Cryptographie élémentaire

1 Problématique et vocabulaire associé

1.1 Notion de codage

D'une manière générale, un codage est une méthode qui permet de passer d'une représentation d'une donnée à une autre. Il existe un certain nombre de codage universels avec parmi eux notamment :

- Le morse, inventé en 1832 pour la télégraphie, qui établit une correspondance entre chaque lettre de l'alphabet et chaque chiffre une combinaison de signaux parmi 2 possible, le trait et le point. A titre d'exemple, la lettre e est codée par un point, le a par un point suivi d'un trait.

  Bien que très ancien, ce code reste largement utilisé à l'heure actuelle.
- Le code ASCII (American Strandard Code for Information Interchange) associe à chaque caractère de l'alphabet latin et plus généralement chaque symbole, signe de ponctuation, un entier unique. Il est utilisé dans la quasi-totalité des ordinateurs personnels pour le stockage d'information.

  Il est souvent complété par des correspondances supplémentaires pour le codage de caractères supplémentaires, comme les lettres accentuées, mais il existe alors différentes normes (ISO, UTF8) qui peuvent parfois générer des problèmes d'encodage.

Dans le contexte de ce cours, nous travaillerons uniquement avec des données sous forme d'une liste de caractères (à priori de taille imposante). Il s'agit donc à partir d'une liste initiale d'en produire une nouvelle suivant un algorithme précis. Parmi les différents objectifs d'un codage, on peut notamment les objectifs suivants :

- **La compression de données :** Etant donné un message `L1`, on produit un message `L2` et l'algorithme doit être conçu pour que `L2` soit de longueur inférieure à `L1`. On parlera alors de
  - compression sans perte, si l'application `L1 -> L2` est injective, et si l'on dispose de l'algorithme implémentant la réciproque `L2 -> L1`.
  - compression avec perte, si l'application `L1 -> L2` n'est pas injective. Cet outil est fréquemment utilisé pour la transmission de données (son, image) lorsque l'on accepte que le message `L2` soit de qualité moins bonne que le message `L1` pour gagner considérablement en efficacité sur la compression.
- **La correction d'erreurs :** Lorsque l'on transmet un message `L1` à un interlocuteur, il peut arriver que le message se dégrade lors de la retransmission. On code dans cet optique `L1` par un message `L2`, à priori plus long, mais pour lequel il est plus facile de déterminer les erreurs de transmission s'il y en a. Ces algorithmes seront étudiés dans un cours ultérieur.
- **Le chiffrement (ou cryptage) :** Il s'agit cette fois de produire à partir d'un message `L1` un message `L2` de manière à ce que seul le destinataire de `L2` soit en mesure de retrouver le message original `L1`. C'est ce point là qui nous intéresse dans ce cours.

A noter que le terme cryptographie ou cryptage est un terme exclusivement français.

1.2 Problématique du chiffrement

Dans la plupart des cours sur le chiffrement, on considère deux personnes, Alice et Bob qui cherchent à communiquer de manière sécurisée. A noter que les prénoms sont presque systématiquement ces deux là dans tous les cours. Alice a donc un message `L1` à transmettre à Bob. Le souci est d'envoyer un message à Bob avec les deux objectifs suivants :

- le message doit être incompréhensible à toute personne qui intercepterai le message (Eve la plupart du temps).
- Bob doit être en mesure de retrouver le message initial `L1` à partir de `L2`.

A noter que pour communiquer de façon sécurisée, d'autre problèmes se posent comme s'assurer de la nature de l'expéditeur du message (ce qui nécessite des signatures numériques), ou masquer l'existence d'une communication (techniques de stéganographie).
Le chiffrement du message commence toujours par le choix d'une clé, qui sera fournie en argument du programme, et utilisée lors de l'enchiffrement du message. En effet, la plupart des algorithmes de codage sont supposés être connus par les personnes indiscrètes, et le secret doit donc reposer sur une donnée qui n'appartient pas au contenu de l'algorithme. La technique de chiffrement est dite :

- **symétrique** si la même clé est utilisée pour chiffrer et déchiffrer.

- **asymétrique** quand il utilise des clés différentes : une paire constituée d'une clé publique, servant au chiffrement et fournie par Bob pour tout le monde (y compris les indiscrets), et une clé privée connue de Bob uniquement. L'expéditeur encode son message en utilisant la clé publique de Bob, et une fois que Bob reçoit le message, il le décripte avec sa clé privée.

