ALICE (l'expéditrice) veut envoyer un texte à BOB (le destinataire) sur un canal non sécurisé. OSCAR est l'opposant (le cryptanalyste).



Remarques

a) C_k et D_k sont **bijectives** et vérifient : $C_k(D_k(y)) = y = I_d(y)$ et $D_k(C_k(x)) = x = I_d(x)$
Autrement dit, C_k et D_k sont **réciproques** l'une de l'autre.

b) Dans ce cours, un texte clair ou chiffré sera un texte en français écrit avec les 26 lettres de notre alphabet en majuscule et sans espace ni ponctuation.

Exemple : texte clair : *ZEBREENCAGE* texte chiffré : *CHEUHHQFDJH*

c) ALICE et BOB qui « jouent » ensemble contre OSCAR auront gagné si :

- 1) ils peuvent chiffrer et déchiffrer facilement des textes, et ce dans un temps humainement raisonnable.
- 2) OSCAR, le cryptanalyste ne peut déchiffrer le texte, même avec la puissance calculatoire des ordinateurs, en un temps raisonnable.

Afin de construire **une clef, une fonction de chiffrement et une fonction de déchiffrement** rigoureuse et mathématique on va d'abord associer à chacune des *26 lettres de notre alphabet* (de A à Z) un *nombre entier* par exemple entre 0 à 25.

En termes mathématiques, nous définissons **une bijection** notée ***b*** définie de la manière suivante :

lettre	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
b(lettre)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Considérons aussi la **bijection réciproque** notée b^{-1} :

nbre	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$b^{-1}(\text{nbre})$	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

Si on reprend notre texte clair : ZEBRENCAGE $\xleftrightarrow[b^{-1}]{b}$ 25_4_1_17_4_4_13_2_0_6_4

Si on reprend notre texte chiffré : CHEUHHQFDJH $\xleftrightarrow[b^{-1}]{b}$ 2_7_4_20_7_7_16_5_3_9_7

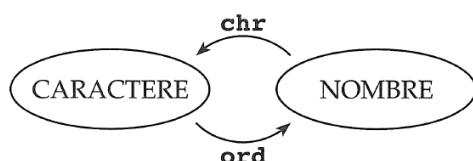
En Python on peut obtenir les bijections b et b^{-1} grâce au code ASCII :



In : ord('A')-65 Out : 0	In : ord('B')-65 Out : 1
In : chr(0+65) Out : 'A'	In : chr(1+65) Out : 'B'
In : ord('Y')-65 Out : 24	In : ord('Z')-65 25
In : chr(24+65) Out : 'Y'	In : chr(25+65) Out : 'Z'

Remarques

On utilise le code ASCII via les deux fonctions Python **chr()** et **ord()**



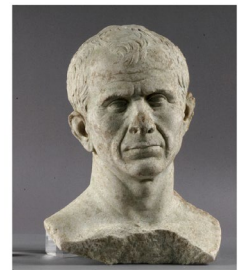
x1	0	1	2	3	4	5	6	7	8	9
x10			espace	!	"	#	\$	%	&	'
3										
4	()	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[\]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	□	€	

b : caractère $\xrightarrow{ASCII} \text{ord}(\text{caractère}) \rightarrow \text{ord}(\text{caractère}) - 65 = \text{nombre}$
 'C' $\rightarrow \text{ord}('C') = 67 \rightarrow 67 - 65 = 2$

b^{-1} : nombre $\xrightarrow{ASCII} \text{nombre} + 65 \rightarrow \text{chr}(\text{nombre} + 65) = \text{caractère}$
 2 $\rightarrow 2 + 65 \xrightarrow{ASCII} \text{chr}(2 + 65) = 'C'$

1.3.3 Chiffrement par décalage (de César)

Le premier système de chiffrement que nous présenterons est emprunté à Jules César (100-44 avant Jésus-Christ). L'idée du chiffrement de Jules de César est d'effectuer *un chiffrement par décalage mono-alphabétique*.



Buste de Jules César effectué de son vivant découvert à Arles en 2007

Explication / exemple :

- **ALICE** 🧑 prend chaque lettre de l'alphabet et lui associe une autre lettre de l'alphabet avec un même décalage précédemment choisis.

Alice (comme César) décide de décaler chaque lettre de l'alphabet de 3 positions.

Alphabet clair 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	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
<div><div>$C_3 \Downarrow$</div><div>$\Uparrow D_3$</div></div>																											
Alphabet chiffré y	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0	1	2	
	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	

La fonction de chiffrement C_3 utilisée par ALICE est :

$$\begin{aligned} \mathbb{Z}_{26} &\rightarrow \mathbb{Z}_{26} \\ x &\rightarrow C_3(x) = x + 3 \bmod 26 \end{aligned}$$

ALICE chiffre le texte clair « ZEBRE » :

texte clair	Z	E	B	R	E
nb clair : x	25	4	1	17	4
nb chiffré : $C_3(x) = x + 3 \bmod 26 = y$	2	7	4	20	7
texte chiffré	C	H	E	U	H

Remarques

a) La valeur du décalage définit une **clef k de chiffrement** ; ici $k = 3$.

Une clef $k = 3$, définit alors une fonction de chiffrement associée. Ici $C_3(x) = x + 3 \bmod 26$.

b) Il y a dans ce procédé de chiffrement la possibilité de créer 26 clefs $k \in \{0; 1; 2; \dots; 24; 25\}$ différentes et donc de créer 26 fonctions de chiffrement différentes : $C_k(x) = x + k \bmod 26$

c) Chaque fonction de chiffrement (ou de déchiffrement) définit une **permutation sur \mathbb{Z}_{26}** c'est-à-dire **une bijection sur l'ensemble des 26 lettres de notre alphabet**.

- **BOB** 🧑 a besoin d'une **fonction de déchiffrement D_k** bijective qui est la réciproque de C_k c'est-à-dire $D_k(C_k(x)) = x = I_d(x)$ sinon il ne pourra pas déchiffrer le message d'ALICE :

$$\begin{aligned} C_k(x) &= x + k \bmod 26 && \text{fonction de chiffrement} && (1.9) \\ \Leftrightarrow y &\equiv x + k \bmod 26 && \text{on soustrait } k \\ \Leftrightarrow y - k &\equiv x \bmod 26 && \text{symétrique} \\ \Leftrightarrow x &\equiv y - k \bmod 26 \\ \Leftrightarrow D_k(y) &= y - k \bmod 26 && \text{fonction de déchiffrement} \end{aligned}$$

La fonction de déchiffrement D_3 utilisée par BOB est :

$$\begin{array}{l} \mathbb{Z}_{26} \rightarrow \mathbb{Z}_{26} \\ y \rightarrow D_3(y) = y - 3 \bmod 26 \end{array}$$

BOB déchiffre le texte chiffré « CHEUH » :

texte chiffré	C	H	E	U	H
nb chiffré : y	2	7	4	20	7
nb clair : $D_3(y) = y - 3 \bmod 26 = x$	25	4	1	17	4
texte clair	Z	E	B	R	E

Remarque La clef k de déchiffrement est la même que celle de chiffrement : $k = 3$.

On peut proposer **le programme Python 3** ci-dessous qui permet de systématiser **la tâche de chiffrement par décalage** d'un texte clair de clef k . Le programme est constitué d'une *fonction Python* à deux paramètres `texte_clair` et k .



Les étapes principales de l'algorithme sont, dans l'ordre :

- 1) Prendre chaque lettre du texte clair et lui associer un nombre « clair » (bijection : b) via le code ASCII.
- 2) Appliquer la fonction de chiffrement :
Prendre chaque nombre « clair » et lui ajouter le nombre k (la clef).
Ensuite, calculer le reste de la division euclidienne par 26. Le résultat est un nombre « chiffré ».
- 3) Prendre chaque nombre « chiffré » et lui associer une lettre « chiffrée » (bijection : b^{-1}) via le code ASCII afin d'obtenir le texte chiffré.

```
def CESAR_CHIFFRE(texte_clair,k):
    texte_chiffre= [ ]
    for lettre in texte_clair:
        nb_clair=ord(lettre)-65
        nb_chiffre=(nb_clair+k)%26
        lettre_chiffre=chr(nb_chiffre+65)
        texte_chiffre.append(lettre_chiffre)
    texte_chiffre="".join(texte_chiffre)
    return texte_chiffre
```

Boucle
Bijection b
Fonction de chiffrement
Bijection b^{-1}
Conversion liste en chaîne de caractère

Exercice 1

a) Compléter le tableau ci-dessous afin de tester la *fonction py3* : **CESAR_CHIFFRE(texte_clair,k)**



	texte_clair	k	nb_clair	nb_chiffre	lettre_chiffre	texte_chiffre
Entrée	' ICI '	3				
Initialisation						
Boucle						
Conversion						
Sortie						

b) Ecrire dans un éditeur et comprendre la *fonction Python 3* :



CESAR_CHIFFRE(texte_clair,k) qui permet de **systématiser** la tâche de **chiffrement par décalage (César)** d'un texte clair de clef *k*.

c) Ecrire dans un éditeur une nouvelle *fonction Python 3* nommée :



CESAR_DECHIFFRE(texte_chiffre,k) qui permet de **systématiser** la tâche de **déchiffrement par décalage (César)** d'un texte chiffré de clef *k*.

Exemple : Sortie Python (console)

```
In : CESAR_CHIFFRE('ZEBRE',3)
Out: 'CHEUH'

In : CESAR_DECHIFFRE('CHEUH',3)
Out: 'ZEBRE'
```

Nom du fichier : ex_1_chiffrement_Cesar.py

Exercice 2

Dans cet exercice, on n'utilisera pas de programme informatique pour résoudre les problèmes.

- a) Chiffrer le texte clair : ' BATAILLE ' à l'aide du *chiffrement par décalage* de clef $K = 10$.

Remarque : c'est le travail d'ALICE.

texte clair	B	A	T	A	I	L	L	E
nb clair								
nb chiffré								
texte chiffré								

- b) Déchiffrer le texte chiffré : ' REYPKENA ' à l'aide du *chiffrement par décalage* de clef $K = 22$.

Remarque : c'est le travail de BOB.

texte chiffré	R	E	Y	P	K	E	N	A
nb chiffré								
nb clair								
texte clair								


Cryptanalyse du chiffrement par décalage (de César)

Définition

La **recherche exhaustive** ou **attaque par force brute** est une méthode algorithmique qui consiste principalement à essayer toutes les solutions possibles.

Exemple

Considérons le texte chiffré : 'INKANNWLJPN' à l'aide du chiffrement de César.

OSCAR  (le cryptanalyste) tient le raisonnement suivant :

Comme il n'y a que **26 clefs possibles** (26 décalages), on essaie le déchiffrement avec toutes les clefs jusqu'à trouver un texte clair compréhensible est aisé avec la puissance de calcul des ordinateurs actuels.

On essaie successivement toutes les fonctions de déchiffrements, $D_0, D_1, D_2, \text{etc.}$ On obtient :

D_0	<i>INKANNWLJPN</i>	D_{13}	<i>VAXNAAJYWCA</i>
D_1	<i>HMJZMMVKIOM</i>	D_{14}	<i>UZWMZZIXVBZ</i>
D_2	<i>GLIYLLUJHNL</i>	D_{15}	<i>TYVLYYHWUAY</i>
D_3	<i>FKHXKKTIGMK</i>	D_{16}	<i>SXUKXXGVTZX</i>
D_4	<i>EJGWJJSHFLJ</i>	D_{17}	<i>RWTJWWFUSYW</i>
D_5	<i>DIFVIIRGEKI</i>	D_{18}	<i>QVSIVVETR XV</i>
D_6	<i>CHEUHHQFDJH</i>	D_{19}	<i>PURHUUDSQWU</i>
D_7	<i>BGDTGGPECIG</i>	D_{20}	<i>OTQGTTCRPVT</i>
D_8	<i>AFCSFFODBHF</i>	D_{21}	<i>NSPFSSBQOUS</i>
D_9	<i>ZEBREENCAGE</i>	D_{22}	<i>MROERRAPNTR</i>
D_{10}	<i>YDAQDDMBZFD</i>	D_{23}	<i>LQNDQQZOMSQ</i>
D_{11}	<i>XCZPCCLAYEC</i>	D_{24}	<i>KPMCAPPYNLRP</i>
D_{12}	<i>WBYOBBKZXDB</i>	D_{25}	<i>JOLBOOXMKQO</i>




En observant les résultats, on a retrouvé **le texte clair** : « ZEBREENCAGE »
et **la clef** qui est ici $K = 9$.

Remarques


Le système de chiffrement par décalage n'assure pas :

- 1) **la confidentialité** ; garantir que seul l'expéditeur et le destinataire pourront avoir une vision intelligible du message.
En effet, on peut en un temps raisonnable, effectuer une attaque par force brute.
- 2) **l'authenticité** du message ; garantir l'identité de l'expéditeur.
- 3) **l'intégrité** du message ; garantir qu'il n'a pas été modifié par une tierce personne.

Exercice 3

Supposons qu'OSCAR  intercepte un texte chiffré envoyé par ALICE  à BOB .

On fait l'hypothèse, qu'OSCAR connaît le système de chiffrement utilisé qui est dans cet exercice le *chiffrement par décalage* (voir principe de Kerckhoffs).

a) En utilisant la fonction **CESAR_DECHIFFRE(texte_chiffre,k)**, écrire *un programme en Python 3* permettant « d'attaquer » par la méthode dite de **recherche exhaustive**, le *chiffrement par décalage* (voir exemple précédant). 

b) Aider OSCAR à faire **une cryptanalyse** sur les textes chiffrés suivants.

Autrement dit, déterminer **les textes clairs** ainsi que **la valeur des clefs**.



1) texte chiffré 01 : ' INKANNWLJPN '


2) texte chiffré 02 : ' LKDKSVVO '

3) texte chiffré 03 : ' REYPKENA '

Nom du fichier : ex_3_cryptanalyse_Cesar_force brute.py

Exercice 4

Supposons qu'OSCAR  vole dans le bureau d'ALICE  une feuille de papier où est écrit **un texte clair et son texte chiffré associé**. Il sait de plus que le système cryptographique utilisé est le *chiffrement par décalage* (voir principe de Kerckhoffs).

a) Ecrire *un programme en Python 3* permettant d'**obtenir la clef k** connaissant un texte clair et son texte chiffré associé. 

b) Aider OSCAR à obtenir les clefs dans les cas suivants :

1) le texte clair : ' PHYSIQUE ' correspondant au texte chiffré : ' XPGAQYCM '.

2) le texte clair : ' SOLEIL ' correspondant au texte chiffré : ' FBYRVY '.

3) le texte clair : ' ORDINATEURQUANTIQUE ' correspondant au texte chiffré : 'DGSXCPITJGFJPCIXFJT'.

Nom du fichier : ex_4_cryptanalyse_Cesar_clef.py

1.3.4 Chiffrement affine

ALICE  et BOB  s'envoient régulièrement des e-mails. Afin de sécuriser leurs échanges, ils décident d'utiliser le **système de chiffrement affine** dont voici la description.

