Compte-rendu de projet d’ISR

Robin Adili

Chiffrement de Vigenère et cryptanalyse

L’objectif de ce TP est de réaliser un programme qui permet de chiffrer et de déchiffrer un texte en utilisant le chiffrement de Vigenère, puis d’étudier et de mettre en oeuvre la cryptanalyse proposé par Kasiski, qui permet de retrouver la clé de chiffrement.

Chiffrement de Vigenère

Il s’agit d’un chiffrement similaire au chiffrement de César, la différence étant que le décalage n’est pas le même pour tout le texte. Chaque lettre est décalé par la lettre de même position dans la clé, la clé est répétée si elle est plus courte que le texte. Le déchiffrement s’opère par décalage du chiffré par la clé dans le sens inverse.

L’implémentation de cet algorithme de chiffrement est assez simple. On fait le choix de l’alphabet ASCII plutôt que UTF-8 ou Unicode afin de préserver une combinatoire de taille raisonnable pour la cryptanalyse.

Implémentation

Les fonctions `encrypt` et `decrypt` effectuent le chiffrement et le

déchiffrement d'une chaine de caractère par une clé. On choisit de généraliser ces fonctions à des chaines de caractère plutôt qu'à des fichiers pour pouvoir les réutiliser notamment lors de la cryptanalyse.

Ces fonctions itèrent simplement sur les caractères de la chaine et appellent respectivement `char_encrypt` et `char_decrypt` avec le caractère courant et le caractère de la clé correspondant.

Le chiffrement d'un caractère se fait par addition de son indice dans la table ASCII avec celui du caractère de la clé, modulo la taille de la table. Le résultat est l'indice du caractère chiffré. Le déchiffrement procède par soustraction de l'indice du caractère chiffré par celui du caractère de la clé.

On réalise également des fonctions qui lisent le texte d'entrée à partir d'un fichier et écrivent le résultat dans un autre.

Interface en ligne de commande

L'utilisation du programme `vigenere.py` en ligne de commande est la suivante :

```
usage: vigenere [-h] {c,d} file_in file_out key

positional arguments:
  {c,d}       c for encrypting, d for decrypting
  file_in     input file
  file_out    output file
```

```
  key

optional arguments:
  -h, --help  show this help message and exit
```

Le traitement des argument en ligne de commande utilise le module argparse de Python. Il permet notamment de spécifier un type d’argument fichier et de récupérer ce fichier ouvert à l’issue du parsing

Cryptanalyse de Kasiski

Le chiffrement de César décalant l’intégralité du message clair par une valeur fixe, il est possible de déchiffrer le message par analyse de la fréquence des caractères. La valeur de la clé est donnée par le décalage entre caractère le plus fréquent du message chiffré et le caractère que l’on suppose le plus fréquent dans le message clair.
Puisque le chiffrement de Vigenère utilise une clé de plusieurs caractères généralement plus petite que le message clair, plusieurs caractères sont chiffrés avec le même caractère de la clé. Il est alors possible de réaliser la même analyse sur les sous-séquences du message qui sont chiffrées par le même caractère. Ces sous-séquences sont telles que l’indice de leur éléments dans le message modulo la taille de la clé est le même.

On doit donc commencer par retrouver la taille de la clé.

Déterminer la taille de la clé

Pour ce faire on utilise l'Indice de Coincidence. Cet indice se calcule sur la distribution de probabilité d'un message. Il prend des valeurs très caractéristiques pour chaque langage. Cela permet d'identifier la langue d'un texte, mais également de différencier un texte écrit dans une langue d'un texte aléatoire. En effet un texte aléatoire aura une distribution de fréquence des caractères bien plus uniforme, et son indice sera donc plus faible.

Une sous-séquence chiffrée par la même lettre a la même distribution de fréquence que le texte clair, elle aura donc un indice de coincidence proche de celui du texte clair. Cet indice sera également plus élevé que celui d'une sous-séquence dont les éléments ne sont pas chiffré par la même lettre, et qui s'apparente à du texte aléatoire quant à sa distribution de fréquence.
On peut ainsi essayer de calculer les indices de coincidence des sous-séquences pour chaque longueur de clé dans un intervalle. La longueur qui maximise les indices de coincidence de ses sous-séquence sera la longueur de la clé la plus probable.

La fonction `findKeyLength` réalise cette opération. Pour chaque longueur `l` d'un intervalle donné, elle calcule les `l` sous-séquences puis fait une moyenne des indices de coincidence de chaque sous-séquence. Le résultat est la longueur pour laquelle cette moyenne est la plus élevée.

Déterminer la clé la plus probable

Grâce à l'analyse précédente, on connait désormais les sous-séquences

chiffrées par la même lettre. On utilise deux méthodes différentes pour déterminer la clé la plus probable.

