TP : Le chiffrement de Vigenère

Option M1 SECU

Compte-rendu et sources à rendre au plus tard le **vendredi 10 octobre 2014**.

Résumé

La cryptographie a pour principal objectif de garantir la confidentialité, l'authenticité et l'intégrité des communications. Pour cela, depuis des siècles, de nombreux mécanismes ont été inventés. Nous ne remontons qu'à la fin du XVI^{e} siècle pour étudier le chiffrement de Vigenère, qui est une extension du chiffrement de César.

Cependant, bien que ce système semble *a priori* robuste, une première méthode d'attaque (ou cryptanalyse) a été proposée près de 3 siècles plus tard, indépendamment, par Babbage et Kasiski.

Le but de ce TP est de programmer ce chiffrement de Vigenère, ainsi que son déchiffrement (inversion du mécanisme à l'aide de la clé secrète). Nous programmerons également la méthode de décryptement (inversion du mécanisme sans la clé secrète) proposée par Babbage et Kasiski. Cette dernière s'applique dès que le message clair contient de la redondance (du texte par exemple), et est suffisamment long, puisque la cryptanalyse repose sur une analyse statistique de la redondance.

1 Modalités de ce TP en binôme

Ce TP sera évalué sur la base :

— des sources (veiller à la clarté des sources, et ne pas hésiter à commenter chaque étape) – à titre d'information, les deux programmes réunis font moins de **200 lignes** ;

— d'un rapport, d'au plus 2 pages détaillant vos réflexions et les problèmes rencontrés, ainsi que ce que fait effectivement votre programme (fonctionnalités disponibles, parfaitement opérationnel, certaines restrictions, . . .).

Ce compte-rendu doit être envoyé au plus tard le **10 octobre 2014**.

2 Présentation du TP

2.1 Le chiffrement de César

Des méthodes de chiffrement ont été recensées bien au-delà du début de notre ère. Mais la plupart sont restées bien rudimentaires jusqu'à la naissance de la cryptographie dite " moderne ". En effet, jusqu'au début de notre siècle, la confidentialité reposait sur l'utilisation d'algorithmes secrets, aussi bien pour le chiffrement que pour le déchiffrement.

Au premier siècle avant J.C., pour chiffrer ses communications pendant la guerre des Gaules, Jules César décalait chaque caractère de trois crans vers la droite (voir figure 1).

$$\begin{array}{ccc} A \longrightarrow D & B \longrightarrow E & C \longrightarrow F \\ \vdots \longrightarrow \vdots & \vdots \longrightarrow \vdots & \vdots \longrightarrow \vdots \\ X \longrightarrow A & Y \longrightarrow B & Z \longrightarrow C \end{array}$$

Figure 1 – Code de César

Ainsi, chaque caractère subit une **substitution** indépendante de sa position : Il s'agit d'un système mono-alphabétique (qui transforme les lettres une à une et toujours de la même manière), qui peut être formalisé de la manière suivante, en utilisant la convention présentée figure 2 :

$$\mathbf{E}_3(x) = x + 3 \bmod 26 \qquad \mathbf{D}_3(y) = y - 3 \bmod 26.$$

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

FIGURE 2 – Table d'addition

2.2 Une première généralisation

Une première généralisation du chiffrement de César permet la variation du décalage (fixé à 3 par César) : la valeur du décalage est alors appelée la *clé secrète* du mécanisme. Mais ce nombre de clés demeure petit, et même énumérable "à la main".

2.3 Code de Vigenère

Au Moyen-Âge, Vigenère a imaginé une extension *poly-alphabétique* du Code de César. Il s'agit d'une substitution variable : le décalage varie en fonction de la position du caractère. Ainsi, un caractère ne sera pas toujours transformé de la même manière. Ces variations sont paramêtrées par la *clé secrète* composée de n lettres, représentant chacune la valeur du décalage (e.g. 'A' = décalage de 0, 'B' = décalage de 1, etc).

Un message de longueur ℓ est découpé en blocs de n caractères, puis chaque bloc subit la transformation suivante : la première lettre est décalée selon la première lettre de la clé, la deuxième lettre est décalée selon la deuxième lettre de la clé, etc. (voir la convention présentée figure 2).