  Le principe fondamental de cette décomposition est qu'il doit être impossible (du moins en un temps raisonnable) de retrouver la clé privée à partir de la clé publique.

L'intérêt essentiel du cryptage asymétrique est qu'il supprime le souci de la transmission de la clé. Toutefois, ses temps de calcul sont souvent nettement plus longs. L'algorithme le plus connu de chiffrement asymétrique est le chiffrement RSA (du nom de ses trois inventeurs Rivest, Shamir et Adleman) qui utilise des propriétés arithmétique et notamment la diffculté de la factorisation d'un entier n en nombres premier lorsque n est très grand.

1.3 Casser un algorithme de chiffrement

Comme précisé précédemment, le principe des algorithmes de chiffrement est supposé connu de tous. La seule inconnue réside donc dans le choix de la clé utilisée pour le décodage du message. Casser un algorithme de chiffrement, c'est donner une méthode pour retrouver la clé, et donc le texte original, à partir du message chiffré. La robustesse d'un algorithme de chiffrement réside donc dans la difficulté de ce travail.
De nombreux algorithmes de chiffrement ont été produits, puis cassés, notamment grâce à la croissance exponentielle de la puissance de calcul des ordinateurs. Si un jour tous les algorithmes sont cassés sans que d'autre ne soient proposés pour prendre la relève, l'informatique s'expose à de nombreux problèmes de sécurité. Fort heureusement, il y a encore de la marge.

2 Codage de César et analyse fréquentielle

2.1 Principe du chiffrement

Le chiffrement par décalage, aussi connu comme le chiffre de César est une méthode de chiffrement très simple, qui doit son nom au fait que Jules César l'aurait utilisé pour certaines de ses transmissions secrètes. Dans le chiffrement de César, la clé consiste simplement en un entier appelé décalage. Selon une biographie attribuée au romain Suétone, César utilisait le chiffre 3 et remplaçait donc chaque lettre par la 3-ième qui la suivait dans l'ordre alphabétique. Ainsi, le A devenait un D, le B un E, et ainsi de suite jusqu'à W -> Z, X -> A, Y -> B, Z -> C. Le déchiffrement se faisait donc de la même manière, en décalant les lettres de trois rangs vers la gauche.

Question 1

Le code ASCII est un code universel qui attribue à n'importe quel caractère d'un clavier un entier, et ce de manière bijective. Le a minuscule (resp. A majuscule) a pour code 97 (resp. 65), le b minuscule 98 (resp. 66 pour B) et ainsi de suite jusqu'à z de code 122 (resp. 90 pour Z). En python, on accède à cette valeur avec la fonction `ord` tandis que sa fonction réciproque est donnée par `chr`.

- Ecrire trois fonctions `est_min`, `est_maj` et `est_lettre` qui testent respectivement si le caractère donnée en argument est une minuscule, une majuscule ou une lettre.
- Ecrire une fonction `code` prenant en argument un caractère `c` supposé représenté une lettre et un entier p et renvoie la lettre obtenue par décalage de longueur p. A titre de vérification, `code 'x' 21` doit renvoyer la lettre `s`.

On remarquera que le nombre de clés est extrêmement faible, seulement 26 si l'on procède par décalage. C'est pourquoi on peut plus généralement définir le chiffrement par substitution monoalphabétique : chaque lettre est plus généralement remplacée par une autre, et la clé est fournie par la fonction de permutation des lettres de l'alphabet. Le nombre possible de clés devient alors égal à 26!, soit de l'ordre de 10^{26}, ce qui est alors bien plus conséquent.

2.2 Analyse fréquentielle (ou comment casser du César)

Si l'on dispose d'un texte codé par chiffrement de César, il est extrêmement facile de reconstituer le texte initial même si l'on ignore le décalage utilisé par la méthode dite de « la force brute ». Celle-ci consiste simplement à tester toutes les clés possibles vu leur faible nombre !
Si la permutation effectuée sur les lettres est aléatoire, il paraît beaucoup plus compliqué de tester toutes les possibilités de clés dans un temps raisonnable. Mais ce type de chiffrement ne résiste cependant à la technique de l'analyse fréquentielle qui se base sur le constat suivant :

- Si l'on compare deux textes suffisamment long écrits dans une même langue, la fréquence d'apparitions des lettres sont relativement voisines. Le tableau ci-dessous donne celles des textes en français et en anglais.

