Challenge Cryptanalyse

● Vous avez dû recevoir une archive spécifique à votre groupe par mail. Tous les chemins indiqués dans ce sujet sont relatifs à la racine de l'archive.

Deux chiffres historiques

Défi n° 1

Le message `Defi1/message.txt`, initialement en français, a été chiffré par une méthode élémentaire.

▲ Saurez-vous le déchiffrer ?

Défi n° 2

Le message `Defi2/message.txt`, initialement en anglais, a été chiffré par une méthode classique à clef, résistante à la cryptanalyse du précédent défi.

▲ Trouvez et implémentez une méthode de cryptanalyse adéquate.

Enigma

■ Une implémentation de la machine Enigma est disponible en Java (dans `enigma/java`) et en Python (dans `enigma/python`). Vous utiliserez indifféremment l'un ou l'autre des langages selon votre préférence. La machine implémentée est absolument identique dans les deux langages. Elle comporte 3 rotors, dont l'ordre peut être modifié, et au plus 6 fiches afin de définir une permutation dans le *brouilleur* (tableau de fiches) par échange de lettres.

Une *clef* de la machine Enigma est la donnée de toutes les informations nécessaires pour configurer la machine afin de crypter/décrypter (cela revient au même du fait du réflecteur) un message. La clef comporte donc : l'ordre des rotors, leurs positions initiales et les fiches utilisées dans le brouilleur.

Rappelons avant de commencer la structure d'un message crypté. Tous les opérateurs allemands disposaient d'une même *clef du jour* définie à l'avance et valable une journée donnée. Afin de ne pas chiffrer tous les messages avec cette même clef (*question subsidiaire :* expliquer pourquoi), l'opérateur choisissait avant l'envoi d'un message un autre jeu de positions initiales pour les rotors. Il mettait alors sa machine dans la configuration du jour, tapait la clef choisie deux fois pour prévenir les erreurs (par exemple, pour les positions $(0, 1, 2)$, il entrait "ABCABC"), puis mettait les rotors dans les positions choisies et chiffrait le message. Ce *préfixe* de 6 caractères a donc un rôle différent et n'est pas chiffré avec la même clef que le corps du message.

Défi n° 3

Les deux messages `Defi3/message1.txt` et `Defi3/message2.txt` ont été interceptés par les télécommunications alliées. Nos espions nous informent en outre que la machine utilisée est identique à celle qui vous a été fournie et qu'au plus trois fiches du brouilleur ont été utilisées pour le chiffrement. En revanche, la clef n'est pas connue et est différente pour les deux messages.

▲ Pourrez-vous les cryptanalyser rapidement en exploitant la puissance des ordinateurs d'aujourd'hui ?

Défi n° 4

Le cryptologue Marian Rejewski fut au début des années 1930 à l'origine de la première (et plus jolie) avancée dans la cryptanalyse d'Enigma. Exploitant la répétition des clefs choisies au début de chaque message, sa méthode permettait d'identifier très rapidement un ensemble généralement très restreint (quelques dizaines) de possibles clefs du jour (hors brouilleur) après avoir analysé les six premiers caractères de seulement quelques dizaines, voire centaines, de messages interceptés dans la journée.
La méthode de Rejewski se base sur des empreintes, extraites des messages reçus dans la journée, que l'on présentera sous la forme de trois relations binaires $\xrightarrow{1\text{-}4}$, $\xrightarrow{2\text{-}5}$ et $\xrightarrow{3\text{-}6}$ liant respectivement les 1er et 4^{e}, 2^{e} et 5^{e},

3e et 6e caractères des préfixes interceptés. Partant initialement de relations vides, on enrichit nos relations binaires, à chaque préfixe "ABCDEF" intercepté, des associations $A \xrightarrow{1\text{-}4} D$, $B \xrightarrow{2\text{-}5} E$ et $C \xrightarrow{3\text{-}6} F$. Dès qu'une relation associe au moins une lettre à chacune des lettres de l'alphabet, on considère sa construction *complète*. Ces trois relations constituent les empreintes extraites par Rejewski.

▲ On pourra supposer dans toute la suite que les Allemands ne font jamais de faute de frappe (il n'y a donc jamais d'erreur dans le préfixe des messages interceptés) et que suffisamment de messages ont été interceptés pour que les empreintes soient complètes.

Les trois questions théoriques suivantes vous aiderons à comprendre la méthode de Rejewski. Il n'est **pas indispensable** d'avoir traité ces questions avant de passer à l'implémentation de la méthode.

- ▲ Justifier que les trois empreintes extraites par Rejewski forment des permutations.
  Indication. Il s'agit pour cela de justifier que les relations formées sont bien fonctionnelles (c'est-à-dire qu'à toute lettre est associée une unique autre lettre) et bijectives.
- ◆ Considérant deux permutations σ et α, quelle permutation obtient-on en composant $\sigma' = \alpha \circ \sigma \circ \alpha^{-1}$?
  On dit que σ' est une *conjuguée* de σ.
- ⬟ Une permutation se décompose de façon unique en un produit de cycles à supports disjoints. On appelle *partition* d'une permutation la donnée des tailles des cycles de cette décomposition.
  Montrer que l'ensemble des conjuguées d'une substitution σ est exactement l'ensemble des permutations ayant la même partition que σ.

Ainsi, la partition est invariante par conjugaison. Elle constitue donc une information ne dépendant que de l'ordre et de la position initiale des rotors. Elle est indépendante du brouilleur de la machine. Or c'est précisément ce dernier qui fait exploser le nombre de configurations ! Il devient donc possible de précalculer pour toutes les configurations (hors brouilleur) de la machine les empreintes associées. Ce travail, effectué à la main, avait demandé un an aux Polonais, mais il ne prendra pas plus de quelques minutes sur votre ordinateur. Il suffisait ensuite chaque jour de construire les empreintes associées à la clef du jour à partir des messages interceptés (quelques centaines de messages suffisent à construire des empreintes complètes) et de retrouver dans le résultat du précalcul les quelques configurations (hors brouilleur) fournissant les mêmes empreintes.