Message =	TEXTECLAIR	:	T	E	X	T	E	C	L	A	I	R
Clé =	BIP	:	B	I	P	B	I	P	B	I	P	B
Chiffré =	UMMUMRMIXS	:	U	M	M	U	M	R	M	I	X	S

Il ne s'agit plus d'une simple substitution indépendante de la position, le nombre de clés possibles est égal à 26^n et devient donc très rapidement énorme.

3 Cryptanalyse

3.1 Codes de César

Il n'y a bien sûr pas de mystère à dire que le code de César n'apporte aucune sécurité, même en permettant un décalage variable. En effet, la recherche exhaustive, qui consiste à essayer toutes les possibilités (en l'occurrence 26), est faisable à la main !

3.2 Code de Vigenère

L'extension de Vigenère, semble beaucoup plus robuste : une clé de 20 caractères présente $2 \times 10^{28} = 2^{94}$ possibilités, ce qui nécessiterait plus d'un million d'années pour toutes les essayer sur un parc d'un million de machines ! Et pourtant, elle ne résiste pas aux cryptanalyses utilisant les redondances de la langue française.

En effet, un cassage total est très rapidement possible sur un chiffré, avec une information supplémentaire sur le texte clair : langue française, ou toute autre langage redondant (anglais, code informatique, etc).

Cette cryptanalyse se décompose en trois étapes :

1. on trouve la longueur n de la clé
2. on trouve une liste de clés probables (la clé à 1 décalage près)
3. on extrait la bonne clé

3.2.1 Test de Kasiski

Nous allons décrire la recherche de la longueur du mot clé. Ceci s'effectue à l'aide du test de Kasiski (1863) qui repose sur le fait que deux segments identiques du texte clair à distance m l'un de l'autre sont chiffrés de la même manière (chiffrement mono-alphabétique) si $m = 0 \bmod n$.

Définition (*Indice de coïncidence.*) Soit $x = x_0x_1...x_{\ell-1}$ une chaîne de ℓ caractères, on appelle *indice de coïncidence* de x, la probabilité $\mathsf{IC}(x)$ que deux caractères aléatoires de x soient égaux.

On suppose d'abord que le texte chiffré provient d'un texte clair écrit dans une *langue naturelle* dont on connaît les propriétés statistiques : nous noterons $p_0, \ldots, p_{25}$, les probabilités des 26 caractères dans la langue naturelle. Pour un texte français ou anglais,

$$\sum_{i=0}^{25} p_i^2 \approx 0.065.$$

Si f_i, pour $i = 0, \ldots, 25$, représentent les fréquences des 26 caractères dans le message, nous avons

$$\mathsf{IC}(x) = \frac{\sum_{i=0}^{25} f_i(f_i - 1)}{\ell(\ell - 1)}.$$

Si le texte est assez long, on peut s'attendre à trouver $\mathsf{IC}(x) \approx \sum_{i=0}^{25} p_i^2 \approx 0.065$. En revanche, pour une chaîne complètement aléatoire x, $\mathsf{IC}(x) \approx 1/26 < 0.04$.

Remarque : *La fréquence d'un caractère dans un message est son nombre d'occurrences, tandis que sa probabilité est la fréquence divisée par la longueur du message :* $p_i = f_i/\ell$.

On va donc extraire du chiffré $c = c_0c_1\ldots$, pour une longueur k de clé, la sous-chaîne $C^{(k)} = C_0^{(k)}C_1^{(k)}\ldots$ définie par $C_i^{(k)} = c_{i\times k}$. En essayant de maximiser $\mathsf{IC}(C^{(k)})$ avec plusieurs longueurs de clés k, il y a de fortes chances pour que le maximum corresponde à la longueur effective de la clé, ou tout du moins à un multiple.

En effet, pour la bonne longueur de clé n (ou tout multiple), la distribution des lettres de la sous-chaîne extraite suit une distribution identique à celle du message initial, à un décalage constant près. Or, l'indice de coïncidence d'une chaîne est indépendant de tout décalage *constant*.