	A	B	C	D	E	F	G	H	I	J	K	L	M
Français	9,42	1,02	2,64	3,39	15,87	0,95	1,04	0,77	8,41	0,89	0,00	5,34	3,24
Anglais	8,08	1,67	3,18	3,99	12,56	2,17	1,80	5,27	7,24	0,14	0,63	4,04	2,60
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Français	7,15	5,14	2,86	1,06	6,46	7,90	7,26	6,24	2,15	0,00	0,30	0,24	0,32
Anglais	7,38	7,47	1,91	0,09	6,42	6,59	9,15	2,79	1,00	1,89	0,21	1,65	0,07

  Cette répartition n'est bien sûr qu'approximative, puisqu'elle dépend de nombreux paramètres, comme le niveau de langue du texte et celui d'écriture.
- Un algorithme de chiffrement par substitution monoalphabétique comme l'algorithme de César conserve la répartition des fréquences : si la lettre E est remplacée par un K par exemple, la lettre K sera, si le texte est suffisamment long, la lettre apparaissant le plus souvent dans le texte codé.

L'analyse fréquentielle consiste donc simplement à calculer les fréquences d'appartion de chaque lettre dans le texte codé, puis à remplacer les lettres les plus fréquentes du texte par les plus fréquentes dans le langage du texte original. Même si quelques lettres peuvent être malgré tout mal remplacées (cela dépend beaucoup de la longueur du texte), on retrouve le plus souvent le sens du message initial.
Cette technique apparaît dès le neuvième siècle dans le premier ouvrage de cryptanalyse (manuscript sur le déchiffrement des messages cryptographiques) écrit par le philosophe Al-kindi.

Question 2

Ecrire une fonction `frequences` qui prend en argument une chaîne de caractères et renvoie un tableau de 26 cases tel que la case 0 donne le pourcentage de lettre `a` et `A` dans les lettres du texte (minuscules et majuscules indifférenciées donc), la case 0 le pourcentage de `b` et `B` et ainsi de suite.
Attention, on ignore donc tout caractère qui n'est pas une lettre dans ce travail.

3 Codage de Vigenere

3.1 Principe du chiffrement

Le chiffrement de Vigenere fait son apparition dans un traité des chiffres paru en 1586 rédigé par un diplomate Blaise de Vigenere. Il résistera à l'analyse pendant prêt de trois siècles, avant d'être cassé dans un premier temps par le mathématicien britannique Charles Babbage (mais qui gardera secrète sa découverte), puis par le major prussien Freidrich Kasiski qui publie sa méthode en 1863. Depuis, il n'offre plus aucune sécurité. Il est fondé sur les principes suivants :

- Comme pour César, il s'agit d'un chiffrement par substitution : chaque lettre du texte est remplacé par une autre.
- Contrairement à César, une même lettre peut être remplacé par des lettres différentes. Cette méthode lui permet notamment de résister à l'analyse de fréquence qui casse les chiffrements de substitution monoalphabétique.
- L'algorithme est symétrique. Il utilise une clé donné par un mot ou plus généralement un texte (voire un passage entier d'une oeuvre littéraire).

Le principe de l'algorithme est basé en partie sur l'algorithme de César. Contrairement à ce dernier qui utilise un unique décalage pour chaque lettre du texte à coder, Vigenere utilise différents décalages pour chaque lettre. Le décalage d'une lettre α du texte original est déterminé en fonction d'une lettre β de la clé (choisie librement par l'encodeur) :

- si β est la lettre a, on ne modifie pas α,
- si β est la lettre b, on décale α de une unité : un a devient un b, un b devient un c, etc ... jusqu'à un z qui devient un a,
- si β est la lettre c, on décale α de deux unités : un a devient un c, un b devient un d, et ainsi de suite jusqu'à y qui devient a et z qui devient b,
- et ainsi de suite ...

Question 3

Ecrire une fonction `encode_lettre` qui prend en argument deux lettres `x` et `y` et qui réalise le codage de `x` par la lettre `y`. On pourra supposer que `y` est nécessairement une minuscule.