- Rappel : x = nombre clair (non-chiffré) qui correspond à une lettre clair.
 y = nombre chiffré qui correspond à une lettre chiffrée.
- La **fonction de chiffrement** C qui doit être bijective de \mathbb{Z}_{26} vers \mathbb{Z}_{26} est définie par :

$$\begin{array}{l} \mathbb{Z}_{26} \rightarrow \mathbb{Z}_{26} \\ x \rightarrow C_{(a;b)}(x) = ax + b \text{ mod } 26 \end{array}$$

- La **clef** est un couple de nombre : $K = (a;b) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26}$.
- Déterminons la **fonction de déchiffrement** D qui doit être bijective de \mathbb{Z}_{26} vers \mathbb{Z}_{26} : **(1.10)**

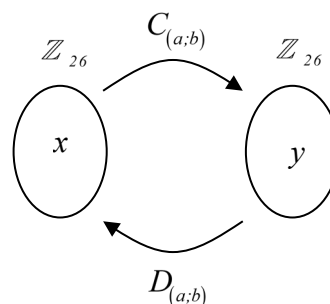
$$\begin{array}{ll} C_{(a;b)}(x) = ax + b \text{ mod } 26 & \text{fonction de chiffrement} \\ \Leftrightarrow y \equiv ax + b \text{ mod } 26 & \text{on soustrait } b \\ \Leftrightarrow y - b \equiv ax \text{ mod } 26 & \text{on multiplie par } a^{-1} \text{ qui est l'inverse de } a \text{ modulo } 26 \\ \Leftrightarrow a^{-1} \cdot (y - b) \equiv a^{-1} \cdot (ax) \text{ mod } 26 & \text{associativité de la multiplication} \\ \Leftrightarrow a^{-1} \cdot (y - b) \equiv (a^{-1} \cdot a) \cdot x \text{ mod } 26 & a \cdot a^{-1} \equiv a^{-1} \cdot a \equiv 1 \text{ mod } 26 \\ \Leftrightarrow a^{-1} \cdot (y - b) \equiv 1 \cdot x \text{ mod } 26 & 1 \cdot x \equiv x \cdot 1 \equiv x \text{ mod } 26 \text{ et symétrique} \\ \Leftrightarrow x \equiv a^{-1} \cdot (y - b) \text{ mod } 26 & \\ \Leftrightarrow D_{(a;b)}(y) = a^{-1} \cdot (y - b) \text{ mod } 26 & \text{fonction de déchiffrement} \end{array}$$

- Si a admet un inverse a^{-1} modulo 26 alors on peut définir une fonction de déchiffrement :

$$\begin{array}{l} \mathbb{Z}_{26} \rightarrow \mathbb{Z}_{26} \\ y \rightarrow D_{(a;b)}(y) = a^{-1}(y - b) \text{ mod } 26 \end{array}$$

- $D_{(a;b)}$ est la fonction réciproque de $C_{(a;b)}$ et les deux fonctions sont bijectives de \mathbb{Z}_{26} vers \mathbb{Z}_{26} .

$$\begin{array}{ll} \text{Autrement dit : } D_{(a;b)}(C_{(a;b)}(x)) = x & \forall x \in \mathbb{Z}_{26} \\ C_{(a;b)}(D_{(a;b)}(y)) = y & \forall y \in \mathbb{Z}_{26} \end{array}$$



Exemple

Considérons la clef $K = (5; 3)$.

$a = 5$ admet un inverse modulo 26 qui est $a^{-1} = 21$. Vérification : $5 \cdot 21 \equiv 1 \text{ mod } 26$

$b = 3$ peut-être quelconque.

La clef est valide car a^{-1} existe et définit deux fonctions bijectives et réciproques l'une de l'autre :

- *Fonction de chiffrement* : $C_{(5;3)}(x) = 5x + 3 \text{ mod } 26$
- *Fonction de déchiffrement* : $D_{(5;3)}(y) = 21(y - 3) \text{ mod } 26$ (l'inverse de 5 est 21 modulo 26)

ALICE chiffre le texte clair « ECLAIR » :

texte clair	E	C	L	A	I	R
nb clair : x	4	2	11	0	8	17
nb chiffré : $C_{(5;3)}(x) = 5x + 3 \text{ mod } 26$	23	13	6	3	17	10
texte chiffré	X	N	G	D	R	K

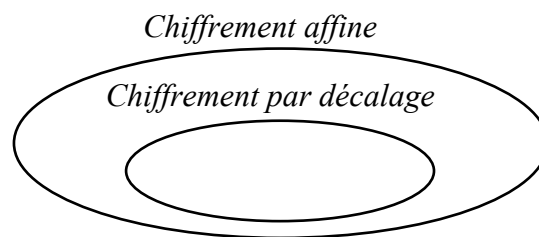
BOB déchiffre le texte chiffré « XNGDRK » :

texte chiffré	X	N	G	D	R	K
nb chiffré : y	23	13	6	3	17	10
nb clair : $D_{(5;3)}(y) = 21(y - 3) \text{ mod } 26$	4	2	11	0	8	17
texte clair	E	C	L	A	I	R

Remarque

Si on considère la clef $K = (1; b)$ on obtient un chiffrement par décalage (de César) :

- $C_{(1;b)}(x) = 1 \cdot x + b \text{ mod } 26 = x + b \text{ mod } 26$
- $D_{(1;b)}(y) = 1 \cdot (y - b) \text{ mod } 26 = y - b \text{ mod } 26$ (l'inverse de 1 est 1 modulo 26)



On peut proposer *le programme Python 3* ci-dessous qui permet de systématiser **la tâche de chiffrement affine** d'un texte clair de clef $K = (a; b)$. Le programme est constitué d'une *fonction Python* à trois paramètres *texte_clair*, *a* et *b*.



```
def AFFINE_CHIFFRE(texte_clair,a,b):
    texte_chiffre= [ ]
    for lettre in texte_clair:
        nb_clair=ord(lettre)-65
        nb_chiffre=(a*nb_clair+b)%26
        lettre_chiffre=chr(nb_chiffre+65)
        texte_chiffre.append(lettre_chiffre)
    texte_chiffre="".join(texte_chiffre)
    return texte_chiffre
```

Boucle
Bijection b
Fonction de chiffrement
Bijection b^-1
Conversion liste en chaîne de caractère

Exercice 5

a) Compléter le tableau suivant afin de tester la *fonction py3* : **AFFINE_CHIFFRE(texte_clair,a,b)**



	texte_clair	$(a;b)$	nb_clair	nb_chiffre	lettre_chiffre	texte_chiffre
Entrée	"ICI"	$(5;3)$				
Initialisation						
Boucle						
Conversion						
Sortie						

b) Ecrire dans un éditeur et comprendre *la fonction Python 3* :

AFFINE_CHIFFRE(texte_clair,a,b) qui permet de systématiser **la tâche de chiffrement affine** d'un texte clair de clef $K = (a;b)$.



c) Ecrire dans un éditeur une nouvelle *fonction Python 3* nommée :

AFFINE_CHIFFRE(texte_clair,a,b) qui permet de systématiser **la tâche de déchiffrement affine** d'un texte chiffré de clef $K = (a;b)$.



Exemple : Sortie Python (console)

```
In : AFFINE_CHIFFRE("BATAILLE",9,2)
Out : 'LCRCWXXM'

In : AFFINE_DECHIFFRE('LCRCWXXM',9,2)
Out : 'BATAILLE'
```

Nom du fichier : ex_5_chiffrement_affine.py

Exercice 6

Dans cet exercice, on n'utilisera pas de programme informatique pour résoudre les problèmes.

Considérons un chiffrement affine de clef $K = (4; 2)$

a) Compléter et observer le tableau de chiffrement ci-dessous :

Alphabet clair 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
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$\Downarrow C_{(4;2)}$																										
Alphabet chiffré y														2	6	10	14	18	22	0	4	8	12	16	20	24
														C	G	K	O	S	W	A	E	I	M	Q	U	Y

b) Chiffrer le texte : 'TELEGRAMME' avec le clef $K = (4; 2)$.

c) Expliquer pourquoi la clef $K = (4; 2)$ ne définit pas une fonction de chiffrement valide.

d) Expliquer pourquoi on ne peut pas obtenir la fonction de déchiffrement $D_{(4;2)}$ associé à la clef $K = (4; 2)$.

e) Expliquer pourquoi il est difficile de déchiffrer le texte : 'YCAESYCAIOESW' avec le clef $K = (4; 2)$.

Exercice 7

Dans cet exercice, on n'utilisera pas de programme informatique pour résoudre les problèmes.

Considérons un chiffrement affine de clef $K = (3; 21)$.


a) La clef $K = (3; 21)$ définit-elle une fonction de chiffrement valide ? Justifier.

b) En utilisant la clef $K = (3; 21)$, déchiffrer le message 'XLCHTC' en effectuant toutes les étapes de calculs.

c) En utilisant la clef $K = (3; 21)$, chiffrer le message 'PAIX' en effectuant toutes les étapes de calculs.

Cryptanalyse du chiffrement affine

Question

OSCAR  (le cryptanalyste) peut-il en un temps raisonnable, effectuer une *attaque par force brute* sur un chiffrement affine ?

Pour **répondre** à cette question, il faut déterminer combien il y a de clefs $K = (a; b) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26}$ distinctes et valides pour cette méthode de chiffrement :

- Il y a 12 entiers a qui admettent un inverse a^{-1} modulo 26 .
En voici la liste : $a \in \{1; 3; 5; 7; 9; 11; 15; 17; 19; 21; 23; 25\}$
- Il y a 26 entiers b modulo 26 . L'entier b n'a pas d'influence sur l'existence des fonctions de chiffrement et de déchiffrement bijectives.
En voici la liste : $b \in \{1; 2; 3; 4; 5; 6; 7; 8; 9; \dots; 19; 20; 21; 22; 23; 24; 25\}$
- Il y donc $12 \cdot 26 = 312$ clefs valides et différentes permettant de définir dans chaque cas, une fonction de chiffrement bijective et une fonction de déchiffrement bijective.




Remarque

En fait on peut considérer **311** clefs « utiles » pour ce système de chiffrement car la clef $K = (1; 0)$ qui est l'identité n'a pas d'intérêt cryptographique.

Conclusion

Le nombre de clefs *n'est pas gigantesque*. La puissance de calcul des ordinateurs actuels et la perspicacité d'OSCAR risque bien de venir à bout de ce système de chiffrement grâce à une attaque par force brute (voir exercice ci-dessous).

Exercice 8

Supposons qu'OSCAR  intercepte un texte chiffré envoyé par ALICE  à BOB .

On fait l'hypothèse, qu'OSCAR connaît le système de chiffrement utilisé qui est dans cet exercice le *chiffrement affine* (voir principe de Kerckhoffs).

a) En utilisant la fonction **AFFINE_DECHIFFRE(texte_chiffre,a,b)**, écrire *un programme en Python 3* permettant « d'attaquer » par la méthode dite de **recherche exhaustive**, le *chiffrement affine*.



b) Aider OSCAR à faire **une cryptanalyse** sur le texte chiffré : 'OIDABZVQVTEMDTGDZNUV'.
Autrement dit, déterminer le texte clair ainsi que la clef $K = (a; b)$.

Nom du fichier : **ex_8_cryptanalyse_affine_force brute.py**

Une **faiblesse du chiffrement affine** est qu'il est **mono-alphabétique**. Autrement dit, une même lettre est toujours chiffrée de la même façon. Dans les textes « longs », les lettres n'apparaissent pas avec la même fréquence. Ces fréquences varient suivant la langue utilisée.

C'est ainsi qu'en français, la lettre la plus fréquente est le **E** suivi du **S** puis du **A**.

- Supposons que nous ayons à déchiffrer le texte ' YMQMGGKAMMGNNELGMYZMN ' qui a été chiffré au moyen d'une transformation affine.
- Avec un peu de chance, cet ordre "fréquentiel" va être suivi, à quelque chose près, par les lettres du texte à déchiffrer. Ici, dans le court message que nous avons à déchiffrer, la lettre la plus fréquente est le M qui apparaît 6 fois, suivi du G qui apparaît 4 fois. Faisons alors l'hypothèse que le M correspond au E et le G au S.

- Passons aux équivalents numériques :

$$\begin{array}{l} E \xrightarrow{b} 4 \xrightarrow{c} a \cdot 4 + b \bmod 26 \equiv 12 \xrightarrow{b^{-1}} M \\ S \xrightarrow{b} 18 \xrightarrow{c} a \cdot 18 + b \bmod 26 \equiv 6 \xrightarrow{b^{-1}} G \end{array} \Rightarrow \begin{cases} 12 \equiv 4a + b \bmod 26 \\ 6 \equiv 18a + b \bmod 26 \end{cases}$$

- Nous sommes donc ramenés à résoudre un système de 2 équations à 2 inconnues modulo 26 ; a et b sont les inconnues et correspondent à la clef du chiffrement $K = (a; b)$
- Résolvons le système de 2 équations à 2 inconnues modulo 26 et obtenir la clef $K = (a; b)$:

$$\begin{cases} 12 \equiv 4a + b \bmod 26 \\ 6 \equiv 18a + b \bmod 26 \end{cases} \quad | \quad E_2 - E_1$$

$$\Leftrightarrow -6 \equiv 14a \bmod 26$$

$$\Leftrightarrow 14a \equiv -6 \bmod 26$$

$$\Leftrightarrow 14a \equiv 20 \bmod 26 \quad | \quad \text{car } 20 \equiv -6 \bmod 26$$

$$\Leftrightarrow 14a = 26k + 20 \quad k \in \mathbb{Z} \quad | \quad \div \text{PGCD}(14; 26) = 2$$

$$\Leftrightarrow 7a = 13k + 10 \quad k \in \mathbb{Z}$$

$$\Leftrightarrow 7a \equiv 10 \bmod 13 \quad | \quad \cdot 2 \text{ qui est l'inverse de 7 modulo 13}$$

$$\Leftrightarrow 14a \equiv 20 \bmod 13$$

$$\Leftrightarrow a \equiv 7 \bmod 13 \quad | \quad \text{car } 14 \equiv 1 \bmod 13 \text{ et } 20 \equiv 7 \bmod 13$$

$$\Leftrightarrow a \equiv 13k + 7 \quad k \in \mathbb{Z}$$

$$\text{Donc } a \in \{7; 20; 33; \dots\}$$

On choisit $a = 7$ car il possède un inverse $a^{-1} = 15$ modulo 26

On substitue $a = 7$ dans l'équation E_1 : $12 \equiv 4 \cdot 7 + b \bmod 26 \Leftrightarrow b \equiv 10 \bmod 26$ donc $b = 10$

- Vérification :
$$\begin{array}{l} E \xrightarrow{b} 4 \xrightarrow{c} 7 \cdot 4 + 10 \bmod 26 \equiv 12 \xrightarrow{b^{-1}} M \\ S \xrightarrow{b} 18 \xrightarrow{c} 7 \cdot 18 + 10 \bmod 26 \equiv 6 \xrightarrow{b^{-1}} G \end{array}$$


- Finalement : la clef de chiffrement est $K = (7; 10)$

- Déterminons la fonction de chiffrement $C_{(a;b)}$ et de déchiffrement $D_{(a;b)}$:

Fonction de chiffrement : $C_{(7;10)}(x) = 7x + 10 \bmod 26$

Fonction de déchiffrement : $D_{(7;10)}(y) = 15(y - 10) \bmod 26$ (l'inverse de 7 est 15 modulo 26)

- Déchiffrons le message : ' YMQMGGKAMMGNNELGMYZMN ' en utilisant *Python 3* :

Sortie Python (console) 

```
AFFINE_DECHIFFRE("YMQMGGKAMMGNNELGMYZMN",7,10)
>>> 'CEMESSAGEESTTOPSECRET'
```

- Remarque

Nous avons effectué une **attaque statistique** sur un chiffrement mono-alphabétique.

Exercice 9

Supposons qu'OSCAR  intercepte un texte chiffré envoyé par ALICE  à BOB .

On fait l'hypothèse, qu'OSCAR connaît le système de chiffrement utilisé qui est dans cet exercice le **chiffrement affine** (voir principe de Kerckhoffs).

- a) En utilisant la méthode dite **d'attaque statistique**, aider OSCAR à faire **une cryptanalyse** sur le texte chiffré : 'CJLZBBNAKJBBAUBFCFUNFWJJVUYWJU' .

Autrement dit, déterminer le texte clair ainsi que la clef $K = (a;b)$.

Indications : On suppose que le texte clair commence par l'article 'LE' .

Résoudre un système de 2 équations à 2 inconnues modulo 26.

- b) Déterminer la fonction de chiffrement $C_{(a;b)}$ et de déchiffrement $D_{(a;b)}$.

Remarques

Le système de chiffrement affine n'assure pas :

- 1) **la confidentialité** ; garantir que seul l'expéditeur et le destinataire pourront avoir une vision intelligible du message. En effet, on peut en un temps raisonnable, effectuer une attaque par force brute ou une attaque statistique.
- 2) **l'authenticité** du message ; garantir l'identité de l'expéditeur.
- 3) **l'intégrité** du message ; garantir qu'il n'a pas été modifié par une tierce personne.

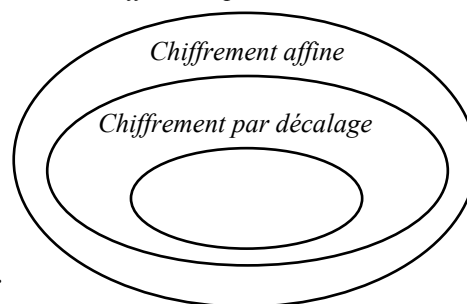
1.3.5 Chiffrement par substitution

Chiffrement par substitution

Un autre système de chiffrement bien connu est *le chiffrement par substitution mono-alphabétique*.