La première s'appuie sur une analyse de fréquence simple, basée sur la supposition du caractère le plus fréquent dans le message clair. Elle revient à réaliser l'analyse du chiffrement de César pour chaque sous-séquence chiffrée par la même lettre.
Cette méthode est implémentée par la fonction `simple_guess`. Pour chaque sous-séquence, elle calcule la distribution de fréquence de ses caractères. Le caractère de la clé qui a chiffré cette sous-séquence vaut alors la différence entre l'indice de son caractère le plus fréquent dans la table ASCII et celui du caractère supposé le plus fréquent dans le message clair. Il y a un modulo taille de la table ASCII sur cette différence par ce qu'un caractère chiffré peut dépasser la taille de la table.

Pour la deuxième méthode, on voudra s'appuyer sur l'indice de coincidence mutuel. Cet indice est calculé sur la distribution de fréquence de deux messages et donne la probabilité qu'un caractère du premier soit égal à un caractère du deuxième. Il permet de comparer les distributions de fréquence de sorte à déterminer si elles sont similaires caractère à caractère.
Les sous-séquences peuvent donc être comparées entre elles pour déterminer le décalage qui rendra une sous-séquence similaire à une autre du point de vue des fréquences de chaque caractère.
Pour implémenter cette méthode, la fonction `icm_guess` calcule les distributions de fréquence des sous-séquences, puis pour chaque décalage possible des d'une distribution, calcule son indice de

coincidence mutuel avec la distribution de la première sous séquence. Est retenu pour chaque sous-séquence le décalage qui maximise cette indice. Ainsi sont obtenus les décalages des caractères de la clé par rapport au premier.
On peut alors extraire toutes les clés possibles en donnant une valeur au premier caractère. La fonction `possible_keys` réalise ce calcul. Ne sont retenues que les clés comportant des caractères imprimables (>32 et <127).

Connaissant l'ensemble des clés possibles, on adopte deux approches différentes pour déterminer la clé la plus probable.
La première est encore une fois de supposer qu'un caractère est le plus courant dans le message clair. Cette supposition permet de trouver le premier caractère de la clé en appliquant la méthode d'analyse simple évoquée précédemment sur la première sous-séquence. Cette première approche est implémentée par la fonction `mf_shift_guess`. Elle détermine le premier caractère `shift0` de la clé d'après le caractère le plus fréquent `mf` et calcule le reste de la clé en appliquant les décalages des autres caractères à celui du premier.
La deuxième approche suppose que l'on dispose d'un message clair de référence, similaire dans sa distribution de fréquence au message clair recherché. Avec ce message de référence, on peut comparer sa distribution caractère à caractère à celles des messages déchiffrés par chaque clé possible. C'est à dire calculer l'indice de coincidence mutuel entre message de référence et déchiffré pour une clé possible. La fonction `icm_shift_guess` implémente cette approche. Pour chaque clé, le déchiffré est calculé avec la fonction `decrypt` de `vigenere.py`, puis sa distribution de fréquence est comparé à celle du message de

référence avec un indice de coincidence mutuel.
La clé retournée est celle dont le déchiffré maximise cette indice de coincidence mutuel.

Interface en ligne de commande

L'utilisation de `kasiski.py` est la suivante :

```
usage: kasiski [-h] [-t TYPICAL_FILE] [-m MOST_FREQUENT_CH
AR] [-s] [-i] file

positional arguments:
 file                  file on which the cryptanalysis is
performed

optional arguments:
 -h, --help            show this help message and exit
 -t TYPICAL_FILE, --typical-file TYPICAL_FILE
                       plain text file of similar characte
r distribution than
                       the encrypted plain text. With icm
guess (-i), is used
                       to guess the offset of the key's fi
rst character given
                       the shifts of every character to th
e first one. Is
                       compared to the plaintext of every
possible key by
```

```
                        computing their icm With simple gue
ss (-s), is used to
                        compute the most frequent char Is n
ot compatible with
                        -m
 -m MOST_FREQUENT_CHAR, --most-frequent-char MOST_FREQUENT
_CHAR
                        most frequent character in the plai
n text, is used to
                        guess the key using simple guess, a
nd to choose one
                        from possible keys using icm guess
Is not compatible
                        with -t
 -s, --simple-guess     Guess the key using the simple gues
s method Only
                        assumes a most frequent character,
and deduces the key
                        from that
 -i, --icm-guess        Guess the key using the icm guess m
ethod Actually try
                        and match frequency distributions o
f sub sequences
                        corresponding to key characters in
order to find a set
                        of possible keys, then use the spec
ified (default: -m
                        ' ') method to choose the most prob
```

```
able key```

A également été utilisé le module argparse de Python. À no
ter que les options `-i` et `-s` pour chaque type de métho
de de détermination de la clé ne sont pas compatibles. Les
 options `-m` et `-t` permettent de choisir une des deux a
pproche de détermination de la clé la plus probable parmi
les possibles dans le cas de la méthode `icm_guess`. Pour
la méthode `simple_guess` ces options permettent dans un c
as de donner le caractère le plus fréquent et dans l'autre
 de le calculer d'après le texte de référence.
```