TD : Méthodes de chiffrement

Nom du fichier à rendre : td_chiffrement_[nom].py

I. Chiffrement par décalage

Dans cette première partie, vous allez implémenter une méthode basique de chiffrement : le chiffrement par décalage.

Pour des raisons de simplicité et d'efficacité, on supposera que toutes les chaînes de caractères manipulées ne sont composées que des symboles suivants :

- Lettres majuscules non accentuées
- Chiffres
- Symbole espace
- Signes de ponctuation

Question 1. Imaginer une fonction `chiffreDeCesarChar` qui prend en argument une chaîne de caractère `c` de longueur 1 et qui renvoie

- le caractère situé 3 lettres plus loin dans l'alphabet que le caractère `c` si `c` est une lettre
- le caractère `c` si `c` n'est pas une lettre

Indice : N'oubliez pas que la lettre `Y` devient la lettre `B`.

Question 2. En déduire une fonction `chiffreDeCesar` telle que `chiffreDeCesar(texte)` renvoie une version chiffrée grâce au chiffre de César de la chaîne de caractère `texte`.

II. Méthode du masque jetable

Dans cette deuxième partie, vous allez implémenter la méthode du masque jetable.

Vous manipulerez donc des messages binaires, représentés en Python par des listes d'entiers dont on suppose qu'ils ne contiennent que des 0 et des 1.

Question 3. Imaginer une fonction `creerMasqueAleatoire` telle que `creerMasqueAleatoire(longeur)` renvoie une liste de longueur `longueur` et composée d'une suite aléatoire de 0 et de 1.

Question 4. Ecrire une fonction `utiliserMasque` telle que `utiliserMasque(message, masque)` applique le masque décrit par la liste `masque` sur le message décrit par la liste `message` et renvoie la liste correspondante

Remarque : On suppose que les chaînes `masque` et `message` ont la même longueur.

Remarque : Cette fonction devrait vous permettre à la fois de chiffrer et de déchiffrer un message.

III. Méthode à clé publique – clé privée

Pour finir, vous allez implémenter la méthode RSA.

Les messages transférés seront donc de simples entiers, mais comme nous l'avons vu en cours, tout type d'information peut être codé par une suite de bits, et à fortiori par une suite d'entiers.

a. Diviseurs et nombres premiers

Question 5. Ecrire une fonction `estDiviseur`, telle que `estDiviseur(a,b)` renvoie `True` si `a` est un diviseur de `b`, et `False` sinon.

Question 6. Ecrire une fonction `listeDiviseurs`, telle que `listeDiviseurs(a)` renvoie la liste des diviseurs de l'entier `a`.

Question 7. Ecrire une fonction `estPremier`, telle que `estPremier(n)` renvoie `True` si le nombre `n` est premier, et `False` sinon.

Rappel : Un nombre premier est un entier naturel qui admet exactement deux diviseurs entiers distincts : 1 et lui-même.

Question 8. Ecrire une fonction `sontPremiersEntreEux`, telle que `sontPremiersEntreEux(n,p)` renvoie `True` si les nombres `n` et `p` sont premiers entre eux, et `False` sinon.

Rappel : Deux nombres sont premiers entre eux s'ils admettent exactement un diviseur entier commun : 1.

Question 9. En utilisant les fonctions définies ci-dessus,

a. Déclarer deux variables p et q, correspondant à deux nombres premiers distincts
b. Déclarer une variable n, telle que $n = pq$
c. Déclarer une variable phi telle que $phi = (p-1)(q-1)$
d. Déclarer une variable e, correspondant à un entier premier avec phi et strictement inférieur à phi (vous pouvez utiliser une fonction pour calculer cette valeur à partir de phi)

b. Inverse modulo

On dit que b est l'inverse de a modulo n si et seulement si $a \times b = 1 \mod n$. En d'autres termes, b est l'inverse de a modulo n si et seulement si il existe un nombre entier relatif k tel quel $a \times b + k \times n = 1$

Remarque : Ce entier b n'existe pas toujours, mais s'il existe, il peut être choisi entre 0 et $n-1$.

Question 10. Ecrire une fonction `trouverInverseModulo`, telle que `trouverInverseModulo(a,n)` renvoie, s'il existe, l'inverse de `a` modulo `n`.

Remarque : Si ce nombre n'existe pas, utiliser l'instruction `raise Exception("Texte du message à afficher")` pour le signaler.

Question 11. En utilisant les résultats des questions précédentes

a. Déclarer une variable d, correspondant à l'inverse de e modulo phi (avec $e < phi$)
b. Afficher la clé publique (n, e) et la clé privée (n, d)

c. Exponentiation modulaire

Question 12. Imaginer une fonction `exponentiationModulaire`, telle que `exponentiationModulaire(a,k,n)` renvoie la valeur de a^k modulo `n`

Remarque : $a^k \mod n = a \times \left(a^{k-1} \mod n\right) \mod n$

Question 13. Grâce aux résultats des questions précédentes,

a. Déclarer une variable M, correspondant au message à chiffrer (soit donc un entier strictement inférieur à n)
b. Chiffrer ce message grâce à la clé publique, et créer la variable $C = M^e \ mod \ n$
c. Déchiffrer ce message grâce à la clé privée, et créer la variable $R = C^d \mod n$
d. Vérifier que $R = M$