L'idée du chiffrement par substitution est de considérer toutes *les permutations (bijections) sur l'ensemble des caractères alphabétiques*.

C'est une généralisation du chiffrement par décalage et affine.



Explication / exemple

(1.11)

- **ALICE**  utilise une **permutation** π de l'ensemble des lettres de l'alphabet vers lui-même.

ALICE décide de définir une permutation de la manière suivante : prendre l'ordre des lettres sur le clavier d'un ordinateur (QWERTZ).

Alphabet clair 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
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
<div><div>$\pi \downarrow$</div><div>$\uparrow \pi^{-1}$</div></div>																										
Alphabet chiffré y	16	22	4	17	19	25	20	8	14	15	0	18	3	5	6	7	9	10	11	24	23	2	21	1	13	12
	Q	W	E	R	T	Z	U	I	O	P	A	S	D	F	G	H	J	K	L	Y	X	C	V	B	N	M

Donc $\pi(A) = Q$, $\pi(B) = W$, $\pi(C) = E$, etc.

La fonction de chiffrement C_π utilisée par ALICE est :

$$\begin{array}{l} \mathbb{Z}_{26} \rightarrow \mathbb{Z}_{26} \\ x \rightarrow C_\pi(x) = \pi(x) \end{array}$$

ALICE chiffre le texte clair « ZEBRE » :

texte clair	Z	E	B	R	E
nb clair : x	25	4	1	17	4
nb chiffré : $C_\pi(x) = \pi(x)$	12	19	22	10	19
texte chiffré	M	T	W	K	T

Remarques

a) Une permutation π définit une **clef de chiffrement** ; ici l'ordre des lettres sur un clavier.

b) Il y a dans ce procédé de chiffrement la possibilité de créer :

$26 \cdot 25 \cdot 24 \cdot 23 \cdot \dots \cdot 3 \cdot 2 \cdot 1 = 26!$ clefs différentes, ce qui est supérieur à $4,0 \cdot 10^{26}$.

Le nombre de clefs est donc gigantesque.

- **BOB**  a besoin d'une **fonction de déchiffrement** $D_{\pi^{-1}}$ qui est la réciproque de C_π c'est-à-dire

$D_{\pi^{-1}}(C_\pi(x)) = x = I_d(x)$ sinon il ne pourra pas déchiffrer le message d'ALICE.

Dans notre cas c'est très facile, si on a pris la permutation π , il suffit de considérer la permutation réciproque π^{-1} et donc $\pi^{-1}(\pi(x)) = x$ ainsi que $\pi(\pi^{-1}(y)) = y$.

Donc $\pi^{-1}(Q) = A$, $\pi^{-1}(W) = B$, $\pi^{-1}(E) = C$, etc.

La fonction de déchiffrement $D_{\pi^{-1}}$ utilisée par BOB est :

$$\begin{array}{l} \mathbb{Z}_{26} \rightarrow \mathbb{Z}_{26} \\ y \rightarrow D_{\pi^{-1}}(y) = \pi^{-1}(y) \end{array}$$

BOB déchiffre le texte chiffré « CHEUH » :

texte chiffré	M	T	W	K	T
nb chiffré : y	12	19	22	10	19
nb clair : $D_{\pi^{-1}}(y) = \pi^{-1}(y)$	25	4	1	17	4
texte clair	Z	E	B	R	E

Remarque

Il suffit de lire le tableau de la permutation π (chiffrement) du bas vers le haut pour obtenir la permutation π^{-1} (déchiffrement).

On peut proposer **le programme Python 3** ci-dessous qui permet de systématiser **la tâche de chiffrement par substitution** d'un texte clair de clef π . Le programme est constitué d'une *fonction Python* à deux paramètres *texte_clair* et *clef*.



Les étapes principales de l'algorithme sont, dans l'ordre :

- 1) Définir la clef de chiffrement en construisant une permutation π sur l'ensemble des lettres de l'alphabet en définissant deux chaînes de caractères CH1 et CH2.
CH1 et CH2 possèdent 26 caractères chacune (les 26 lettres de l'alphabet).
- 2) Parcourir le « texte clair » (chaîne de caractère) et obtenir l'image de toutes les lettres par la permutation π .
- 3) L'image de toutes les lettres ainsi obtenues est une chaîne de caractère qui est le « texte chiffré » possédant le même nombre de caractère que le « texte clair ».

```
def SUB_CHIFFRE(texte_clair,clef):
```

```
    CH1='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    CH2=clef
```

```
    texte_chiffre= ' ' # variable : chaîne de caractère
```

```
    for lettre in texte_clair :
```

```
        for j in range(26):
```

```
            if lettre == CH1[j]:
```

```
                texte_chiffre= texte_chiffre+CH2[j]
```

```
    return texte_chiffre
```

Exercice 10

a) Compléter le tableau suivant afin de tester la *fonction python 3* : **SUB_CHIFFRE(texte_clair,clef)**

La clef (permutation π) utilisée est : CH2='QWERTZUIOPASDFGHJKLYXCVBNM'



	texte_clair	lettre	CH1[j]	CH2[j]	texte_chiffre
Entrée	"ZEBRE"				
Initialisation					
Boucle					
Sortie					

b) Ecrire dans un éditeur et comprendre la *fonction Python 3* :

SUB_CHIFFRE(texte_clair,clef) qui permet de **systématiser**
la tâche de **chiffrement par substitution** d'un texte clair de clef CH2.



c) Ecrire dans un éditeur une nouvelle *fonction Python 3* nommée :

SUB_DECHIFFRE(texte_chiffre,clef) qui permet de **systématiser**
la tâche de **déchiffrement par substitution** d'un texte chiffré de clef CH2.



Exemple : Sortie Python (console)

```
In : SUB_CHIFFRE('CECIESTUNESSAIDEPHRASEVRAIE','QWERTZUIOPASDFGHJKLYXCVBNM')
Out : 'ETEOTLYXFTLLQORTHIKQLTCKQOT'

In : SUB_DECHIFFRE('ETEOTLYXFTLLQORTHIKQLTCKQOT','QWERTZUIOPASDFGHJKLYXCVBNM')
Out : 'CECIESTUNESSAIDEPHRASEVRAIE'
```

Nom du fichier : ex_10_chiffrement_subtitution.py

Exercice 11

Dans cet exercice, on n'utilisera pas de programme informatique pour résoudre les problèmes.

- a) 1) Chiffrer le texte clair : ' BATAILLE ' à l'aide du *chiffrement par substitution* dont le clef est la permutation π_1 . Remarque : c'est le travail d'ALICE.

Alphabet en clair 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
$\Downarrow \pi_1$																										
Alphabet chiffré y	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

texte clair	B	A	T	A	I	L	L	E
texte chiffré								

- 2) Quel est la particularité de ce chiffrement ?

- b) Considérons la clef d'un chiffrement par substitution définie par la permutation π_2 suivante :

Alphabet en clair 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
$\Downarrow \pi_2$																										
Alphabet chiffré y	M	N	B	V	C	X	Y	L	K	J	H	G	F	D	S	A	P	O	I	U	Z	T	R	E	W	Q

Déchiffrer le texte chiffré : ' NSDJSZO '. Remarque : c'est le travail de BOB.

texte chiffré	N	S	D	J	S	Z	O
texte clair							

Cryptanalyse du chiffrement par substitution

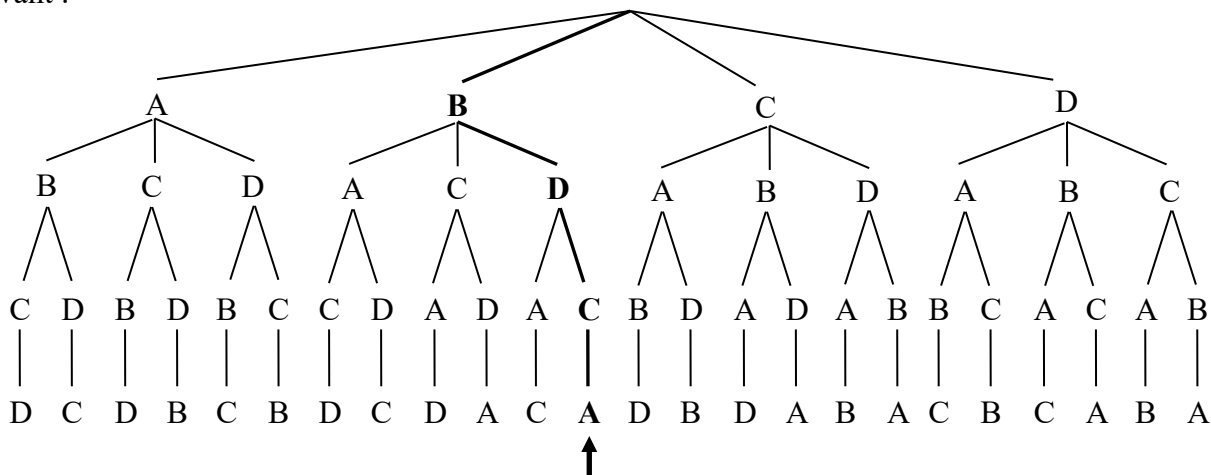
Nombres de clefs

Si on utilise le *chiffrement par substitution* le nombre de clefs est gigantesque.

En faisant un arbre de classement on se rend compte qu'il y a : $26 \cdot 25 \cdot 24 \cdot 23 \cdot \dots \cdot 3 \cdot 2 \cdot 1 = 26!$ clefs différentes (chemins de l'arbre), ce qui est supérieur à $4,0 \cdot 10^{26}$.

Explications

Considérons un « alphabet » composé de 4 lettres distinctes A, B, C, D et l'arbre de classement suivant :



$BDCA$ est une permutation π des 4 lettres $ABCD$.

Autrement dit : $A \rightarrow B$; $B \rightarrow D$; $C \rightarrow C$; $D \rightarrow A$

Chaque « chemin » de l'arbre correspond à une permutation π des 4 lettres : A, B, C, D .

Combien y a-t-il de permutations de ces 4 lettres distinctes ?

Il y a 4 possibilités pour écrire la première lettre de la permutation, 3 possibilités pour écrire la deuxième lettre, 2 possibilités pour écrire la troisième lettre et finalement 1 possibilité pour écrire la quatrième lettre.

On a donc en tout : $4 \cdot 3 \cdot 2 \cdot 1 = 24$ ^{notation factorielle} $= 4!$ permutations différentes des 4 lettres distinctes.

Cas général :

Si on a un alphabet de n lettres distinctes alors on a $n \cdot (n-1) \cdot \dots \cdot 2 \cdot 1 = n!$ permutations différentes des n lettres distinctes.

Remarques

- a) Une **recherche exhaustive**, c'est-à-dire, tester toutes les clefs est donc inutile en pratique.
- b) Le *chiffrement par substitution* n'est pas sûr malgré le nombre immense de clefs, car on peut utiliser une **attaque statistique** en étudiant la fréquence d'apparition des lettres dans un texte suffisamment long.

Explication

La principale faiblesse de ce chiffrement est qu'il est **mono-alphabétique**. Autrement dit, une même lettre est toujours chiffrée de la même façon. Dans les textes « longs », les lettres n'apparaissent pas avec la même fréquence. Ces fréquences varient suivant la langue utilisée.

En français, les lettres les plus rencontrées sont dans l'ordre :

E_S_A_I_N_T_R_L_U_O_D_C_P_M_V_G_F_B_Q_H_X_J_Y_Z_K_W

Voici le tableau des fréquences d'apparition des lettres en français :

Lettre	Fréquence	Lettre	Fréquence
A	8.08 %	N	7.13 %
B	1.06 %	O	5.26 %
C	3.03 %	P	3.01 %
D	4.18 %	Q	0.99 %
E	17.26 %	R	6.55 %
F	1.12 %	S	8.40 %
G	1.27 %	T	7.07 %
H	0.92 %	U	5.74 %
I	7.34 %	V	1.32 %
J	0.31 %	W	0.04 %
K	0.05 %	X	0.45 %
L	6.01 %	Y	0.30 %
M	2.96 %	Z	0.12 %

Remarques : a) Certaines lettres ont des fréquences très proches.

b) Les résultats du tableau dépendent de l'échantillon utilisé et peut donc varier.

Méthodologie pour effectuer une attaque statistique

- Dans le texte chiffré, on cherche d'abord *la lettre qui apparaît le plus fréquemment*, et si le texte est assez long cela devrait être le chiffrement du *E*.
- La *lettre qui apparaît ensuite dans l'étude des fréquences* devrait être le chiffrement du *S*, puis le chiffrement du *A*, etc.
- On obtient des morceaux de texte clair sous la forme d'une texte à trous et il faut ensuite deviner les lettres manquantes.

Exemple



OSCAR (le cryptanalyste) essaie de déchiffrer le texte chiffré :

' ETEOTLYXF TLLQORTH I KQLTCKQOT ' qui comporte 27 caractères.

Il sait que le chiffrement par substitution a été utilisé.

- On compte les apparitions des lettres les plus fréquentes du texte chiffré ci-dessus :
T : 6 L : 4 Q : 3 O : 3
- On suppose donc que le T chiffre la lettre E. On remplace toutes les occurrences de la lettre T par la lettre 'e' : ' EeEOeLYXF eLLQOR eHIKQL eCKQOe '
- Ensuite on suppose que le L chiffre la lettre S. On remplace toutes les occurrences de la lettre L par la lettre 's' : ' EeEOesYXFessQOR eHIKQseCKQOe '
- D'après les statistiques Q et O devraient chiffrer en A et I (ou I et A).
La séquence ' essQO ', se complète donc en ' **essai** ' ou ' **essia** '.

De plus, **en français**, voici les **20 bi-grammes** les plus fréquents et dans l'ordre :

ES	DE	LE	EN	RE	NT	ON	ER	TE	EL
AN	SE	ET	LA	AI	IT	ME	OU	EM	IE

La première solution semble correcte. Ainsi Q chiffre le A et O chiffre le I.

- Après substitution, la phrase est maintenant : ' EeEiesYXFessaiReHIKaseCKaie '
- En réfléchissant un petit peu, on déchiffre le message : ' **ceci est un essai de phrase vraie** '
- Avec les bons espaces : ' **ceci est un essai de phrase vraie** '

Remarques

- a) On a obtenu que partiellement la clef de chiffrement (permutation π) car le texte chiffré est court et ne contient pas toutes les lettres de l'alphabet permettant d'établir toutes les correspondances. Si l'image d'une lettre par π n'est pas connue, on utilise le caractère ' * '.

Alphabet chiffre y	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	
$\Downarrow \pi$																											
Alphabet clair x	Q	*	E	R	T	*	*	I	O	*	*	*	*	F	*	H	*	K	L	Y	X	C	*	*	*	*	

Pour obtenir l'intégralité de la clef de chiffrement (permutation π) il faudrait avoir un texte chiffré d'une longueur plus importante contenant toutes les lettres de l'alphabet.

- b) Plus la longueur du texte chiffré est grand meilleur sera l'analyse des fréquences.
- c) Dans cet exemple ALICE a utilisé la clef : CH1 = ' ABCDEFGHIJKLMNOPQRSTUVWXYZ '
CH2 = ' QWERTZUIOPASDFGHJKLYXCVBNM '

Historique

Par sa simplicité (permutation) et par sa force (nombre de clefs) **le chiffrement par substitution**, a dominé la technique des écritures secrètes pendant tout le premier millénaire. Il a résisté aux cryptanalystes jusqu'à ce que le savant arabe **Al-Kindi** mette au point, au IXème siècle, une technique appelée analyse des fréquences. Al-Kindi (801-873) rédige sa méthode dans son plus important traité intitulé *Manuscrit sur le déchiffrement des messages cryptographiques*. C'est le premier manuscrit connu faisant mention des fréquences d'apparition des lettres.

Activité 12

Etudions **plusieurs programmes en Python 3** permettant d'effectuer les tâches répétitives rencontrées lors de la cryptanalyse du chiffrement par substitution avec une attaque statistique.