3.2.2 Recherche des clés probables

Maintenant que l'on a la longueur n de la clé, on considère les sous-chaînes $C^{(i)}$ (pour $i = 0$ à $n-1$), où $C^{(i)} = c_i c_{n+i} c_{2n+i} \dots$ du chiffré c. Notons k_i le décalage de la sous-chaîne $C^{(i)}$ par rapport au message clair : cette valeur correspondant à la i-ième lettre de la clé. Pour tout couple (i, j), la sous-chaîne $C^{(j)}$ aura la même distribution de caractères que la chaîne $C^{(i)}$ avec un décalage $k_j - k_i$. On calcule donc l'indice de coïncidence mutuelle (voir ci-dessus) entre la chaîne $C^{(0)}$ et chacun des (26) décalages de $C^{(i)}$. La valeur de décalage $k_i - k_0$ maximisera cet indice.

Définition (*Indice de coïncidence mutuelle.*) Soient x et y deux chaînes de longueur ℓ et ℓ'. On appelle *indice de coïncidence mutuelle* de x et de y, la probabilité qu'un caractère aléatoire de x soit égal à un caractère aléatoire de y :

$$\mathsf{ICM}(x, y) = \frac{\sum_{i=0}^{25} f_i f'_i}{\ell \ell'},$$

où

- les f_i sont les fréquences des 26 caractères dans x,
- les f'_i sont les fréquences des 26 caractères dans y.

La recherche exhaustive est alors très rapide : $26 \times (n-1)$ indices à calculer. En choisissant les maxima, on obtient tous les décalages relatifs $k_1 - k_0$, $k_2 - k_0$, etc.

3.2.3 Extraction de la clé

Nous avons désormais tous les décalages relatifs (par rapport au premier) : $k_1 - k_0$, $k_2 - k_0$, $\dots$, $k_{n-1} - k_0$, soit la clé à un décalage près. Une recherche exhaustive sur k_0 nous fournit la clé. Mais de façon plus rusée, si on sait que le message clair est du texte, le caractère le plus fréquent dans le message chiffré correspond certainement à la lettre 'e', ou à l'espace...

Remarque : *On pourra aussi comparer la distribution des déchiffrés possibles avec un message type, représentatif du langage du message. On cherchera notamment à maximiser l'indice de coïncidence mutuelle entre ce message de référence et chacun des déchiffrés possibles.*
Ceci permet d'appliquer la cryptanalyse à tout type de langage redondant (anglais, français, mais tout langage de programmation, Byte-Code Java, exécutable, etc).

3.2.4 Morale...

À une recherche exhaustive sur la clé, exponentielle en n, on a substitué une cryptanalyse en temps linéaire en n. Cette cryptanalyse radicale montre que la recherche exhaustive n'est pas toujours la seule attaque possible, même lorsque le chiffrement " semble " robuste.

4 Les fonctions utiles

4.1 Entrées-sorties

Une difficulté dans la plupart des langages de programmation est la gestion des entrées-sorties, soit les lectures et écritures de fichiers. Afin de faciliter cette tache, un nouveau type est mis à votre disposition, le type `string` qui est en fait une chaîne de caractères, avec sa longueur

```
typedef struct {
  char *content;
  int length;
} string;
```

ainsi que les fonctions de lecture et d'écriture de fichiers :

```
string readstring(char *file);
void writestring(char *file, string s);
```

La première prend comme argument un nom de fichier, puis retourne un objet de type `string` contenant les caractères du fichier. La seconde fait l'opération inverse : elle prend comme arguments un nom de fichier, ainsi qu'un objet de type `string` qu'elle copie dans le fichier.

On pourra se demander pourquoi on redéfinit ce type `string`, qui semble être contenu dans le type " chaîne de caractères ", qui admet déjà une fonction de longueur (fonction `strlen`). Cependant, les objets manipulés dans ce projet seront des contenus de fichiers, c'est-à-dire des séries d'octets qui peuvent prendre toutes les valeurs, de 0 à 255. Or, dans une chaîne de caractères, l'octet de valeur 0 représente le caractère `'\0'`, et donc le caractère de fin de chaîne : le contenu du fichier serait alors tronqué à la première occurrence d'un octet nul. Par conséquent, il n'est pas question d'utiliser la fonction `strlen` sur le champ `content`.

