R3.09 -- Cryptographie
TP 1+2

Durée : 2 séances de TP

Langage : indifférent entre Java, Python, Rust

Date limite dépôt de livrables : 05/10 à 23h 59

Livrables : 2 livrables :

1. Le code source dans une archive .zip ou .7z. Convention de nommage : <nom1>_<nom2>TP_1_et_2
2. Un rapport de max. 5 pages (avec des bouts du code), qui explique comment vous « traitez » le texte en entrée pour le chiffrement et déchiffrement Vigenère (quel type de structure de données est utilisée, comment vous transformez –ou non—les caractères en entiers, comment vous traitez les majuscules et minuscules, etc.), ainsi que les étapes suivantes de la méthode de Kasiski : le traitement du texte pour retrouver les répétitions, le stockage de répétitions, comment retrouver la taille de la clé.

Exercice 1 : le chiffrement de Vigenère

En tant que premier exercice, implémentez le chiffrement Vigenère.

Votre programme devra demander en entrée un texte clair, ainsi qu'une clé.

L'output sera le texte chiffré correspondant à un chiffrement Vigenère du texte clair avec la clé donnée.

Particularités : Votre application doit faire des choix (qui seront par la suite justifiés dans votre rapport) quant au traitement des majuscules, minuscules, ponctuation, caractères spéciaux, chiffres, etc. qui peuvent apparaître en entrée. Le chiffrement doit fonctionner peu importe la clé et le texte clair renseignées, tant que ces deux éléments ne contiennent que des lettres (majuscules, minuscules, combinaisons de ces deux), des signes de ponctuation, des caractères spéciaux et des espaces.

Extensions possibles (si jamais vous avez fini l'entièreté du TP) : permettre au programme de prendre en entrée des fichiers pour le texte clair et la clé, et sortir l'output dans un autre fichier. Si vous vous ennuyez encore à la fin de cette tâche, pourquoi pas penser à une interface explicative (utilisable par exemple aux JPOs).

Exercice 2 : le déchiffrement avec Vigenère

Faites une extension à votre programme précédent pour déchiffrer un texte chiffré Vigenère.

Votre programme devra demander en entrée un texte chiffré et une clé.

L'output sera un texte clair correspondant au déchiffrement Vigenère du chiffré avec la clé donnée.

Particularités : dans la continuité du premier exercice, vous allez expliquer comment le déchiffrement est réalisé, notamment par rapport aux majuscules/minuscules, aux caractères spéciaux et aux espaces.

Vérification : Votre programme devra présenter un jeu d'essais qui testent le bon fonctionnement de votre application, selon les règles que vous avez choisies, et notamment il devra vérifier que si on demande de chiffrer un message m (qui doit peut-être subir des modifications telles que la suppression d'espaces, etc.), et que, par le chiffrement, on obtient un texte chiffré c, alors le déchiffrement de ce texte chiffré donne un message m' qui est équivalent (à des espaces/majuscules près) au message initial m.

Détaillez votre jeu d'essais dans le rapport accompagnant le rendu du TP, en argumentant pourquoi ce jeu d'essais couvre l'espace de possibilités.

Exercice 3 : Kasiski et la taille de la clé en Vigenère

Dans cet exercice vous allez implémenter la méthode de Kasiski pour retrouver la taille de la clé utilisée dans un chiffrement Vigenère.

L'exercice doit prendre en entrée un fichier contenant un texte avec les mêmes caractéristiques que ci-dessus (lettres en majuscules et minuscules, avec des espaces, de la ponctuation et des caractères spéciaux) qui est censé d'être un texte chiffré par l'algorithme de Vigenère.

L'output de l'algorithme doit être une valeur, ou éventuellement une liste de valeurs, représentant la taille potentielle de la clé de chiffrement utilisée. Un symbole spécial, comme par exemple « ? » peut être utilisé pour indiquer le fait que l'algorithme ne réussit pas à formuler une hypothèse.

Stratégie : pour rappel, l'algorithme de Kasiski suit les étapes suivantes :

1. En examinant le texte chiffré donné en entrée, trouver des fragments de texte qui se répètent.
2. Calculer, pour chaque fragment, la distance entre les deux (ou plusieurs) occurrences.
3. Lister, dans un tableau qu'on dénote `repet`, les répétitions et les distances qui les séparent, en ordre inverse de la taille du texte qui se répète. Les répétitions de taille égale l'une à l'autre

peuvent être ordonnées aléatoirement. Pour maintenir seulement les valeurs hautement probables de taille de clé, il serait mieux de ne lister que des fragments d'au minimum 3 lettres qui se répètent.

4. Si le tableau `repet` est vide, alors l'algorithme ne peut pas formuler une hypothèse quant à la taille de la clé. L'algorithme s'arrête et l'output est le symbole d'erreur choisi ci-dessus.
5. Si le tableau `repet` n'est pas vide, la première étape est de regarder la première ligne du tableau (qui figure le fragment le plus long qui se répète et la distance entre les occurrences). L'algorithme formule l'hypothèse que la clé doit être un diviseur de la distance entre les occurrences. Une liste temporaire de candidats pour la taille de clé, qu'on dénote `candidats`, est calculée.
6. À partir de maintenant l'algorithme traite le tableau ligne par ligne, en allant de haut en bas. À chaque ligne :
    a. Trouver le PGCD entre chacun des candidats actuels pour la taille de la clé (stockés dans la liste `candidats`) est la distance figurée dans la ligne actuelle du tableau `repet` et les stocker dans une liste intermédiaire qu'on dénote `temp`.
    b. Éliminer toutes les valeurs égales à 1 de la liste intermédiaire `temp`.
    c. Si la liste `temp` n'est pas vide : remplacer `candidats` par `temp`.
    d. Si la liste `temp` est vide :
        i. S'il restent des lignes toujours à parcourir dans le tableau : maintenir la liste candidats telle qu'elle était, passer à la prochaine ligne du tableau.
        ii. S'il ne reste plus aucune ligne dans le tableau, s'arrêter et retourner `candidats`

Vérifications : Faites plusieurs jeux d'essais pour tester la pertinence de votre algorithme et incluez dans l'archive un README contenant les entrées pour ces jeux d'essais : le README doit indiquer le plaintext que vous avez chiffré pour obtenir l'entrée pour votre test, la clé avec laquelle vous les avez chiffrés, ainsi que l'output de votre algorithme. Dans votre rapport, discutez votre choix de plaintexts et de clés pour montrer que vos jeux d'essais couvrent bien l'espace de possibilités.