Le premier programme est une fonction Python 3 dont l'entrée est *une chaîne de caractère qui dans notre cas sera le texte chiffré* et, en sortie, donne *une liste contenant la fréquence d'apparition des lettres* contenu dans le texte chiffré. La somme de ces fréquences vaut un.

Programme Python 3 (zone de script)

```
# Calcul des fréquences d'apparition des lettres dans un texte

def STAT_TEXTE(texte):

    compteur=[0 for i in range(26)]           # Liste contenant des 0
    for lettre in texte:                       # On parcourt le texte
        k=ord(lettre)-65                       # On utilise le code ASCII
        if 0<=k<26:
            compteur[k]=compteur[k]+1         # Compte le nombre de lettres
    frequence=[ ]
    for j in range(26):                       # On calcule la fréquence des lettres
        frequence.append(compteur[j]/len(texte))
    return frequence
```

Sortie Python 3 (console)

```
In : STAT_TEXTE( ' ETEOTLYXFLLQORTHKQLTCKQOT ' )

Out : [0.0 , 0.0 , 0.03703 , 0.0 ,0.07407 , 0.03703 , 0.0 , 0.03703 , 0.03703 , 0.0 ,
       0.07407 , 0.14814 , 0.0 , 0.0 , 0.11111 , 0.0 , 0.11111 , 0.03703 , 0.0 ,
       0.22222 , 0.0 , 0.0 , 0.0 , 0.03703 , 0.03703 , 0.0]
```

Le deuxième programme est aussi une fonction Python 3 dont l'entrée est une chaîne de caractère qui dans notre cas sera le texte chiffré. Cette fonction utilise la fonction précédente pour afficher en sortie, conjointement les fréquences d'apparition des lettres de la langue française avec celle du texte chiffré à l'aide de diagrammes en bâtons afin de pouvoir comparer facilement les fréquences.

Programme Python 3 (zone de script)

```
def DIAGRAMME_FREQ(texte):

    # Frequences d apparition des lettres dans la langue francaise (ordre : E_S_A_I_N)
    frequ_lettres_Fr=[0.0808,0.0106,0.0303,0.0418,0.1726,0.0112,0.0127,0.0092,
0.0734,0.0031,0.0005,0.0601,0.0296,0.0713,0.0526,0.0301,0.0099,0.0655,0.084,
0.0707,0.0574,0.0132,0.0004,0.0045,0.0030,0.0012]

    # Diagrammes en batons des frequences

    import matplotlib.pyplot as plt
    plt.figure(figsize=(12,6),dpi=100)

    for k in range(26):
        p1, = plt.plot([k-0.2,k-0.2], [0, STAT_TEXTE(texte)[k]], 'r-', linewidth=10)
        p2, = plt.plot([k+0.2,k+0.2], [0, frequ_lettres_Fr[k]], 'g-', linewidth=10)

    plt.xlabel("Lettres")
    plt.ylabel("Frequences")
    plt.xticks([k for k in range(26)], [chr(k+65) for k in range(26)])
    plt.legend([p1,p2], ['rouge (freq. lettres texte chiffre)', 'vert (freq. th. lettres francais)'])

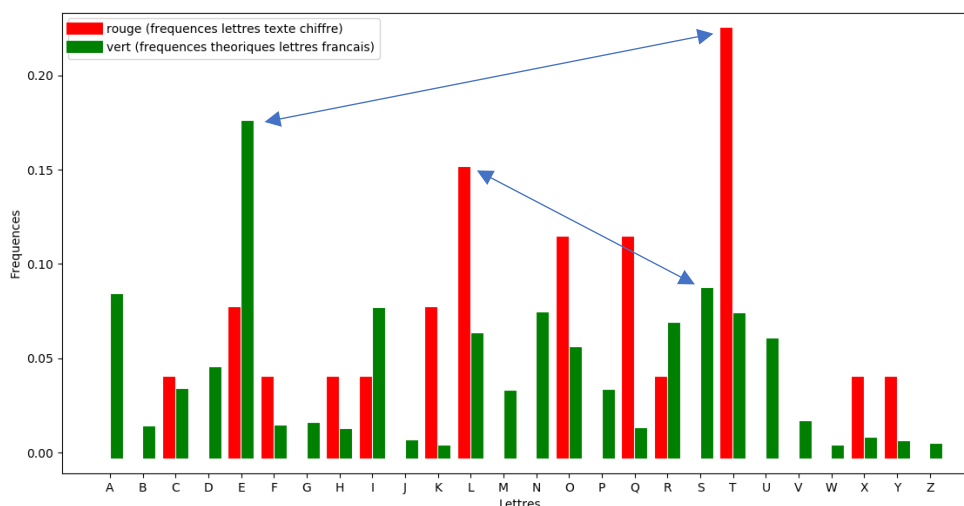
    plt.show()

    return None
```

Sortie Python 3 (console)

In : DIAGRAMME_FREQ (' ETEOTLYXFTLLQORTHILKQLTCKQOT ')

Out :



Remarque Ces diagrammes en bâtons permettent facilement d'effectuer les correspondances entre les lettres les plus fréquentes de la langue française et celle du texte chiffré.

On fait l'hypothèse que : **T** chiffre la lettre **E**, **L** chiffre la lettre **S**, etc.

Le troisième programme est une fonction Python 3 qui permet au cryptanalyste après avoir effectué une analyse des fréquences du texte chiffré, d'effectuer les substitutions d'une lettre « chiffee » (en majuscule) par une lettre « en clair » (en minuscule) dans le texte.

Programme Python 3 (zone de script)

```
# Remplace dans la chaine de caractere 'texte' toutes les occurences  
# de la lettre ' Li ' par la lettre ' Lf '
```

```
def SUB_LETTRE(texte,Li,Lf):  
    texte_hyp=""  
    for lettre in texte:  
        if lettre==Li:  
            texte_hyp=texte_hyp+Lf  
        else:  
            texte_hyp=texte_hyp+lettre  
    return texte_hyp
```

Sortie Python 3 (console)

```
In : SUB_LETTRE('ETEOTLYXFTLLQORTHIKQLTCKQOT','T','e')
```

```
Out : 'EeEOeLYXFeLLQOReHIKQLeCKQOe'
```

```
In : SUB_LETTRE('EeEOeLYXFeLLQOReHIKQLeCKQOe','L','s')
```

```
Out : 'EeEOesYXFessQOReHIKQseCKQOe'
```

```
In : SUB_LETTRE('EeEOesYXFessQOReHIKQseCKQOe','Q','a')
```




```
Out : 'EeEOesYXFessaOReHIKaseCKaOe'
```

```
In : SUB_LETTRE('EeEOesYXFessaOReHIKaseCKaOe','O','i')
```

```
Out : 'EeEiesYXFessaiReHIKaseCKaie'
```



Exercice 13

Considérons la situation suivante :

- Vous êtes **OSCAR**  un cryptanalyste engagé par les services secrets de votre pays.
- **ALICE**  et **BOB**  sont des terroristes qui s'envoient des emails par internet contenant des textes chiffrés afin d'organiser un attentat.
- Vous avez comme mission, après interception des textes chiffrés envoyé par ALICE à BOB d'effectuer une cryptanalyse sur les textes chiffrés et d'obtenir les textes clairs. Le système cryptographique utilisé par ALICE et BOB est connu : c'est *le chiffrement par substitution* (principe de Kerckhoffs).

a) **Ecrire** dans un éditeur **les cinq fonctions Python** ci-dessous (étudiées en cours) vous permettant de systématiser, en partie, la cryptanalyse du chiffrement par substitution. 

```
SUB_CHIFFRE(texte_clair,clef)          STAT_TEXTE(texte)
SUB_DECHIFFRE(texte_chiffre,clef)      DIAGRAMME_FREQ(texte)
                                         SUB_LETTE(texte,Li,Lf)
```

b) **Effectuer**, à l'aide des cinq fonctions Python 3 ci-dessus, **une cryptanalyse** sur les deux textes chiffrés suivants : 

texte_chiffre_01 = 'RJISQQSMRIPSJKLSINOGONQBJQINJSQBOJSLIKQGNIVRISPBXAIRNIK'

texte_chiffre_02 = 'LIKIXGLOKBDKKOJQPSJKLIKORKKOLPILBHHIRTLIPRKIGQNRIPKGOJQK'

Indications :

- On fera l'hypothèse que les textes ont été chiffrés avec la même clef.
- Dans les textes on suppose que les mots : 'ATTAQUE ' et 'EXPLOSIFS ' ont été utilisés.

c) **Déterminer** la clef de chiffrement / déchiffrement (permutation π).

Alphabet en clair 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
Alphabet chiffré y																										

Remarque : Si l'image d'une lettre par π n'est pas connue, on utilise le caractère ' * '.

Nom du fichier : ex_13_cryptanalyse_substitution.py

Remarques

Le système de chiffrement par substitution n'assure pas :

- 1) **la confidentialité** ; garantir que seul l'expéditeur et le destinataire pourront avoir une vision intelligible du message. En effet, on peut en un temps raisonnable, effectuer une attaque statistique si le texte est assez long.
- 2) **l'authenticité** du message ; garantir l'identité de l'expéditeur.
- 3) **l'intégrité** du message ; garantir qu'il n'a pas été modifié par une tierce personne.

1.3.6 Chiffrement de Vigenère

Dans le cas du chiffrement par substitution (décalage et affine compris), dès qu'une clef est fixée, chaque caractère alphabétique est transformé en un unique caractère alphabétique. Pour cette raison, le procédé est appelé **mono-alphabétique**.

On présente maintenant **le chiffrement de Vigenère** qui n'est pas mono-alphabétique et qui ne pourra pas être « cryptanalysé » par une méthode exhaustive ou statistique « classique ». Son nom provient de *Blaise de Vigenère* qui vécut au seizième siècle.

L'idée du chiffrement de Vigenère est de définir un clef K qui est une chaîne de caractère de longueur m appelé **mot-clef**. On chiffre le texte clair **par bloc** de m caractères alphabétiques à la fois.



Blaise de Vigenère
(1523 - 1596)

Explication / exemple

(1.12)

- ALICE prend par exemple comme mot clef : ' CODE ' qui est de longueur $m = 4$.
Convertissons cette chaîne de caractère en nombre avec la bijection b .

lettre	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
b(lettre)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Cela donne la clef : $K = (2; 14; 3; 4)$

Prenons maintenant le texte clair : ' CEMESSAGEESTSECRET ' que nous voulons chiffrer.

Le chiffrement de Vigenère consiste à effectuer sur chaque bloc de longueur $m = 4$ du texte clair un chiffrement par décalage (César) et dont le décalage dépend du rang de la lettre dans un bloc.

texte clair	C	E	M	E	S	S	A	G	E	E	S	T	S	E	C	R	E	T
nb clair x_i	2	4	12	4	18	18	0	6	4	4	18	19	18	4	2	17	4	19
mot clef	C	O	D	E	C	O	D	E	C	O	D	E	C	O	D	E	C	O
décalage k_i	2	14	3	4	2	14	3	4	2	14	3	4	2	14	3	4	2	14
nb chiffré y_i $\equiv x_i + k_i \pmod{26}$	4	18	15	8	20	6	3	10	6	18	21	23	20	18	5	21	6	7
texte chiffré	E	S	P	I	U	G	D	K	G	S	V	X	U	S	F	V	G	H


La fonction de chiffrement C_K bijective avec la clef $K = (2; 14; 3; 4)$ et $m = 4$ est :

$$\mathbb{Z}_{26}^4 \rightarrow \mathbb{Z}_{26}^4$$

$$x = (x_0; x_1; x_2; x_3) \rightarrow C_K(x) = \left(\underbrace{x_0 + \overset{k_0}{2}}_{\pmod{26}}; \underbrace{x_1 + \overset{k_1}{14}}_{\pmod{26}}; \underbrace{x_2 + \overset{k_2}{3}}_{\pmod{26}}; \underbrace{x_3 + \overset{k_3}{4}}_{\pmod{26}} \right) = (y_0; y_1; y_2; y_3)$$

Remarque

Ce procédé est **poly-alphabétique** puisque dans notre exemple chaque caractère alphabétique n'est pas transformé en un unique caractère alphabétique. Dans l'exemple précédant la lettre **E** a été chiffrée successivement par **S, I, G, S, S** et **G** qui n'est pas la même lettre.

- **BOB**  a besoin d'une **fonction de déchiffrement** D_k bijective qui est la réciproque de C_k c'est-à-dire $D_k(C_k(x)) = x = I_d(x)$ sinon il ne pourra pas déchiffrer le message d'ALICE :


$$\mathbb{Z}_{26}^4 \rightarrow \mathbb{Z}_{26}^4$$

$$y = (y_0; y_1; y_2; y_3) \rightarrow D_K(y) = \left(\underbrace{y_0 - \overset{k_0}{2}}_{\text{mod } 26}; \underbrace{y_1 - \overset{k_1}{14}}_{\text{mod } 26}; \underbrace{y_2 - \overset{k_2}{3}}_{\text{mod } 26}; \underbrace{y_3 - \overset{k_3}{4}}_{\text{mod } 26} \right) = (x_0; x_1; x_2; x_3)$$

texte chiffré	E	S	P	I	U	G	D	K	G	S	V	X	U	S	F	V	G	H
nb chiffré y_i	4	18	15	8	20	6	3	10	6	18	21	23	20	18	5	21	6	7
mot clef	C	O	D	E	C	O	D	E	C	O	D	E	C	O	D	E	C	O
décalage k_i	2	14	3	4	2	14	3	4	2	14	3	4	2	14	3	4	2	14
nb clair x_i $\equiv y_i - k_i \text{ mod } 26$	2	4	12	4	18	18	0	6	4	4	18	19	18	4	2	17	4	19
texte clair	C	E	M	E	S	S	A	G	E	E	S	T	S	E	C	R	E	T

Remarques

- On utilise la même clef $K = (k_0; k_1; k_2; k_3) = (2; 14; 3; 4)$ pour le chiffrement et le déchiffrement.
- Le mot-clef est simple à transmettre, les modes de chiffrement et de déchiffrement sont simples.

On peut proposer **le programme Python 3** ci-dessous qui permet de systématiser **la tâche de chiffrement** d'un texte clair par **la méthode de Vigenère**. Le programme est constitué d'une **fonction Python** à deux paramètres **texte_clair** et **mot_clef**. 

```
def VIGENERE_CHIFFRE(texte_clair, mot_clef): # mot_clef est une chaîne de caractère

    clef = [ ]
    for lettre in mot_clef: # Boucle sur mot_clef
        nb_clef = ord(lettre) - 65 # Bijection b
        clef.append(nb_clef) # clef est une liste de nombre [k1, k2, ..., km]

    texte_chiffre = [ ]
    m = len(clef) # Calcule la longueur de la clef
    i = 0 # Rang dans le bloc
    for lettre in texte_clair: # Boucle sur texte clair
        nb_clair = ord(lettre) - 65 # Bijection b
        nb_chiffre = (nb_clair + clef[i]) % 26 # Fonction de chiffrement
        lettre_chiffre = chr(nb_chiffre + 65) # Bijection b^-1
        i = (i + 1) % m # On passe au rang suivant
        texte_chiffre.append(lettre_chiffre)
    texte_chiffre = " ".join(texte_chiffre) # Conversion liste en chaîne de caractère

    return texte_chiffre
```

Exercice 14

a) Compléter le tableau afin de tester la fonction *py3* : **VIGENERE_CHIFFRE(texte_clair,mot_clef)**



	texte_clair	mot_clef	nb_clef	clef	m	nb_clair	nb_chiffre	lettre_chiffre	i	texte_chiffre
Entrée	' MATH '	' OK '								
Initialisation										
Boucle 1										
Initialisation										
Boucle 2										
Conversion										
Sortie										

b) Ecrire dans un éditeur et comprendre la fonction *Python 3* :

VIGENERE_CHIFFRE(texte_clair,mot_clef) qui permet de **systématiser** la tâche de chiffrement de Vigenère d'un *texte clair* avec la clef : *mot_clef*.



c) Ecrire dans un éditeur et comprendre la fonction *Python 3* :

VIGENERE_DECHIFFRE(texte_chiffre,mot_clef) qui permet de **systématiser** la tâche de déchiffrement de Vigenère d'un *texte chiffré* avec la clef : *mot_clef*.