4.2 Fonctions de base

Pour programmer la cryptanalyse du chiffrement de Vigenère, nous avons besoin des fonctions suivantes [1] :

- `int * Frequence(string s);`
  qui prend un message en entrée (de type `string`), et retourne la table des fréquences. L'alphabet que l'on va utiliser ne va pas se limiter aux 26 lettres, mais va s'étendre sur tous les caractères possibles dans `char`, soit 256 valeurs. Cela rend à la fois le programme plus général, et plus facile à écrire.
- une fonction qui calcule l'indice de co"incidence d'une chaîne, à partir de sa table de fréquences.
- une fonction qui calcule l'indice de coïncidence mutuelle de deux chaînes, à partir de leur table de fréquences.

5 Programmes à effectuer

5.1 Chiffrement/déchiffrement de Vigenère

Avant de tenter toute cryptanalyse, il faut bien maîtriser le système à attaquer. Vous allez donc tout d'abord programmer le chiffrement et le déchiffrement de Vigenère. Il doit permettre le chiffrement de tout fichier : l'alphabet utilisé est donc l'ensemble des `char`.

L'utilisation de l'exécutable doit être la suivante :

```
vigenere c/d <file-in> <file-out> <cle>
```

- le nom de l'exécutable est `vigenere` ;
- le premier argument est soit le caractère `c`, pour chiffrement, soit le caractère `d`, pour déchiffrement ;

1. `http://www.di.ens.fr/~fouque/my_string.c`, `http://www.di.ens.fr/~fouque/my_string.h`

— le deuxième argument est le nom du fichier à traiter ;
— le troisième argument est le nom du fichier dans lequel on sauvegarde le résultat ;
— le quatrième argument est la clé — on pourra se limiter aux 26 lettres de l'alphabet pour éviter les interprétation du SHELL (puis pour permettre une mémorisation plus aisée).

5.2 Cryptanalyse

Pour la cryptanalyse, une fois les fonctions de base écrites, il s'agit de simples manipulations de messages (de type `string`) :

1. On commence par lire le fichier à traiter, pour le transformer en `string` ;
2. Pour toutes les longueurs de clé possibles (on se fixera un maximum, disons 50), on calcule l'indice de coïncidence des sous-messages composés des caractères à distance la longueur de clé testée. Le maximum correspond probablement à un multiple k de la longueur de la clé ;
3. On extrait les k sous-messages, correspondants à chacun des caractères de la clé. On calcule les indices de coïncidence mutuelle entre le premier sous-message et tous les décalages possible du second : l'indice maximum fournit le décalage relatif probable entre ces deux sous-messages. On fait de même entre le premier sous-message et le troisième, ..., entre le premier et le k-ième ;
4. On pourra se contenter d'afficher les clés possibles, pour tous les décalages possibles du premier sous-message.

L'utilisation de l'exécutable doit être la suivante :

```
kasiski <file>
```

où l'argument est le fichier à traiter, un message chiffré par Vigenère. Il retourne la longueur de la clé, la liste des clés possibles, et éventuellement la clé la plus probable.

5.3 Amélioration

Deux améliorations sont possibles et intéressantes :
— la première est simple, puisqu'elle utilise des fonctions déjà écrites. Pour l'étape 4, on peut trouver automatiquement la bonne clé si on connaît de quel type est le message clair (texte, exécutable, source C, etc). On prend alors un fichier de référence de même type que le message clair, puis on calcule l'indice de coïncidence mutuelle entre ce message de référence et chaque déchiffrement (avec chacune des clés possibles). L'indice maximum correspond à la clé la plus probable.
— comme indiqué ci-dessus, l'étape 2 fournit un multiple de la taille réelle de la clé, mais on ne peut guère faire mieux à cette étape. Cependant, après l'étape 3, on voit apparaître les répétitions dans la clé. On peut alors réduire la taille de la clé à sa valeur minimale en prenant le motif de base (sans ses répétitions). Il s'agit alors d'extraire le motif de base d'une chaîne.

L'utilisation de l'exécutable devient :

```
kasiski <file> <file-ref>
```

où le premier argument est le fichier à traiter, tandis que le second est le message de référence (contenant le même type de redondance que le message clair recherché). Il retourne la longueur de la clé ainsi que la clé la plus probable.