Université de Caen Normandie, UFR des Sciences
Licence 3
Module : Algorithmique, structures informatiques et cryptologie

Cryptologie : TP

Récupérez le fichier TP à compléter sur la plateforme. Ce fichier est un script « vide » où toutes les fonctions à coder sont présentes mais non encore implémentées. Dans le fichier, vous trouverez en commentaires (entre """ """) des tests pour un certain nombre de fonctions. Vous devez implémenter toutes les fonctions et vous assurer à minima que les tests présents dans le fichier sont satisfaits par vos fonctions. Vous pouvez réalisez le travail par deux, mais dans ce cas veuillez à bien indiquer vos deux au début du fichier. Essayer également de prendre l'habitude de mettre uniquement des noms de variables, fonctions et des commentaires en anglais. Pensez à bien lire les indications données en commentaire dans le corps des fonctions.

1 Cryptographie symétrique

1.1 Chiffrement de César

On commence par implémenter des fonctions utiles pour le chiffrement et déchiffrement de César.

1. Fonction $get\_char\_idx(char : str) \rightarrow int$: fonction qui prend en entrée un caractères *char* et retourne son indice dans la chaine de caractère `CHARACTERS = abcdefghijklmnopqrstuvwxyz0123456789` .

2. Fonction $get\_translation\_table(string1 : str, string2 : str) \rightarrow dict$: fonction qui prend en entrée deux chaines de caractères *string*1 et *string*2 et qui retourne un dictionnaire faisant correspondre chaque caractère de *string*1 à un caractère de *string*2.

3. Fonction $cesear\_enc(plaintext : str, key : str) \rightarrow str$: fonction qui prend en entrée une chaine de caractères *plaintext* et un caractère *key* et qui retourne une chaine de caractère correspondant au chiffrement par décalage de *plaintext* avec la clé *key*.

4. Fonction $cesear\_dec(ciphertext : str, key : str) \rightarrow str$: fonction qui prend en entrée une chaine de caractères *ciphertext* et un caractère *key* et qui retourne une chaine de caractère correspondant au déchiffrement par décalage de *ciphertext* avec la clé *key*.

1.2 Cryptanalyse du chiffrement de César

On implémente ensuite des fonctions pour réaliser la cryptanalyse du chiffrement de César.

1. Fonction $chars\_occurences(string : str) \rightarrow dict$: fonction qui prend en entrée une chaine de caractères *string*, compte le nombre d'occurences de chaque caractère de la chaine et rend le résultat sous la forme d'un dictionnaire.

2. Fonction $chars\_frequency(string : str) \rightarrow dict$: fonction qui prend en entrée une chaine de caractères *string*, calcule la fréquence d'apparition de chaque caractère de la chaine et rend le résultat sous la forme d'un dictionnaire.

3. Fonction $guess\_key(char1 : str, char2 : str) \rightarrow str$: fonction qui prend deux caractères *char*1 et *char*2 tel que *char*2 est le chiffré de *char*1 par le chiffrement de César et la clé inconnue, et retourne un caractère correspondant à la clé.

4. Fonction $cesear\_cryptanalysis(ciphertext : str) \rightarrow dict$: fonction qui prend en entrée une chaine de caractères *ciphertext* et retourne sous la forme d'un dictionnaire le résultat du déchiffrement par les caractères les plus présents dans la langue française.

1.3 Chiffrement de Vigenère

On implemente maintenant les fonctions de chiffrement et de déchiffrement de Vigenère. On limite la taille de la clé à 15 pour simplifier.

1. Fonction $vigenere\_enc(plaintext : str, key : str) \rightarrow str$: fonction qui prend en entrée deux chaines de caractères *plaintext* et *key* et retourne une chaine de caractères correspondant au chiffré de *plaintext* par *key*.

2. Fonction *vigenere_dec*($ciphertext : str, key : str) \rightarrow str$: fonction qui prend en entrée deux chaînes de caractères *ciphertext* et *key* et qui retourne une chaine de caractères correspondant au déchiffrement de *ciphertext* par *key*.

1.4 Cryptanalyse de Vigenère

Enfin on implemente les fonctions pour la cryptanalyse de Vigenère. Ici on utilisera le test de Friedman (avec les indices de coïncidences) pour déterminer la taille de la clé (et non le test de Kasiski comme en TD).

1. Fonction *coincidence_index*($string : str) \rightarrow float$: fonction qui prend en entrée une chaine de caractère *string* et qui un flotant correspondant à l'indice de coïncidence de *string*.