Exemple : Sortie Python (console):

```
In : VIGENERE_CHIFFRE('MATH','OK')
Out : 'AKHR'

In : VIGENERE_DECHIFFRE('AKHR','OK' )
Out : 'MATH'

In : VIGENERE_CHIFFRE('BATAILLE','AIR')
Out : 'BIKAQCLM'

In : VIGENERE_DECHIFFRE('BIKAQCLM','AIR' )
Out : 'BATAILLE'
```

Nom du fichier : ex_14_chiffrement_Vigenere.py

Exercice 15

Dans cet exercice, on n'utilisera pas de programme informatique pour résoudre les problèmes.

- a) Chiffrer le texte clair : ' BATAILLE ' à l'aide du *chiffrement de Vigenère* avec le mot clef ' AIR '. Remarque : c'est le travail d'ALICE.

texte clair	B	A	T	A	I	L	L	E
nb clair								
mot clef								
décalage								
nb chiffré								
texte chiffré								

- b) Déchiffrer le texte chiffré : ' SUFXUHHH ' à l'aide du *chiffrement de Vigenère* avec le mot clef ' STOP '. Remarque : c'est le travail de BOB.

texte chiffré	S	U	F	X	U	H	H	H
nb chiffré								
mot clef								
décalage								
nb clair								
texte clair								

Exercice 16

Voici le **carré de Vigenère** qui permet d'appliquer la méthode de chiffrement et déchiffrement de Vigenère.

	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	
A	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	0
B	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	A	1
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	2
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	3
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	4
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	5
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	6
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	7
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	8
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	9
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	10
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	11
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	12
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	13
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	14
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	15
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	16
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	17
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	18
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	19
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	20
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	21
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	22
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	23
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	24
Z	Z	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	25

La lettre de la clef (en rose) est dans la colonne la plus à gauche, la lettre du message clair (en vert) est dans la ligne tout en haut. La lettre chiffrée (en jaune) est à l'intersection de la ligne de la lettre clef et de la colonne de la lettre claire.

Exemple : chiffrons le texte clair 'CHIFFREDEVIGENERE' avec la clef 'BACHELIER'

Clair	C	H	I	F	F	R	E	D	E	V	I	G	E	N	E	R	E
Clef	B	A	C	H	E	L	I	E	R	B	A	C	H	E	L	I	E
Décalage	1	0	2	7	4	11	8	4	17	1	0	2	7	4	11	8	4
Chiffré	D	H	K	M	J	C	M	H	V	W	I	I	L	R	P	Z	I


Enoncé Sans utiliser d'ordinateur (programme), utiliser *le carré de Vigenère* pour :

- 1) chiffrer le texte clair : 'BATAILLE' avec le mot clef 'AIR'.
- 2) déchiffrer le texte chiffré : 'BIKAQCLM' avec la clef 'AIR'.
- 3) chiffrer le texte clair : 'ABRICOTS' avec la clef 'STOP'.
- 4) déchiffrer le texte chiffré : 'SUFXUHHH' avec la clef 'STOP'.

Cryptanalyse du chiffrement de Vigenère

- a) Il y a dans ce procédé de chiffrement la possibilité de créer 26^m clefs K car une clef est un m-uplet $K = (k_1; k_2; \dots; k_m)$. Pour k_1 il y a 26 choix, pour k_2 il y a 26 choix, etc. donc $\underbrace{26 \cdot 26 \cdot \dots \cdot 26}_{m \text{ fois}} = 26^m$.

Pour des blocs de longueur $m = 4$ cela donne déjà $26^4 = 456'976$ clefs différentes et donc autant de fonctions de chiffrement. Même si un ordinateur teste toutes les combinaisons possibles sans problème, il n'est pas question de parcourir cette liste pour trouver le texte clair, c'est-à-dire celui qui est compréhensible. Une **attaque exhaustive** a donc peu de sens avec cette méthode.

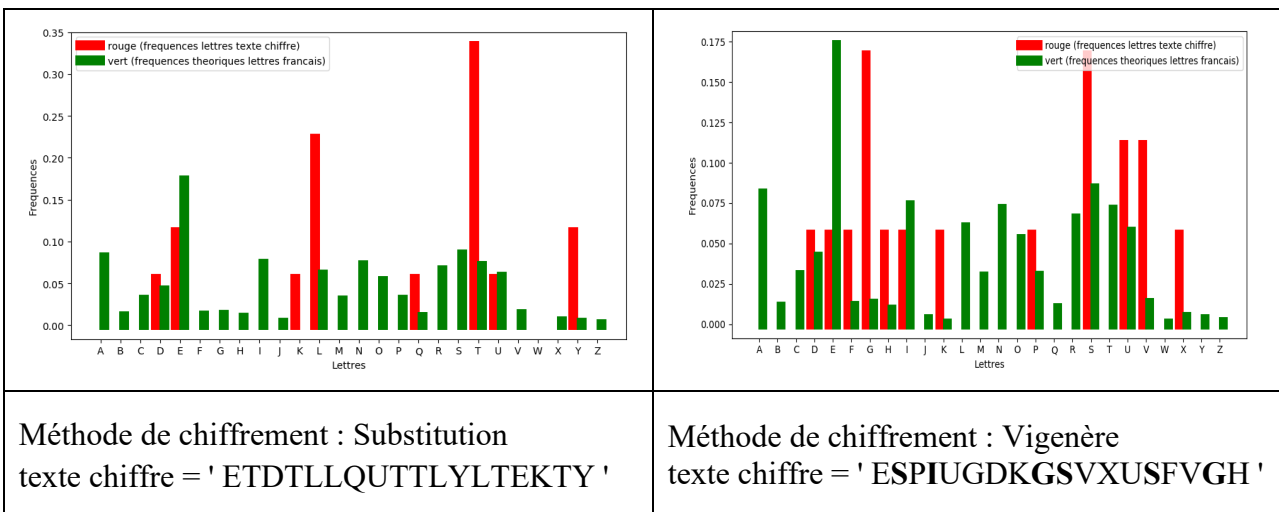
- b) Le cryptanalyste  est en difficulté s'il étudie la fréquence d'apparition des lettres.

En effet, une même lettre du texte clair peut être chiffrée par m lettres différentes.

Dans l'exemple précédant la lettre E a été chiffrée successivement par S, I, G, S, S et G qui n'est pas la même lettre.

texte clair = 'CEMESSAGEESTSECRET'

Comparaison des fréquences d'apparition des lettres de la langue française avec celle du texte chiffré :



Une **attaque statistique** de ce type ne fonctionne donc plus avec le chiffrement de Vigenère.



C'est encore ce principe de chiffrement qui était utilisé par l'armée allemande pendant la Seconde Guerre mondiale et qui était automatisé avec *la machine Enigma*.



Le Mathématicien anglais **Alan Turing** (1912-1954) joue un rôle majeur dans la cryptanalyse de la machine Enigma, utilisée par les armées allemandes. Ses méthodes aidèrent grandement à casser ce chiffrement et, selon plusieurs historiens, de raccourcir la Seconde Guerre mondiale de deux ans.



Lien vers le documentaire /vidéo :
« la drôle de guerre d'Alan Turing ».

- c) Ce chiffrement a résisté trois siècles aux cryptanalystes. Plusieurs méthodes ont été mises au point. On peut citer : la méthode de *Charles Babagge et Friedrich Kasiski*, la méthode du **mot probable** conçue par le commandant *Bazeries*, le calcul de **l'indice de coïncidence**, etc.

Méthode du «mot probable»


Le *commandant Bazeries* est l'inventeur d'une méthode de cryptanalyse relativement simple. Elle se base sur l'existence d'un **mot probable** et préconise la recherche du **mot clef**.

Étant donné un texte chiffré au moyen du *chiffrement de Vigenère* et renfermant un mot supposé connu, on « soustrait » le **mot probable** à une **séquence du texte chiffré** de même longueur jusqu'à ce que **la clef** apparaisse.

Rappel

lettre	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
b(lettre)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Exemple

OSCAR  (le cryptanalyste) essaie de déchiffrer le texte chiffré :

"ESPIUGDKGSVXUSFVGH" qui est supposé renfermer le mot probable "MESSAGE".
Il sait que le chiffrement de Vigenère a été utilisé.

En soustrayant le mot probable "MESSAGE" à la séquence débutant à la **position 1** du message chiffré, on obtient :

texte chiffré	E	S	P	I	U	G	D
nb chiffré y_i	4	18	15	8	20	6	3
texte clair	M	E	S	S	A	G	E
nb clair x_i	12	4	18	18	0	6	4
décalage $k_i \equiv y_i - x_i \text{ mod } 26$	18	14	23	16	20	0	25
mot clef	S	O	X	Q	U	A	Z

ce qui semble ne rien donner. En commençant en **position 2** on obtient :

texte chiffré	S	P	I	U	G	D	K
nb chiffré y_i	18	15	8	20	6	3	10
texte clair	M	E	S	S	A	G	E
nb clair x_i	12	4	18	18	0	6	4
décalage $k_i \equiv y_i - x_i \text{ mod } 26$	6	11	16	2	6	23	6
mot clef	G	L	Q	C	G	X	G

ce qui ne semble pas meilleur. On continue ainsi jusqu'à la **position 3** et :




texte chiffré	P	I	U	G	D	K	G
nb chiffré y_i	15	8	20	6	3	10	6
texte clair	M	E	S	S	A	G	E
nb clair x_i	12	4	18	18	0	6	4
décalage $k_i \equiv y_i - x_i \text{ mod } 26$	3	4	2	14	3	4	2
mot clef	D	E	C	O	D	E	C

Le mot "**CODE**" est apparu. C'est probablement le **mot-clef** que l'on cherchait.
En utilisant cette clef, le déchiffrement donne : "CEMESSAGEESTSECRET".


Remarque

Le *mot probable* doit avoir autant ou plus de caractères que le *mot clef*.

Exercice 17

Supposons qu'OSCAR  intercepte un texte chiffré envoyé par ALICE  à BOB .

On fait l'hypothèse, qu'OSCAR connaît le système de chiffrement utilisé qui est dans cet exercice le *chiffrement de Vigenère* (voir principe de Kerckhoffs).

- a) Ecrire *un programme en Python 3* permettant « d'attaquer » par la méthode dite du **mot probable**, le *chiffrement de Vigenère* (voir exemple précédant). 

Exemple : Sortie Python 3 (console)

<pre>texte_chiffre_01="ESPIUGDKGSVXUSFVGH" mot_prob_01="MESSAGE" Out : sequ_texte_ch_ ESPIUGD mot_prob_____ MESSAGE mot_clef_____ SOXQUAZ sequ_texte_ch_ SPIUGDK mot_prob_____ MESSAGE mot_clef_____ GLQCGXG </pre>	<pre>sequ_texte_ch_ PIUGDKG mot_prob_____ MESSAGE mot_clef_____ DECODEC sequ_texte_ch_ IUGDKGS mot_prob_____ MESSAGE mot_clef_____ WQOLKAO </pre>
---	--

- b) Aider OSCAR à faire **une cryptanalyse** sur le texte chiffré :

"WOLVIIXNJVCORUEHDVXLVQBRJTKDYHNCVPIGRNGHZGNSZPXBATRQBRB"
qui est supposé renfermer le mot probable "ATTAQUE".

Indications : *Chiffrement Vigenère* : $x_i \rightarrow x_i + k_i \equiv y_i \text{ mod } 26$

Déchiffrement Vigenère : $y_i \rightarrow y_i - k_i \equiv x_i \text{ mod } 26$

Cryptanalyse Vigenère : $y_i \rightarrow y_i - x_i \equiv k_i \text{ mod } 26$

Nom du fichier : **ex_17_cryptanalyse_Vigenere.py**

Remarques

Le système de chiffrement de Vigenère n'assure pas :

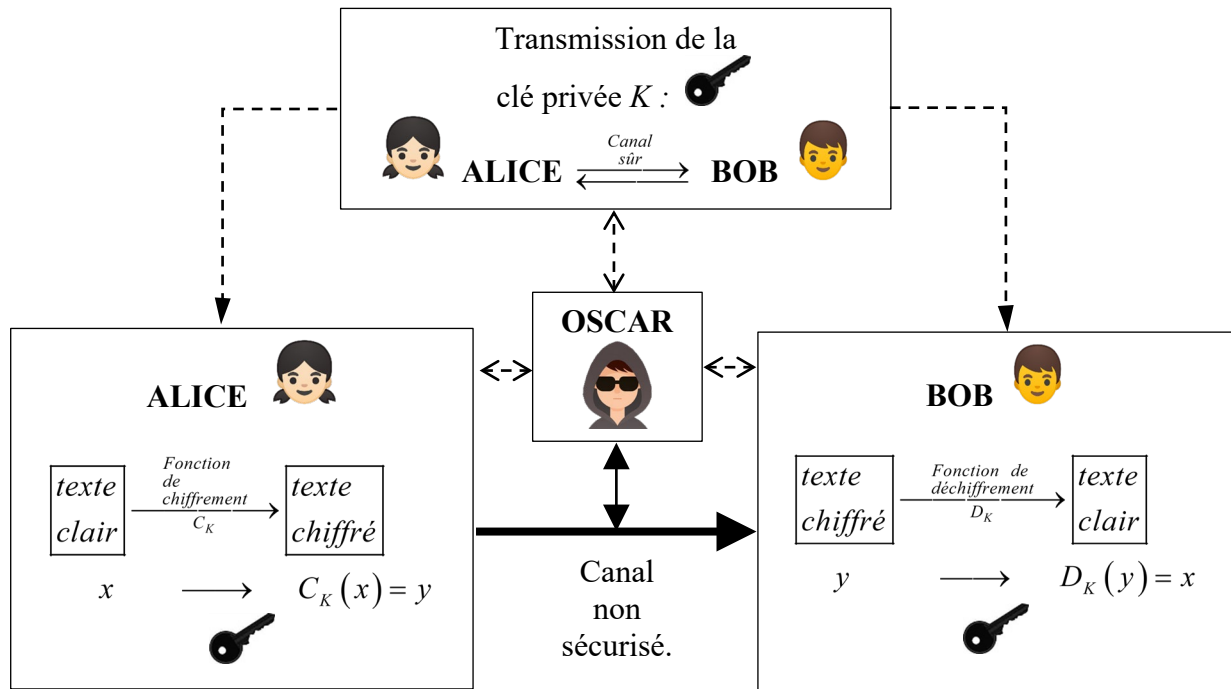
- 1) *la confidentialité* ; garantir que seul l'expéditeur et le destinataire pourront avoir une vision intelligible du message. En effet, on peut en un temps raisonnable, effectuer une attaque par mot probable.
- 2) *l'authenticité* du message ; garantir l'identité de l'expéditeur.
- 3) *l'intégrité* du message ; garantir qu'il n'a pas été modifié par une tierce personne.

1.3.7 Concept de chiffrement symétrique

Définition

Une méthode de chiffrement est dite **symétrique** ou encore à **clé privée**, si elle utilise **une même clé pour chiffrer et déchiffrer un texte (message)**.

Schéma ALICE (l'expéditrice) veut envoyer un texte à **BOB** (le destinataire) sur un canal non sécurisé. **OSCAR** est l'opposant (le hacker ou le cryptanalyste). (1.13)



C_k et D_k sont bijectives et vérifient : $C_k(D_k(y)) = y = I_d(y)$ et $D_k(C_k(x)) = x = I_d(x)$
Autrement dit, C_k et D_k sont réciproques l'une de l'autre.

Exemple

Le **chiffrement par décalage (de César)** est un chiffrement symétrique. Il peut être utilisé par ALICE et BOB que s'ils se sont transmis au préalable la même clé privée qui est un nombre entier $K \in \{0; 1; 2; \dots; 24; 25\}$ représentant le décalage des lettres de l'alphabet.

Remarques

a) Un **chiffrement symétrique** présente **deux inconvénients** majeurs :

1) la **transmission** de la clé entre les interlocuteurs que sont ALICE et BOB.

En effet, la clé qui doit rester totalement confidentielle, doit être transmise au correspondant et de façon sûre. Toute interception par OSCAR, détruit la *confidentialité* de l'échange d'information entre ALICE et BOB.

2) l'**authentification** par BOB de l'expéditrice qui est ici ALICE.