- ⬢ Implémenter la méthode de Rejewski.
  Le fichier `Defi4/prefixes.txt` contient une liste de préfixes interceptés dans la journée (et suffisante pour construire des empreintes complètes). Appliquer la méthode pour retrouver les configurations possibles des rotors de la machine ayant produit ces préfixes.

Il reste encore à identifier la bonne clef partielle parmi les clefs retrouvées, puis, pour compléter la clef du jour, à déterminer la configuration du brouilleur. Comme il n'y a que 6 fiches, une proportion non négligeable des messages comporte un préfixe non affecté par le brouilleur. Si l'on a choisi la bonne clef, on peut alors déchiffrer correctement (au brouillage près) le corps de ces messages et des portions de mots en allemand devraient apparaître. Ne reste plus qu'une simple substitution monoalphabétique à casser.

▲ **On ne vous demande pas d'implémenter ces dernières étapes.**

Images brouillées

Défi n° 5

La *transformation du boulanger* est une transformation bijective d'image, i.e. une réorganisation réversible des pixels d'une image, consistant à étirer l'image horizontalement en entrelaçant les pixels des lignes paires et impaires, de façon à produire une image deux fois plus large mais deux fois moins haute que l'image d'origine, puis à rabattre la moitié droite de cette image sous la moitié gauche de façon à retrouver le format d'origine. Pour que ce procédé soit bien défini, il faut que les deux dimensions de l'image soient **paires**. Pour une image de dimensions $w \times h$, l'image d'un pixel (x, y) — où x est la position horizontale, y la verticale et

par convention $(0, 0)$ désigne le coin supérieur gauche — est calculée de la façon suivante :

```
def boulanger(x,y,w,h):
  if x<w/2: return (2*x+y%2,y/2)
  else: return (2*(w-1-x)+(1-y%2),h-1-y/2)
```

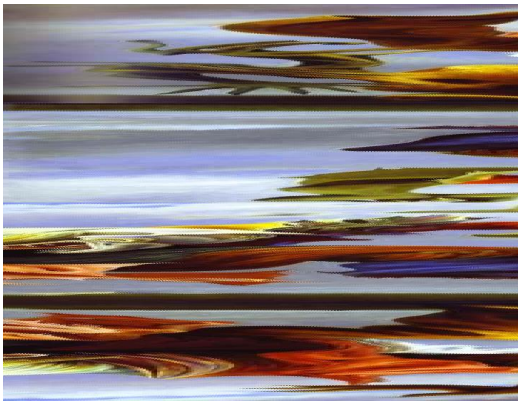
Le répertoire `code_brouillage` de l'archive contient du code afin d'illustrer la manipulation d'image et l'application de cette transformation en Python (requiert la bibliothèque Imaging, package `python-imaging` sous Debian/Ubuntu) et en Java. Encore une fois, vous travaillerez avec le langage de votre préférence.

Au cours de ce défi, vous risquez d'avoir à manipuler de **grands entiers**... Python gère nativement ce type de donnée et permet de faire de l'arithmétique en précision arbitraire de façon totalement transparente pour le programmeur avec les opérateurs usuels. En Java, en revanche, il faut utiliser la classe `BigInteger` et les méthodes spécifiques à cette classe pour réaliser les opérations arithmétiques de base. Des exemples d'utilisation sont fournis dans le fichier `code_brouillage/java/GrandsEntiers.java`.

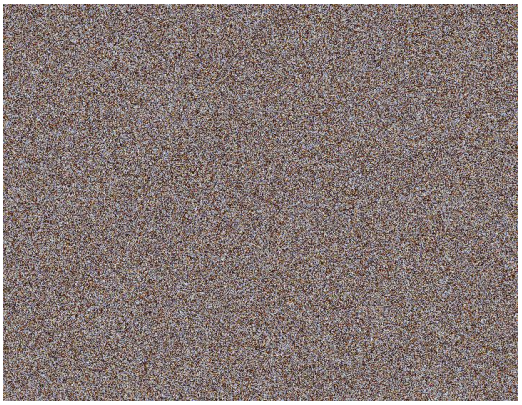
On peut envisager d'itérer la transformation... Si les premières itérations restent lisibles, l'image est par la suite assez rapidement brouillée, comme l'image `Defi5/image.png` que vous devez révéler.

(a) Image initiale

(b) Itération 1



(c) Itération 2



(d) Itération 100

▲ Justifier qu'en itérant la transformation on revient nécessairement en un nombre fini d'étapes sur l'image initiale.
Le nombre minimal d'étapes nécessaires pour revenir est appelé *temps de retour* et ne dépend que des dimensions de l'image.

◆ En considérant la trajectoire de chaque pixel au fil des itérations, proposez un moyen simple d'exprimer le temps de retour d'une image. En déduire une façon rapide (linéaire en la taille de l'image) de le

calculer.
Implémentez cette méthode et calculez le temps de retour de votre image. Qu'en pensez-vous ?

- En déduire une méthode pour calculer la n^{e} itération de la transformation efficacement (en temps linéaire en la taille de l'image et indépendamment de n, si l'on ignore la complexité arithmétique).
- Vous avez maintenant toutes les clefs en main pour révéler l'image `Defi5/image.png` à l'aide des indices contenus dans le fichier `Defi5/indices.txt`. À vous de jouer...