Le principe de l'algorithme est alors de coder chaque lettre du texte via la table suivante à l'aide d'une lettre de la clé, en parcourant les lettres de la clé de manière circulaire. Voici un exemple sur le texte "winter is coming" et la clé "tyrion" :

texte	w	i	n	t	e	r		i	s		c	o	m	i	n	g
cle	t	y	r	i	o	n		t	y		r	i	o	n	t	y
code	p	g	e	b	s	e		b	q		t	w	a	v	g	e

Le texte chiffré est donc "pgebse bq twavge". A noter qu'encore une fois, on ne modifie aucun caractère qui n'est pas une lettre.

Question 4

La syntaxe `C = ""` crée une chaîne de caractère vide. Pour rajouter le caractère `x` à la fin de la chaîne `C`, on peut alors comme pour les listes utiliser la syntaxe `C = C+'x'`.

- Ecrire une fonction prenant en argument une chaîne de caractères `texte` et une deuxième chaîne `cle` et qui renvoie la chaîne codée obtenue à partir de `texte` et de `cle` via l'algorithme de Vigenere.
- Ecrire de même la fonction de décodage prenant en argument le texte codé et la clé et renvoyant le message initial.

3.2 Cryptanalyse du chiffrement

Si l'on connaît la longueur k de la clé, le déchiffrement du message est extrêmement simple : il suffit d'appliquer un déchiffrage de César à chaque sous-message obtenu en ne conservant qu'une lettre sur k. Plus précisément, si les lettres du message codé forment la chaîne $L = [c_0, c_1, \dots, c_n]$, il suffit de considérer les sous-listes

$$L_{k,0} = [c_0, c_k, c_{2k}, \dots] \qquad L_{k,1} = [c_1, c_{k+1}, c_{2k+1}, \dots] \qquad \cdots \qquad L_{k,k-1} = [c_{k-1}, c_{2k-1}, c_{3k-1}, \dots]$$

Chaque liste de caractère a été encodé suivant un même décalage de César. On peut facilement trouver ce décalage, soit par une analyse force-brute, soit par analyse des fréquences. A partir de là, on reconstitue chaque lettre de la clé, et donc la clé elle-même permettant de décoder le message.

Question 5

Ecrire une fonction `extrait_lettres` qui prend en argument une chaîne de caractères `L` et en extrait la chaîne `L'` des lettres de `L` (où l'on a donc enlevé tous les autres caractères), puis une fonction `sous_texte` prenant en argument la chaîne `L'` et deux entiers $k \in \mathbb{N}^*$ et $r \in [\![0; k-1]\!]$ et renvoyant la chaîne $L_{k,r}$.

Casse l'algorithme de Vigenere revient donc essentiellement à trouver une technique pour retrouver la longueur de la clé utilisée pour le codage. On distingue pour cela la technique historique, et les améliorations modernes :

Test de Kasiski

L'idée de Kasiski (et de Babbage avant lui) est d'analyser les séquences de 3 lettres répétées dans un texte codé, et de se dire que ses répétitions ne sont pas le fruit du hasard. Il y a en effet de fortes chances pour qu'il s'agisse de la même séquence de trois lettres du texte original, codé par les mêmes séquences de 3 lettres du texte initial. Si l'on note alors d la distance entre les deux séquences dans le texte (exprimées en nombre de caractères) et m la longueur de la clé, il y a donc de fortes chances que m divise d.
La méthode consiste donc à déterminer le pgcd des distances entre des séquences répétées. Bien entendu, cela nécessite de ne pas prendre en compte certaines valeurs pour lesquels le hasard aurait « mal » fait les choses. Concrètement, il faut donc :

- repérer les séquences de trois lettres qui apparaissent assez fréquemment dans le texte codé ;
- calculer les distances entre ces répétitions ;
- chercher le diviseur commun le plus fréquent à ces valeurs, quitte à en éliminer quelques unes

La technique peut paraître un peu délicate à employer et rébarbative, mais il faut bien comprendre qu'à l'époque, c'est-à-dire sans ordinateur, il s'agissait d'une véritable révolution. En l'occurrence, cette technique ne nous intéresse pas aujourd'hui.