2. Fonction *guess_key_size*($cipertext : str) \rightarrow int$: fonction qui prend en entrée une chaîne de caractères *ciphertext* et qui devine la taille de clé utilisée pour obtenir le chiffré.

3. Fonction *vigenere_guess_key*($ciphertext : str) \rightarrow str$: fonction qui prend en entrée une chaine de caractère *ciphertext* et qui retourne la clé (devinée) qui a été utilisée pour obtenir ce message chiffré.

4. Fonction *vigenere_cryptanalysis*($ciphertext : str) \rightarrow str$: fonction qui prend en entrée une chaine de caractère *ciphertext* et qui retourne le message decrypté.

2 Fonctions arithmétiques

1. Fonction *euclid*(a, b) : fonction qui prend en entrée deux entiers a et b dans $\mathbb{Z}$ et qui retourne le pgcd (positif) des deux entiers en utilisant l'algorithme d'Euclide.

2. Fonction *extended_euclid*(a, b) : fonction qui prend en entrée deux entiers a et b dans $\mathbb{Z}$ et qui retourne le triplet d, u, v (dans cet ordre) avec d le pgcd (positif) de (a, b) et (u, v) des coefficients de Bezout vérifiant $a \times u + b \times v = d$. L'algorithme utilisé est l'algorithme d'Euclide étendu.

3. Fonction *modular_inverse*(a, n) : fonction qui prend en entrée deux entiers a et n dans $\mathbb{Z}$ et qui retourne l'unique inverse modulaire de a modulo n compris entre 0 et $n - 1$. Si a n'est pas inversible modulo n, alors 0 est renvoyé.

4. Fonction *naive_euler_function*(n) : fonction qui prend en entrée un entier positif $n > 1$ et qui retourne l'indicatrice d'Euler $\phi(n)$ en testant un par un tous les entiers compris entre 1 et n.

5. Fonction *euler_function*(L_1, L_2) : fonction qui prend en entrée deux listes $L_1 = [p_1, p_2, \cdots, p_k]$ et $L_2 = [\alpha_1, \alpha_2, \cdots, \alpha_k]$ de même longueur avec L_1 une liste de nombres premiers et L_2 une liste d'entiers strictement positifs. La fonction retourne l'indicatrice d'Euler $\phi(n)$ avec $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$.

6. Fonction *inversibles*(n) : fonction qui prend en entrée un entier positif n et qui retourne la liste de tous les éléments inversibles modulo n (les inverses ne sont pas demandés).

7. Fonction *ordre*(a, n) : fonction qui prend en entrée deux entiers positifs a et n $(a > 1)$ avec $pgcd(a; n) = 1$ et qui retourne l'ordre de a modulo n défini
   comme le plus petit indice k tel que $a^k = 1 \mod n$.

8. Fonction *generateur*(n) : fonction qui prend en entrée un entier n et qui retourne un élément d'ordre $\phi(n)$ si elle en trouve. Si aucun générateur n'est trouvé, 0 est renvoyé.

9. Fonction *naive_exponentiation*(a, k, n) : fonction qui prend en entrée trois entiers a, k et n avec k positif et qui calcule $a^k \mod n$. L'algorithme naif effectue $k - 1$ multiplications modulaires $a \times a \times \cdots \times a \mod n$ (le mod n est fait après chaque multiplication)

10. Fonction *square_and_multiply*(a, k, n) : fonction qui prend en entrée trois entiers a, k et n avec k positif et qui calcule $a^k \mod n$ avec l'algorithme **Square and Multiply** vu en cours. Vous pourrez utiliser la fonction *bin* de python pour la décomposition binaire d'un entier.

11. Fonction *generate_prime*(k, d) : fonction qui prend en entrée deux entiers positifs k et p et qui retourne un nombre premier ayant exactement k bits. L'algorithme pourra se tromper sur la primalité du nombre avec une probabilité inférieure à $1/4^d$. Vous utiliserez pour tester la primalité la fonction *miller_rabin* qui implémente le test de Miller Rabin vu en cours.

3 Protocole R.S.A.

Dans le même fichier, implémentez les fonctions suivantes liées au protocole R.S.A. .

1. Fonction *generate_key*(k) : fonction qui prend en entrée un entier positif et pair k et qui retourne les éléments $[p, q, N = pq, \phi(N), d, e]$ (dans cet ordre) avec (N, e) la clé publique du protocole R.S.A. et $(p, q, \phi(n), d)$ la clé privée associée. p et q devront être des nombres premiers distincts avec exactement $k/2$ bits et N aura exactement k bits. Pour tester la primalité, vous pourrez prendre comme paramètre $d = 40$.

2. