

Introduction à la sécurité

TP 1 : Cryptographie historique

N'oubliez pas :

- Les TPs doivent être rendus par courriel au plus tard la veille de la séance suivante avec “[is]” suivi du numéro de la séance et de votre nom dans le sujet du mail, par exemple “[is] TP1 Rauzy”.
- Quand un exercice demande des réponses qui ne sont pas du code, vous les mettez dans un fichier texte `reponses.txt` à rendre avec le code.
- Le TP doit être rendu dans une archive, par exemple un tar gzippé obtenu avec la commande `tar czvf NOM.tgz NOM`, où `NOM` est le nom du répertoire dans lequel il y a votre code (idéalement, votre nom de famille et le numéro de la séance, par exemple “rauzy-tp1”).
- Si l'archive est lourde (> 1 Mo), merci d'utiliser <https://bigfiles.univ-paris8.fr/>.
- Les fichiers temporaires (si il y en a) doivent être supprimés avant de créer l'archive.
- Le code doit être proprement indenté et les variables, fonctions, constantes, etc. correctement nommées, en respectant des conventions cohérentes.
- Le code est de préférence en anglais, les commentaires (si besoin) en français ou anglais, en restant cohérent.
- **N'hésitez jamais à chercher de la documentation par vous-même sur le net!**

Dans ce TP, les objectifs sont d'implanter :

- le chiffrement de Cesar avec une généralisation possible, le chiffrement affine;
- des attaques statistiques sur ces chiffrements;
- le chiffrement de Vigenère;
- une attaque statistique sur ce chiffrement.

Exercice 0.

Recommandations.

1. Ce TP est à faire en Python.
2. Faites les questions dans l'ordre et pensez à tester votre code.
3. N'hésitez jamais à rajouter de la sortie de debug pour comprendre tout ce qui se passe.

Exercice 1.

Chiffrement affine

1. Dans cet exercice, on souhaite **implanter et attaquer un chiffrement affine simple**. On considère un ensemble de symboles (l'alphabet) formé des 26 lettres courantes en majuscule `A-Z`, et de trois caractères spéciaux : l'espace « », l'apostrophe « `'` » et le point « `.` ».

On souhaite encoder ces 29 caractères avec les entiers allant de 0 à 28, en commençant par le `A` et en terminant par le point.

→ Écrire des fonctions `encode` et `decode`, qui permettent respectivement d'encoder du texte comme une liste d'entiers, et de décoder une liste d'entiers comme du texte, selon la règle d'encodage énoncée ci-dessus.

2. Le chiffrement de Cesar vu en cours s'applique aussi bien sur l'alphabet classique à 26 éléments, que sur notre alphabet à 29 éléments.

→ Écrire des fonctions `chiffre_cesar` et `dechiffre_cesar`, qui permettent respectivement de chiffrer un texte clair et de déchiffrer un texte chiffré avec une clé représentée sous la forme d'un entier compris entre 0 et 28.

3. Dans notre alphabet à 29 éléments, la distribution des caractères est différente de celle présentée en cours, car on a ajouté trois éléments dont le caractère d'espacement qui est assez courant. Dorénavant, les caractères les plus fréquents sont les suivants :

caractère	espace	<i>E</i>	<i>A</i>	<i>S</i>	<i>I</i>
fréquence	0.165	0.136	0.076	0.069	0.065

→ Écrire une fonction `plus_frequent` qui retourne le caractère le plus fréquent d'un texte.

- Retrouver le texte clair associé au texte chiffré par un chiffrement de Cesar, qui est donné dans le fichier `enc_cesar.txt`.
- On s'intéresse maintenant à une autre catégorie de chiffrement, appelée « chiffrement affine ». L'idée est de généraliser le chiffrement de Cesar à d'autres types de substitutions de lettres. Pour Cesar, la substitution était donnée par :

$$x \mapsto x + \Delta \pmod{29}$$

où x est la lettre à chiffrer, et Δ est la clé de décalage.

Pour un chiffrement affine, la permutation est de la forme :

$$x \mapsto ax + b \pmod{29}$$

où le couple (a, b) forme la clé secrète, avec comme seule contrainte que a soit être différent de 0. Pour déchiffrer, on doit alors appliquer la permutation inverse, à savoir :

$$y \mapsto (y - b) \cdot a^{-1} \pmod{29}$$

Remarque. Pour obtenir l'inverse modulaire « $a^{-1} \pmod{29}$ », en `python` il suffit d'exécuter `pow(a, -1, 29)`.

→ Écrire des fonctions `chiffre_affine` et `dechiffre_affine`, qui permettent respectivement de chiffrer un texte clair et de déchiffrer un texte chiffré avec une clé `cle` représentée sous la forme d'un couple d'entiers.

- Vérifier que le chiffré de `INFORMATIQUE` par la clé $(a = 13, b = 12)$ est `AHTUBXM'ARLG`, puis que le déchiffrement se déroule correctement.
- Pour retrouver la clé Δ du chiffrement de Cesar, il a suffi de reconnaître le chiffré d'une seule lettre (en l'occurrence, la plus fréquente). Pour retrouver la clé du chiffrement affine, qui est formée de deux entiers (a, b) , il nous faudra la connaissance du chiffrement de deux lettres.

Supposons que l'on connaisse y le chiffré d'un caractère x , et y' le chiffré de x' . Alors on a le système de deux équations à deux inconnues (a, b) :

$$\begin{cases} y = ax + b \pmod{29} \\ y' = ax' + b \pmod{29} \end{cases}$$

que l'on sait résoudre pour obtenir

$$a = (y - y') \cdot (x - x')^{-1} \pmod{29}, \quad \text{puis} \quad b = y - ax \pmod{29}.$$

→ Écrire une fonction `deux_plus_frequents` qui retourne les deux caractères les plus fréquents d'un texte.

- Retrouver le texte clair associé au texte chiffré par un chiffrement affine, qui est donné dans le fichier `enc_affine.txt` du dossier `affine`.

Exercice 2.

Attaque sur le chiffrement de Vigenère.

- On souhaite attaquer le chiffrement de Vigenère tel que vu en cours, c'est-à-dire avec un alphabet de 26 lettres. Implanter les fonctions de chiffrement et de déchiffrement de Vigenère, puis tester ces fonctions sur un exemple.
- Implanter une fonction `IC` qui calcule l'indice de coïncidence d'un texte. Pour rappel, cet indice est donné par la relation :

$$IC = \frac{1}{N(N-1)} \sum_{i=1}^{26} M_i(M_i - 1)$$

où N est le nombre de caractères dans le texte, et M_i est le nombre de fois que le i -ème caractère apparaît dans le texte.

- Implanter une fonction `calcule_longueur_cle` qui prend en entrée un texte chiffré, et qui retourne la longueur de clé probable utilisée pour obtenir ce chiffré.
- Implanter (ou réutiliser de l'exercice précédent) une fonction `calcule_decalage` qui calcule le décalage probable d'un morceau de texte chiffré avec Cesar.
- Implanter une fonction `attaque_vigenere` qui décrypte un chiffré de Vigenère (sans la clé, donc).
- Décrypter les chiffrés contenus dans les fichiers `file<i>_enc.txt` du dossier `vigenere`.