Indice de coïncidence

On appelle indice de coïncidence d'un texte la probabilité que deux lettres au hasard soient identiques. Si un texte est composé de n lettres dont n_a fois la lettre a, n_b fois la lettre b et ainsi de suite, l'indice de coïncidence I_c est donné par

$$I_c = \sum_{\alpha \in \{a,b,\dots,z\}} \frac{n_\alpha(n_\alpha - 1)}{n(n-1)} \qquad (\star)$$

Question 6

Ecrire une fonction `indice` prenant en argument une chaîne de caractères et renvoyant son indice de coïncidence.

Pour un texte parfaitement aléatoire, cette valeur vaut $1/26$, soit approximativement $I \approx 0,038$. Toutefois, pour un texte en français dont les lettres n'ont pas toutes la même probabilité d'appartition, on trouve une valeur proche de $I_f \approx 0,074$. Considérons maintenant un texte de longueur n, codé par l'algorithme de Vigenere à l'aide d'une clé de longueur k. L'idée consiste à calculer une valeur approchée de l'indice de coïncidence en fonction de k. Celle-ci se base sur l'analyse suivante. Soient deux lettres quelconques dans le texte codé. Alors,

- soit elles ont été codées avec la même lettre de la clé : la probabilité est d'environ $1/k$;
- soit elles ont été codées avec une lettre différente : la probabilité est d'environ $1 - 1/k$.

Dans le premier cas, la probabilité que les deux lettres soient égales est la même que celle qu'elles soient égale dans le texte original, soit I_f environ pour un texte en français suffisamment long. Dans le second, elle est proche de celle pour une distribution aléatoire des lettres, c'est-à-dire I. On en déduit donc la valeur approchée de l'indice de coïncidence :

$$I_c \approx \frac{1}{k} \cdot I_f + \left(1 - \frac{1}{k}\right) \cdot I \qquad (\star\star)$$

On peut ainsi inverser cette formule et en déduire une valeur (très) approchée de k donnée par

$$k \approx \frac{I_f - I}{I_c - I}$$

Question 7

Ecrire une fonction prenant en argument une chaîne de caractère codée à l'aide du codage de Vigenere et renvoyant une valeur approchée de la longueur de la clé k utilisée pour le codage à l'aide de la technique de l'indice de coïncidence.

Remarque : Cette dernière méthode est relativement peu précise. Il convient dans tous les cas de tester plusieurs longueurs de clé. Une vérification simple consiste à extraire pour différentes valeurs de k le texte $L_{k,0}$ avec les notations de la partie 3.2. Si celui-ci a un indice de coïncidence proche de I_f, c'est très certainement que k est la bonne longueur. On peut alors effectuer une analyse de fréquence de tous les $L_{k,r}$ pour $r \in [\![0; k-1]\!]$ afin de deviner une à une les lettres de la clé.

3.3 Un exemple pratique

Entrées/sorties de texte en Python

Les instructions suivantes permettent d'importer et d'exporter des fichiers de texte dans une session Python. Attention, je n'ai pas eu le temps de les tester au lycée, seulement sur mon installation personnelle.

- `from os import chdir` puis `chdir` pour indiquer dans quel répertoire de travail chercher le document à importer. Par défault, il cherche à la racine du compte. On écrira donc par exemple `chdir("Info/TP1/")` si le fichier se trouve dans la répertoire `Info/TP1/` depuis la racine de votre compte.
- `f=open('mon_texte.txt','r')` suivi de `C=f.read()` crée une chaîne de caractères `C` contenant les caractères du fichier `mon_texte` situé dans le répertoire spécifié comme précédemment. Un saut de ligne dans le .txt original est donné par le caractère '\n'.

  Remarque : cette commande plante chez moi dès lors que le fichier importé comporte des lettres accentuées ou spécifiques au français (cédille typiquement).
- Enfin, `s=open('nouveau_nom.txt','w')` suivi de `s.write(L')` et enfin `s.close()` permet de créer un nouveau fichier `nouveau_nom` de type txt contenant les caractères de la chaîne `L'` (obtenue par exemple après codage de `L`).

Question 8

A l'aide des méthodes établies dans cet énoncé, déterminer le nom de l'oeuvre et de l'auteur dont est issu le texte `texte_code_2022.txt`. Donner également la valeur de le clé.