Si OSCAR c'est emparé de la clé, il pourrait très bien fabriquer de faux messages ou changer subrepticement une partie d'un message intercepté ; l'intégrité du message n'est donc pas assuré. Ces falsifications demeurent indétectable pour BOB.

Ces deux inconvénients **sont illustrés par la situation concrète** suivante :

ALICE (l'expéditrice) communique avec sa banque via internet (e-banking) et donne **un ordre de paiement** à son banquier, BOB (le destinataire). Pour que leurs communications restent confidentielles ils utilisent un système de chiffrement symétrique dont la clé privée est K .

1) ALICE n'habite pas dans la même ville que le siège de la banque ou travaille BOB.

Pour se transmettre la clé privée K ils doivent **se déplacer**.

Il est exclu que BOB envoie la clé privée dans **un mail non chiffré** ou dans une lettre et par la poste à ALICE car OSCAR peut intercepter le mail (ou la lettre) et donc la clé.

Il est inutile que BOB envoie la clé privée K dans **un mail chiffré** à ALICE car elle ne pourra pas déchiffrer le mail ; elle n'est pas encore en possession de la clé.

2) Supposons qu'OSCAR ait pu intercepter l'ordre de paiement chiffré par ALICE et qu'il arrive avec une méthode de cryptanalyse (par exemple la force brute) a **obtenir l'ordre de paiement en clair** et aussi **la clé privée K** .

OSCAR décide de **modifier le contenu de l'ordre de paiement** par exemple, le montant ou le nom du destinataire. BOB **ne sera pas en mesure de vérifier l'authenticité ou l'intégrité du message**.

En l'absence d'une **signature** de l'expéditeur, ces falsifications demeurent indétectables.

b) Jusqu'à la veille des années 1970, toutes les méthodes de chiffrement étaient **symétriques** et ne possédaient pas de **signatures** :

Chiffrement de César, affine, substitution, Vigenère, etc.

1.3.8 Concept de chiffrement asymétrique

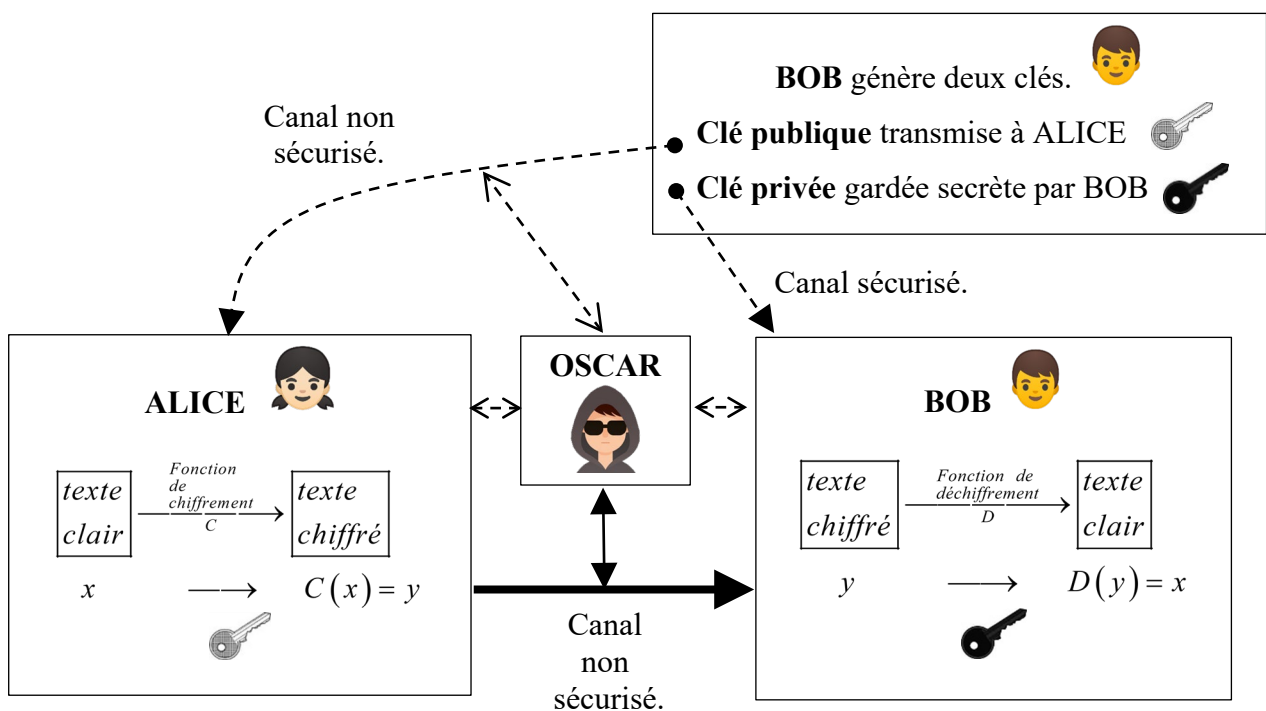
Le problème de la **transmission des clés** et de l'**authentification de l'expéditeur** été résolu grâce au **chiffrement asymétrique** et en particulier grâce au **système de chiffrement RSA** dont nous allons décrire les étapes et les concepts mathématiques en détails dans le prochain chapitre.

Définition

Une méthode de chiffrement est dite **asymétrique** ou encore **à clé publique**, si elle utilise deux clés : une **clé publique pour chiffrer** un texte (message) et une autre **clé privée pour le déchiffrer**.

La clé privée ne doit **pas pouvoir être déduite** à partir de la clé publique.

Schéma ALICE (l'expéditrice) veut envoyer un texte à **BOB** (le destinataire) sur un canal non sécurisé. **OSCAR** est l'opposant (le hacker ou le cryptanalyste). (1.14)



C et D sont bijectives et vérifient : $C(D(y)) = y = I_d(y)$ et $D(C(x)) = x = I_d(x)$

Autrement dit, C et D sont réciproques l'une de l'autre.

Remarques

a) BOB génère deux clefs :

- une **clef publique** qu'il diffuse largement (sur internet par exemple) et qui va permettre à ALICE de chiffrer son texte. OSCAR peut lui aussi chiffrer des messages avec la clef publique.
- une **clef privée** qu'il conserve et gardée secrète. Cette clef va permettre à BOB de déchiffrer le texte chiffré par ALICE ou par une autre personne ayant reçu la clef publique.

Autrement dit, tout le monde peut chiffrer un texte destiné à BOB, mais il est le seul à pouvoir le déchiffrer.

Il y a une condition cruciale pour que ce système fonctionne : il doit être impossible pour une tierce personne, par exemple OSCAR, de déduire la clé privée à partir de la clé publique ou à partir du texte chiffré intercepté.

Le problème de la **transmission des clés** entre interlocuteurs distant est alors résolu : il n'est pas nécessaire de se voir au préalable pour partager une unique clé privée. La communication chiffrée est alors possible sur des réseaux informatiques non-sécurisé. Cela permet développer le commerce en ligne.

- b) OSCAR peut cependant **intercepter** le texte chiffré par ALICE et envoyé à BOB sur le canal peu sûr et le remplacer par son propre texte chiffré (la fonction de chiffrement étant publique). OSCAR peut ainsi **se faire passer** pour ALICE et BOB n'en sera rien.

L'**authenticité** comme l'**intégrité** du message n'est alors pas assurée.

On peut contourner ce problème grâce au concept **d'authentification par signature** que nous étudierons dans le prochain chapitre et basé sur le **système de chiffrement RSA**.

- c) Le **système de chiffrement RSA** paraît résister encore à notre époque aux algorithmes de cryptanalyse les plus récents et à la puissance de calculs des supers ordinateurs moyennant certaines précautions. Les ordinateurs quantiques avec des « algorithmes quantiques » risquent peut-être de faire « tomber » ce chiffrement.

Le principe de Kerckhoffs

Auguste Kerckhoffs von Nieuwenhoff (1835-1903) est un cryptologue militaire néerlandais.



Le principe de *Kerckhoffs* s'énonce ainsi :

«La sécurité d'un système de chiffrement ne doit reposer que sur la clé.»

Autrement dit :

« L'ennemi peut avoir connaissance du système de chiffrement (fonction de chiffrement et de déchiffrement) mais pas de la clé.»

Ce principe est novateur dans la mesure où intuitivement il semble opportun de dissimuler le maximum de choses possibles à l'ennemi : clé et système de chiffrement utilisés.

Ce principe se base sur **deux constats de Kerckhoffs** :

- 1) le système de chiffrement sera forcément connu un jour ou l'autre de l'ennemi (secret vendu par un traître).
- 2) un système de chiffrement connu de tous sera testé, attaqué, étudié, et finalement utilisé s'il s'avère intéressant et robuste.

Question 01

Quels concepts mathématiques faut-il utiliser pour « construire » deux clés, une privée et une publique, dont il n'est pas possible en un temps « raisonnable » de déduire la clé privée à partir de la clé publique sachant que le système de chiffrement est connu de tous ?

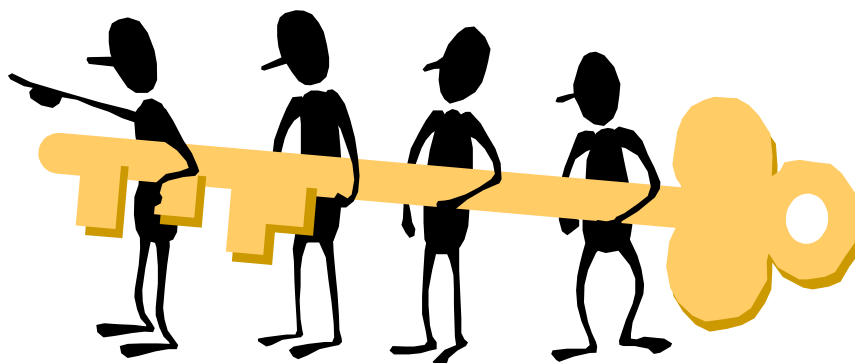
La réponse à cette question se trouve dans la deuxième partie du cours avec l'étude du **chiffrement RSA** faisant intervenir le produit de deux grands nombres premiers.

CRYPTOGRAPHIE

PARTIE 1

CORRECTIONS

ACTIVITES ET EXERCICES



Correction exercice 1

a) Utilisation de la fonction Python 3 : **CESAR_CHIFFRE(texte_clair,k)**



	texte_clair	k	nb_clair	nb_chiffre	lettre_chiffre	texte_chiffre
Entrée	'ICI '	3				
Initialisation	'ICI '	3				[]
Boucle	'ICI '	3	8	11	L	[L]
	'ICI '	3	2	5	F	[L,F]
	'ICI '	3	8	11	L	[L,F,L]
Conversion	'ICI '	3				'LFL'
Sortie						'LFL'

b) Programme en Python 3 (zone de script)



```
def CESAR_CHIFFRE(texte_clair,k):
    texte_chiffre= [ ]
    for lettre in texte_clair:
        nb_clair=ord(lettre)-65
        nb_chiffre=(nb_clair+k)%26
        lettre_chiffre=chr(nb_chiffre+65)
        texte_chiffre.append(lettre_chiffre)
    texte_chiffre="".join(texte_chiffre)
    return texte_chiffre
```

Boucle
Bijection b
Fonction de chiffrement
Bijection b^{-1}
Conversion liste en chaîne de caractère

c) Programme Python 3 (zone de script)



```
def CESAR_DECHIFFRE(texte_chiffre,k):
    texte_clair= [ ]
    for lettre in texte_chiffre:
        nb_chiffre=ord(lettre)-65
        nb_clair=(nb_chiffre-k)%26
        lettre_clair=chr(nb_clair+65)
        texte_clair.append(lettre_clair)
    texte_clair="".join(texte_clair)
    return texte_clair
```

Boucle
Bijection b
Fonction de déchiffrement
Bijection b^{-1}
Conversion liste en chaîne de caractère

Exemple : Sortie Python (console)

```
In : CESAR_CHIFFRE('ZEBRE',3)
Out : 'CHEUH'

In : CESAR_DECHIFFRE('CHEUH',3)
Out : 'ZEBRE'
```

Nom du fichier : ex_1_chiffrement_Cesar.py

Remarque :

$$\boxed{D_k = C_{26-k}} \quad \text{tel que } k \text{ est l'opposé de } 26 - k \text{ modulo } 26$$

Explication :

$$\begin{aligned} \text{Soit } 0 \leq x \leq 25. \quad x &\xrightarrow{C_k} \underbrace{x + k \bmod 26}_{=y} \xrightarrow{C_{26-k}} y + 26 - k \bmod 26 \\ &= x + k + 26 - k \bmod 26 \\ &= x + 26 \bmod 26 \\ &= x + 0 \bmod 26 \\ &= x \end{aligned}$$

Correction Exercice 2

a) Chiffrons le texte clair :

'BATAILLE' à l'aide du *chiffrement par décalage* de clef $K = 10$.

texte clair	B	A	T	A	I	L	L	E
nb clair : x	1	0	19	0	8	11	11	4
nb chiffré : $C_3(x) = x + 10 \bmod 26 = y$	11	10	3	10	18	21	21	14
texte chiffré	L	K	D	K	S	V	V	O

b) Déchiffrons le texte chiffré :

'REYPKENA' à l'aide du *chiffrement par décalage* de clef $K = 22$.

texte chiffré	R	E	Y	P	K	E	N	A
nb chiffré : y	17	4	24	15	10	4	13	0
nb clair : $D_3(y) = y - 22 \bmod 26 = x$	21	8	2	19	14	8	17	4
texte clair	V	I	C	T	O	I	R	E

Correction Exercice 3

a) Programme Python 3 (zone de script)

```
texte_chiffre_01 = 'INKANNWLJPN'
for k in range(26):
    print("D",k," -> ",CESAR_DECHIFFRE(texte_chiffre_01,k))
```

b) Sortie Python (console)

Out :	Out :	Out :
D 0 -> INKANNWLJPN	D 0 -> LKDKSVVO	D 0 -> REYPKENA
D 1 -> HMJZMMVKIOM	D 1 -> KJCJRUUN	D 1 -> QDXOJDMZ
D 2 -> GLIYLLUJHNL	D 2 -> JIBIQTTM	D 2 -> PCWNICLY
D 3 -> FKHXKKTIGMK	D 3 -> IHAHPSSL	D 3 -> OBVMHBKX
D 4 -> EJGWJJSHFLJ	D 4 -> HGZGORRK	D 4 -> NAULGAJW
D 5 -> DIFVIIRGEKI	D 5 -> GFYFNQQJ	D 5 -> MZTKFZIV
D 6 -> CHEUHHQFDJH	D 6 -> FEXEMPPI	D 6 -> LYSJEYHU
D 7 -> BGDGTGGPECIG	D 7 -> EDWDLOOH	D 7 -> KXRIDXGT
D 8 -> AFCSFFODBHF	D 8 -> DCVCKNNG	D 8 -> JWQHCWFS
D 9 -> ZEBREENCAGE	D 9 -> CBUBJMMF	D 9 -> IVPGBVER
D 10 -> YDAQDDMBZFD	D 10 -> BATAILLE	D 10 -> HUOFAUDQ
D 11 -> XCZPCCLAYEC	D 11 -> AZSZHKKD	D 11 -> GTNEZTCP
D 12 -> WBYOBBKZXDB	D 12 -> ZYRYGJJC	D 12 -> FSMDYSBO
D 13 -> VAXNAAJYWCA	D 13 -> YXQXFIIB	D 13 -> ERLCXRAN
D 14 -> UZWMZZIXVBZ	D 14 -> XWPWEHHA	D 14 -> DQKBWQZM
D 15 -> TYVLYYHWUAY	D 15 -> WVOVDGGZ	D 15 -> CPJAVPYL
D 16 -> SXUKXXGVTZX	D 16 -> VUNUCFFY	D 16 -> BOIZUOXK
D 17 -> RWTJWWFUSYW	D 17 -> UTMTBEEEX	D 17 -> ANHYTNWJ
D 18 -> QVSIVVETRXV	D 18 -> TSLSADDW	D 18 -> ZMGXSMVI
D 19 -> PURHUUDSQWU	D 19 -> SRKRZCCV	D 19 -> YLFWRLUH
D 20 -> OTQGTTCPVPT	D 20 -> RQJQYBBU	D 20 -> XKEVQKTG
D 21 -> NSPFSSBQOUS	D 21 -> QPIPXAAAT	D 21 -> WJDUPJSF
D 22 -> MROERRAPNTR	D 22 -> POHOWZZS	D 22 -> VICTOIRE
D 23 -> LQNDQQZOMSQ	D 23 -> ONGNVYYR	D 23 -> UHBSNHQD
D 24 -> KPMCAPPYNLRP	D 24 -> NMFMUXXQ	D 24 -> TGARMGPC
D 25 -> JOLBOOXMKQO	D 25 -> MLELTWWP	D 25 -> SFZQLFOB

Nom du fichier : ex_3_cryptanalyse_Cesar_force brute.py

Correction Exercice 4

a) Programme Python 3 (zone de script)



```
texte_clair='ZEBREENCAGE'
texte_chiffre='INKANNWLJPN'

lettre_clair=texte_clair[0]
nb_clair=ord(lettre_clair)-65
lettre_chiffre=texte_chiffre[0]
nb_chiffre=ord(lettre_chiffre)-65
K=(nb_chiffre-nb_clair)%26
print("La clef de chiffrement/déchiffrement vaut K=",K)
```

b) Sortie Python (console)

```
texte_clair='PHYSIQUE'
texte_chiffre='XPGAQYCM'
Out : La clef de chiffrement/déchiffrement vaut K= 8
```

```
texte_clair='SOLEIL'
texte_chiffre='FBYRVY'
Out : La clef de chiffrement/déchiffrement vaut K= 13
```

```
texte_clair='ORDINATEURQUANTIQUE'
texte_chiffre='DGSXCPITJGFJPCIXFJT'
Out : La clef de chiffrement/déchiffrement vaut K= 15
```

Nom du fichier : ex_4_cryptanalyse_Cesar_clef.py

Correction exercice 5

a) Utilisation de la fonction Python 3 : **AFFINE_CHIFFRE(texte_clair,a,b)** 


	texte_clair	$(a;b)$	nb_clair	nb_chiffre	lettre_chiffre	texte_chiffre
Entrée	'ICI '	$(5;3)$				
Initialisation	'ICI '	$(5;3)$				[]
Boucle	'ICI '	$(5;3)$	8	17	R	[R]
	'ICI '	$(5;3)$	2	13	N	[R,N]
	'ICI '	$(5;3)$	8	17	R	[R,N,R]
Conversion	'ICI '	$(5;3)$				'RNR'
Sortie						'RNR'

b) Programme en Python 3 (zone de script) 

```
def AFFINE_CHIFFRE(texte_clair,a,b):

    texte_chiffre= [ ]
    for lettre in texte_clair:                # Boucle
        nb_clair=ord(lettre)-65                # Bijection b
        nb_chiffre=(a*nb_clair+b)%26           # Fonction de chiffrement
        lettre_chiffre=chr(nb_chiffre+65)      # Bijection b^-1
        texte_chiffre.append(lettre_chiffre)
    texte_chiffre="".join(texte_chiffre)      # Conversion liste en chaîne de caractère

    return texte_chiffre
```

c) Programme Python 3 (zone de script) 

```
def AFFINE_DECHIFFRE(texte_chiffre,a,b):

    for c in range(1,26):
        if (a*c)%26==1:
            inverse=c                        # c est l'inverse de a mod 26

    texte_clair= [ ]
    for lettre in texte_chiffre:            # Boucle
        nb_chiffre=ord(lettre)-65            # Bijection b
        nb_clair=inverse*(nb_chiffre-b)%26    # Fonction de déchiffrement
        lettre_clair=chr(nb_clair+65)         # Bijection b^-1
        texte_clair.append(lettre_clair)
    texte_clair="".join(texte_clair)        # Conversion liste en chaîne de caractère

    return texte_clair
```

Sortie Python (console)

In : AFFINE_CHIFFRE("BATAILLE",9,2) Out : 'LCRCWXXM'	In : AFFINE_DECHIFFRE('LCRCWXXM',9,2) Out : 'BATAILLE'
---	---

Nom du fichier : ex_5_chiffrement_affine.py

Correction Exercice 6

Considérons un chiffrement affine de clef $K = (4; 2)$

a) Complétons et observons le tableau de chiffrement ci-dessous :

Alphabet clair 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	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
<div>↓ $C_{(4;2)}$</div>																											
Alphabet chiffré y	2	6	10	14	18	22	0	4	8	12	16	20	24	2	6	10	14	18	22	0	4	8	12	16	20	24	
	C	G	K	O	S	W	A	E	I	M	Q	U	Y	C	G	K	O	S	W	A	E	I	M	Q	U	Y	

Fonction de chiffrement : $C_{(4;2)}(x) = 4x + 2 \mod 26$

b) Chiffrer le texte : « TELEGRAMME » avec le clef $K = (4; 2)$.

'ASUSASCYYYS'

c) Expliquer pourquoi la clef $K = (4; 2)$ ne définit pas une fonction de chiffrement valide.

La fonction $C_{(4;2)}$ n'est pas bijective de \mathbb{Z}_{26} vers \mathbb{Z}_{26} .

Par exemple : $T \xrightarrow{C} A$ et $G \xrightarrow{C} A$

d) Expliquer pourquoi on ne peut pas obtenir la fonction de déchiffrement $D_{(4;2)}$ associé à la clef $K = (4; 2)$.

Pour obtenir la fonction de déchiffrement $D_{(a;b)}(y) = a^{-1} \cdot (y - b) \mod 26$

on doit calculer a^{-1} qui est l'inverse de $a = 4$ modulo 26.

Cependant il n'existe pas de nombre a^{-1} tel que $a^{-1} \cdot 4 \equiv 1 \mod 26$.

e) Expliquer pourquoi il est difficile de déchiffrer le texte : « YCAESYCAIOESW » avec le clef $K = (4; 2)$.

$C_{(4;2)}$ n'étant pas une fonction bijective de \mathbb{Z}_{26} vers \mathbb{Z}_{26} il n'existe pas de fonction réciproque $D_{(4;2)}$.

En regardant le tableau de chiffrement « à l'envers », le texte « YCAESYCAIOESW » peut correspondre à :

" ZNGHEZNGIQUES " ou "MATHEMATIQUES" ouetc.

Correction Exercice 7

a) La clef $K = (3; 21)$ définit-elle une fonction de chiffrement valide ?

$a = 3$ admet un inverse modulo 26 qui est $a^{-1} = 9$: $3 \cdot 9 \equiv 1 \text{ mod } 26$

$b = 21$ peut-être quelconque.

La clef est valide et définit deux fonctions bijectives et réciproques l'une de l'autre :

- *Fonction de chiffrement* : $C_{(3;21)}(x) = 3x + 21 \text{ mod } 26$
- *Fonction de déchiffrement* : $D_{(3;21)}(y) = 9 \cdot (y - 21) \text{ mod } 26$ (l'inverse de 3 est 9 modulo 26)

b) Déchiffrons le texte « XLCHTC » :

texte chiffré	X	L	C	H	T	C
nb chiffré : y	23	11	2	7	19	2
nb clair : $D_{(3;21)}(y) = 9 \cdot (y - 21) \text{ mod } 26$	18	14	11	4	8	11
texte clair	S	O	L	E	I	L

c) Chiffrons le texte « PAIX » :

texte clair	P	A	I	X
nb clair : x	15	0	8	23
nb chiffré : $C_{(3;21)}(x) = 3x + 21 \text{ mod } 26$	14	21	19	12
texte chiffré	O	V	T	M

Correction Exercice 8

a) Programme Python 3 (zone de script)



```
texte_chiffre = 'OIDABZVQVTEMDTGDZNUV'

A = [1,3,5,7,9,11,15,17,19,21,23,25]
B = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25]

for a in A:
    for b in B:
        print("(" ,a,";" ,b,) -> ",AFFINE_DECHIFFRE(texte_chiffre,a,b))
```

b) Sortie Python (console)

```
Out :

( 1 ; 0 ) -> OIDABZVQVTEMDTGDZNUV
( 1 ; 1 ) -> NHCZAYUPUSDLCSEFCYMTU
( 1 ; 2 ) -> MGBYZXTOTRCKBREBXLST
( 1 ; 3 ) -> LFAXYWSNSQBJAQDAWKRS
( 1 ; 4 ) -> KEZWXVRMRPAIZPCZVJQR
....

( 11 ; 0 ) -> GWFATHJSJXYUFXKFHNQJ
( 11 ; 1 ) -> NDMHAQQZQEFCBMRMOUXQ
( 11 ; 2 ) -> UKTOHVXGXLMTLYTVBEX
( 11 ; 3 ) -> BRAVOCENESTPASFACILE
( 11 ; 4 ) -> IYHCVJLULZAWHZMHJPSL
( 11 ; 5 ) -> PFOJCQSBGSHDOGTOQWZS
( 11 ; 6 ) -> WMVQJXZIZNOKVNAVXDGZ
....

( 25 ; 21 ) -> HNSVUWAFACRJSCPSWIBA
( 25 ; 22 ) -> IOTWVXBGBDSKTDQTXJCB
( 25 ; 23 ) -> JPUXWYCHCETLUERUYKDC
( 25 ; 24 ) -> KQVYXZDIDFUMVFSVZLED
( 25 ; 25 ) -> LRWZYAEJEGVNWGTWAMFE
```

Nom du fichier : ex_8_cryptanalyse_affine_force brute.py

Correction Exercice 9

a)

$$\begin{array}{l} L \xrightarrow{b} 11 \xrightarrow{C} a \cdot 11 + b \bmod 26 \equiv 2 \xrightarrow{b^{-1}} C \\ E \xrightarrow{b} 4 \xrightarrow{C} a \cdot 4 + b \bmod 26 \equiv 9 \xrightarrow{b^{-1}} J \end{array} \Rightarrow \begin{cases} 2 \equiv 11a + b \bmod 26 \\ 9 \equiv 4a + b \bmod 26 \end{cases} \quad a \text{ et } b \text{ sont les inconnues.}$$

$$\begin{cases} 2 \equiv 11a + b \bmod 26 \\ 9 \equiv 4a + b \bmod 26 \end{cases} \quad | \quad E_2 - E_1$$

$$\Leftrightarrow -7 \equiv 7a \bmod 26$$

$$\Leftrightarrow 7a \equiv -7 \bmod 26 \quad | \quad 19 \equiv -7 \bmod 26$$

$$\Leftrightarrow 7a \equiv 19 \bmod 26 \quad | \quad \cdot 15 \text{ qui est l'inverse de 7 modulo 26}$$

$$\Leftrightarrow 15 \cdot 7a \equiv 15 \cdot 19 \bmod 26$$

$$\Leftrightarrow 105a \equiv 285 \bmod 26$$

$$\Leftrightarrow a \equiv 25 \bmod 26 \quad | \quad \text{car } 105 \equiv 1 \bmod 26 \text{ et } 285 \equiv 25 \bmod 26$$

On choisit $\underline{a = 25}$ car il possède un inverse $\underline{a^{-1} = 25}$ modulo 26

On substitue $a = 25$ dans l'équation E_1 :

$$2 \equiv 11 \cdot 25 + b \bmod 26 \Leftrightarrow b \equiv 2 - 275 \bmod 26 \Leftrightarrow b \equiv -273 \bmod 26 \Leftrightarrow b \equiv 13 \bmod 26$$

donc $\underline{b = 13}$

Vérification :

$$\begin{array}{l} L \xrightarrow{b} 11 \xrightarrow{C} 25 \cdot 11 + 13 \bmod 26 \equiv 2 \xrightarrow{b^{-1}} C \\ E \xrightarrow{b} 4 \xrightarrow{C} 25 \cdot 4 + 13 \bmod 26 \equiv 9 \xrightarrow{b^{-1}} J \end{array}$$

Finalement : la clef de chiffrement est $\underline{\underline{K = (25; 13)}}$

Sortie Python (console)



```
In : AFFINE_DECHIFFRE('CJLZBBNAKJBBAUBFCFUNFWJJVUYWJU',25,13)
Out : 'LECOMMANDEMENTMILITAIREESTPRET'
```

b) Fonction de chiffrement : $C_{(25;13)}(x) = 25x + 13 \bmod 26$

Fonction de déchiffrement : $D_{(25;13)}(y) = 25(y - 13) \bmod 26$ (l'inverse de 25 est 25 modulo 26)

Correction Exercice 10

a) Utilisation de la fonction Python 3 : **SUB_CHIFFRE(texte_clair,clef)**



Clef de chiffrement : CH2='QWERTZUIOPASDFGHJKLYXCVBNM'

	texte_clair	lettre	CH1[j]	CH2[j]	texte_chiffre
Entrée	' ZEBRE '				
Initialisation	' ZEBRE '				' '
Boucle	' ZEBRE '	Z	CH1[25] = Z	CH2[25] = M	' M '
	' ZEBRE '	E	CH1[4] = E	CH2[4] = T	' MT '
	' ZEBRE '	B	CH1[1] = B	CH2[1] = W	' MTW '
	' ZEBRE '	R	CH1[17] = R	CH2[17] = K	' MTWK '
	' ZEBRE '	E	CH1[4] = E	CH2[4] = T	' MTWKT '
Sortie					' MTWKT '

b) Programme Python (zone de script)



```
def SUB_CHIFFRE(texte_clair,clef):
    CH1='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    CH2=clef
    texte_chiffre= ' ' # variable : chaîne de caractère
    for lettre in texte_clair :
        for j in range(26):
            if lettre == CH1[ j ]:
                texte_chiffre= texte_chiffre + CH2[ j ]
    return texte_chiffre
```

c) Programme Python 3 (zone de script)



```
def SUB_DECHIFFRE(texte_chiffre,clef):
    CH1='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    CH2=clef
    texte_clair=' ' # variable : chaine de caractere
    for lettre in texte_chiffre:
        for j in range(26):
            if lettre == CH2[ j ]:
                texte_clair= texte_clair + CH1[ j ]
    return texte_clair
```

Sortie Python 3 (console)

```
In : SUB_CHIFFRE('CECIESTUNESSAIDEPHRASEVRAIE','QWERTZUIOPASDFGHJKLYXCVBNM')
Out : 'ETEOTLYXFLLQORTHIKQLTCKQOT'
In : SUB_DECHIFFRE('ETEOTLYXFLLQORTHIKQLTCKQOT','QWERTZUIOPASDFGHJKLYXCVBNM')
Out : 'CECIESTUNESSAIDEPHRASEVRAIE'
```

Nom du fichier : ex_10_chiffrement_substitution.py

Correction Exercice 11

a) 1) Clef : permutation π_1

Alphabet en clair 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	
$\Downarrow \pi_1$																											
Alphabet chiffré y	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	

texte clair	B	A	T	A	I	L	L	E
texte chiffré	E	D	W	D	L	O	O	H

2) Quel est la particularité de chiffrement ?

C'est un chiffrement par décalage (de César) de clef $k = 3$

b) Clef : permutation π_2

Alphabet en clair 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	
<div>↓ π_2 </div>																											

Déchiffrons le texte chiffré : 'NSDJSZO' à l'aide la permutation π_2^{-1} .

Il suffit de lire le tableau de la permutation π_2 (chiffrement) du bas vers le haut pour obtenir la permutation π_2^{-1} (déchiffrement).

texte chiffré	N	S	D	J	S	Z	O
texte clair	B	O	N	J	O	U	R

Correction Exercice 13

DONNEES

```
texte_chiffre_01 = 'RJISQQSMRIPSJKLSINOGONQBJQINJSQBOJSLIKQGNIVRISPBXAIRNIK'  
texte_chiffre_02 = 'LIKIXGLOKBDKKOJQPSJKLIKORKKOLPILBHIRTLPKRIGQNRIPKGOJQK'
```



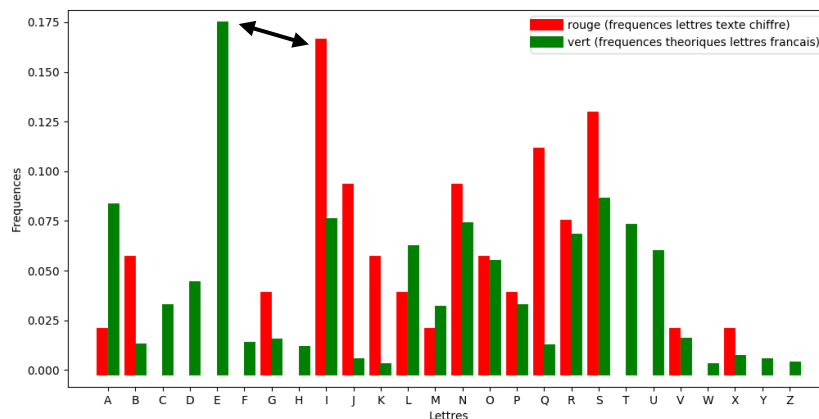
TRAITEMENT_Sortie Python (console)



```
In : DIAGRAMME_FREQ('RJISQQSMRIPSJKLSINOGONQBJQINJSQBOJSLIKQGNIVRISPBXAIRNIK')
```

En français, les lettres les plus rencontrées sont dans l'ordre :
E_S_A_I_N_T_R_L_U_O_D_C_P_M_V_G_F_B_Q_H_X_J_Y_Z_K_W

Out :



Hypothèse : I -> e

```
In : SUB_LETTRE('RJISQQSMRIPSJKLSINOGONQBJQINJSQBOJSLIKQGNIVRISPBXAIRNIK','I',"e")  
Out : 'RJeSQQSMRePSJKLSeNOGONQBJQeNJSQBOJSLeKQGNeVReSPBXAeRNeK'
```

```
In : SUB_LETTRE('LIKIXGLOKBDKKOJQPSJKLIKORKKOLPILBHIRTLPKRIGQNRIPKGOJQK','I',"e")  
Out : 'LeeXGLOKBDKKOJQPJaKLeKORKKOLPeLBHHeRTLePRKeGQNRePeKGOJQK'
```

Hypothèse : Dans les textes on suppose que les mots « **ATTACHE » et « **EXPLOSIFS** » ont été utilisés.**

Hypothèse : S -> a

```
In : SUB_LETTRE('RJeSQQSMRePSJKLSeNOGONQBJQeNJSQBOJSLeKQGNeVReSPBXAeRNeK',"S","a")  
Out : 'RJeaQQaMRePaJKLaeNOGONQBJQeNJaQBOJaLeKQGNeVReaPBXAeRNeK'
```

```
In : SUB_LETTRE('LeKeXGLOKBDKKOJQPSJKLeKORKKOLPeLBHHeRTLePRKeGQNRePeKGOJQK',"S","a")  
Out : 'LeeXGLOKBDKKOJQPJaKLeKORKKOLPeLBHHeRTLePRKeGQNRePeKGOJQK'
```

Hypothèse : Q -> t

```
In : SUB_LETTRE('RJeSQQaMRePaJKLaeNOGONQBJQeNJaQBOJaLeKQGNeVReaPBXAeRNeK',"Q","t")  
Out : 'RJeattaMRePaJKLaeNOGONtBJteNJatBOJaLeKtGNeVReaPBXAeRNeK'
```

```
In : SUB_LETTRE('LeKeXGLOKBDKKOJQPJaKLeKORKKOLPeLBHHeRTLePRKeGQNRePeKGOJQK',"Q","t")  
Out : 'LeeXGLOKBDKKOJtPaJKLeKORKKOLPeLBHHeRTLePRKeGtNRePeKGOJtK'
```

Hypothèse : M -> q

```
In : SUB_LETTRE('RJeattaMRePaJKLaeNOGONtBJteNJatBOJaLeKtGNeVReaPBXAeRNeK',"M","q")  
Out : 'RJeattaqRePaJKLaeNOGONtBJteNJatBOJaLeKtGNeVReaPBXAeRNeK'
```

```
In : SUB_LETTRE('LeKeXGLOKBDKKOJtPaJKLeKORKKOLPeLBHHeRTLePRKeGtNRePeKGOJtK',"M","q")  
Out : 'LeeXGLOKBDKKOJtPaJKLeKORKKOLPeLBHHeRTLePRKeGtNRePeKGOJtK'
```

Hypothèse : R -> u

In : SUB_LETTRE('RJeattaqRePaJKLaeNOGONtBJteNJaTBOJaLeKtGNeVReaPBXAeRNeK','R','u')
Out : 'uJe**attaq**uePaJKLaeNOGONtBJteNJaTBOJaLeKtGNeVueaPBXAeuNeK'

In : SUB_LETTRE('LeKeXGLOKBDKKOJtPaJKLeKORKKOLPeLBHHeRTLePRKeGtNRePeKGOJtK','R','u')
Out : 'LeKe**XGLOKBDK**KOJtPaJKLeKOuKKOLPeLBHHeuTLePuKeGtNuePeKGOJtK'

Hypothèse : G -> p

In : SUB_LETTRE('uJeattaquePaJKLaeNOGONtBJteNJaTBOJaLeKtGNeVueaPBXAeuNeK','G','p')
Out : 'uJe**attaq**uePaJKLaeNOpONtBJteNJaTBOJaLeKtpNeVueaPBXAeuNeK'

In : SUB_LETTRE('LeKeXGLOKBDKKOJtPaJKLeKOuKKOLPeLBHHeuTLePuKeGtNuePeKGOJtK','G','p')
Out : 'LeKe**XpLOKBDK**KOJtPaJKLeKOuKKOLPeLBHHeuTLePuKeptNuePeKpOJtK'

Hypothèse : K -> s

In : SUB_LETTRE('uJeattaquePaJKLaeNOpONtBJteNJaTBOJaLeKtpNeVueaPBXAeuNeK','K','s')
Out : 'uJe**attaq**uePaJsLaeNOpONtBJteNJaTBOJaLestpNeVueaPBXAeuNes'

In : SUB_LETTRE('LeKeXpLOKBDKKOJtPaJKLeKOuKKOLPeLBHHeuTLePuKeptNuePeKpOJtK','K','s')
Out : 'Les**eXpLOsBDs**OJtPaJsLesOussOLPeLBHHeuTLePuseptNuePespOJts'

Hypothèse : B -> i

In : SUB_LETTRE('uJeattaquePaJsLaeNOpONtBJteNJaTBOJaLestpNeVueaPBXAeuNes','B','i')
Out : 'uJe**attaq**uePaJsLaeNOpONtiJteNJaTiOJaLestpNeVueaPiXAeuNes'

In : SUB_LETTRE('LeseXpLOsBDssOJtPaJsLesOussOLPeLBHHeuTLePuseptNuePespOJts','B','i')
Out : 'Les**eXpLOsiDs**OJtPaJsLesOussOLPeLiHHeuTLePuseptNuePespOJts'

Hypothèse : D -> f

In : SUB_LETTRE('uJeattaquePaJsLaeNOpONtiJteNJaTiOJaLestpNeVueaPiXAeuNes','D','f')
Out : '**uJeattaq**uePaJsLaeNOpONtiJteNJaTiOJaLestpNeVueaPiXAeuNes'

In : SUB_LETTRE('LeseXpLOsiDssOJtPaJsLesOussOLPeLiHHeuTLePuseptNuePespOJts','D','f')
Out : 'Les**eXpLOsifs**sOJtPaJsLesOussOLPeLiHHeuTLePuseptNuePespOJts'

Hypothèse : J -> n

In : SUB_LETTRE('uJeattaquePaJsLaeNOpONtiJteNJaTiOJaLestpNeVueaPiXAeuNes','J','n')
Out : 'uneattaque**Pans**LaeNOpONt**inteNnatiOnaL**estpNeVueaPiXAeuNes'

In : SUB_LETTRE('LeseXpLOsifssOJtPaJsLesOussOLPeLiHHeuTLePuseptNuePespOJts','J','n')
Out : 'LeseXpLOsifssOnt**Pans**LesOussOLPeLiHHeuTLePuseptNuePespOnts'

Hypothèse : N -> r

In : SUB_LETTRE('uneattaquePansLaeNOpONt**inteNnatiOnaL**estpNeVueaPiXAeuNes','N','r')
Out : 'uneattaque**Pans**LaerOpOrt**internatiOnaL**estpreVueaPiXAeures'

In : SUB_LETTRE('LeseXpLOsifssOntPansLesOussOLPeLiHHeuTLePuseptNuePespOnts','N','r')
Out : 'LeseXpLOsifssOntPansLesOussOLPeLiHHeuTLePuseptNuePespOnts'

Hypothèse : P -> d

In : SUB_LETTRE('uneattaquePansLaerOpOrtinternatiOnaLestpreVueaPiXAeures','P','d')
Out : 'uneattaquedansLaerOpOrtinternatiOnaLestpreVueadiX**Aeures**'

In : SUB_LETTRE('LeseXpLOsifssOntPansLesOussOLPeLiHHeuTLePuseptNuePespOnts','P','d')
Out : 'LeseXpLOsifssOntdansLesOussOLdeLiHHeuTleduseptNuePespOnts'

Hypothèse : A -> h

In : SUB_LETTRE('uneattaquedansLaerOpOrtinternatiOnaLestpreVueadiXAeures','A','h')
Out : 'uneattaquedansLaerOpOrtinternatiOnaLestpreVueadiXheures'

In : SUB_LETTRE('LeseXpLOsifssOntdansLesOussOLdeLiHHeuTleduseptNuePespOnts','A','h')
Out : 'LeseXpLOsifssOntdansLesOussOLdeLiHHeuTleduseptNuePespOnts'

REPONSES



texte_clair_01 ='UNEATTAQUEDANSLAEROPORTINTERNATIONALESTPREVUEADIXHEURES'

texte_clair_02 ='LESEXPLOSIFSSONTDANSLESOUSSOLDELIMMEUBLEDUSEPTRUEDESPONTS'

HYPOTHESE CLEF

Alphabet en clair 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
$\Downarrow \pi$																										
Alphabet chiffré y	S	T	*	P	I	D	*	A	B	*	*	L	H	J	O	G	M	N	K	Q	R	V	*	X	*	*

REMARQUE _CLEF COMPLETE

clef="STUPEIDEABCFLHJOGMNKQVRWXYZ"

Correction exercice 14

a) Utilisation de la fonction Python 3 : **VIGENERE_CHIFFRE(texte_clair,mot_clef)**



	texte_clair	mot_clef	nb_clef	clef	m	nb_clair	nb_chiffre	lettre_chiffre	i	texte_chiffre
Entrée	' MATH '	' OK '								
Initialisation	' MATH '	' OK '		[]						
Boucle 1	' MATH '	' OK '	14	[14]						
	' MATH '	' OK '	10	[14,10]						
Initialisation	' MATH '	' OK '	10	[14,10]	2				0	[]
Boucle 2	' MATH '	' OK '	10	[14,10]	2	12	0	A	1	[A]
	' MATH '	' OK '	10	[14,10]	2	0	10	K	0	[A,K]
	' MATH '	' OK '	10	[14,10]	2	19	7	H	1	[A,K,H]
	' MATH '	' OK '	10	[14,10]	2	7	17	R	0	[A,K,H,R]
Conversion										' AKHR '
Sortie										' AKHR '

b) Programme Python (zone de script)

```
def VIGENERE_CHIFFRE(texte_clair,mot_clef): # mot_clef est une chaine de caractère

    clef=[ ]
    for lettre in mot_clef:                # Boucle sur mot_clef
        nb_clef=ord(lettre)-65             # Bijection b
        clef.append(nb_clef)              # clef est une liste de nombre [k1,k2,...,km]

    texte_chiffre= [ ]
    m=len(clef)                            # Calcule la longueur de la clef
    i=0                                    # Rang dans le bloc
    for lettre in texte_clair:             # Boucle sur texte clair
        nb_clair=ord(lettre)-65            # Bijection b
        nb_chiffre=(nb_clair+clef[i])%26   # Fonction de chiffrement
        lettre_chiffre=chr(nb_chiffre+65)  # Bijection b^-1
        i=(i+1)%m                         # On passe au rang suivant
        texte_chiffre.append(lettre_chiffre)
    texte_chiffre=" ".join(texte_chiffre)  # Conversion liste en chaine de caractère

    return texte_chiffre
```

c) Programme Python 3 (zone de script)

```
def VIGENERE_DECHIFFRE(texte_chiffre,mot_clef): # mot_clef est une chaine de caractère

    clef=[ ]
    for lettre in mot_clef:                # Boucle sur mot_clef
        nb_clef=ord(lettre)-65             # Bijection b
        clef.append(nb_clef)              # clef est une liste de nombre [k1,k2,...,km]

    texte_clair= [ ]
    m=len(clef)                            # Calcule la longueur de la clef
    i=0                                    # Rang dans le bloc
    for lettre in texte_chiffre:           # Boucle sur texte chiffre
        nb_chiffre=ord(lettre)-65          # Bijection b
        nb_clair=( nb_chiffre - clef[i] )%26 # Fonction de dechiffrement
        lettre_clair=chr(nb_clair+65)      # Bijection b^-1
        i=(i+1)%m                         # On passe au rang suivant
        texte_clair.append(lettre_clair)
    texte_clair=" ".join(texte_clair)      # Conversion liste en chaine de caractère

    return texte_clair
```

c) Sortie Python (console) :

In : VIGENERE_CHIFFRE('BATAILLE','AIR') Out : 'BIKAQCLM'	In : VIGENERE_DECHIFFRE('BIKAQCLM','AIR') Out : 'BATAILLE'
---	---

Nom du fichier : ex_14_chiffrement_Vigenere.py

Correction Exercice 15

- a) Chiffrons le texte clair : ' BATAILLE ' à l'aide du *chiffrement de Vigenère* avec le mot clef ' AIR '.

Remarque : c'est le travail d'ALICE.

texte clair	B	A	T	A	I	L	L	E
nb clair x_i	1	0	19	0	8	11	11	4
mot clef	A	I	R	A	I	R	A	I
décalage k_i	0	8	17	0	8	17	0	8
nb chiffré y_i $\equiv x_i + k_i \bmod 26$	1	8	10	0	16	2	11	12
texte chiffré	B	I	K	A	Q	C	L	M

- b) Déchiffrons le texte chiffré : ' SUFXUHHH ' à l'aide du *chiffrement de Vigenère* avec le mot clef ' STOP '.

Remarque : c'est le travail de BOB.

texte chiffré	S	U	F	X	U	H	H	H
nb chiffré y_i	18	20	5	23	20	7	7	7
mot clef	S	T	O	P	S	T	O	P
décalage k_i	18	19	14	15	18	19	14	15
nb clair x_i $\equiv y_i - k_i \bmod 26$	0	1	17	8	2	14	19	18
texte clair	A	B	R	I	C	O	T	S

Correction Exercice 16 Aucune

Correction Exercice 17

a) Programme Python 3 (zone de script)



```
texte_chiffre="ESPIUGDKGSVXUSFVGH"
mot_prob="MESSAGE"

m=len(texte_chiffre)
n=len(mot_prob)
p=m-n

for i in range(p+1):
    sequence=texte_chiffre[0+i:n+i]
    print("sequ_texte_ch_",sequence)
    print("mot_prob_____",mot_prob)

    mot_clef=""
    for j in range(n):
        nb=(ord(sequence[j])-ord(mot_prob[j]))%26
        lettre_clef=chr(nb+65)
        mot_clef=mot_clef+lettre_clef
    print("mot_clef_____",mot_clef)
    print("")
```

b) Sortie Python 3 (console)

```
texte_chiffre_01 = "ESPIUGDKGSVXUSFVGH"
mot_prob_01 = "MESSAGE"
```

Out :

.....

```
sequ_texte_ch_ PIUGDKG
mot_prob_____ MESSAGE
mot_clef_____ DECODEC
```

.....

```
In  : VIGENERE_DECHIFFRE(texte_chiffre_01,"CODE")
Out : 'CEMESSAGEESTSECRET'
```

```
texte_chiffre_02= "WOLVIIJNJV CORUEHDVXLVQBRJTKDYHNCVPIGRNGHZGNSZPXBATRQBRB"
mot_prob_02 = "ATTAQUE"
```

Out :

.....

```
sequ_texte_ch_ JTKDYHN
mot_prob_____ ATTAQUE
mot_clef_____ JARDINJ
```

.....

```
In  : VIGENERE_DECHIFFRE(texte_chiffre_02,"JARDIN")
Out : 'NOUSAVONSSUBIUNEVIOLENTEATTAQUECEMATINPERTESIMPORTANTES'
```

Nom du fichier : ex_17_cryptanalyse_Vigenere.py

[illegible]

Notes personnelles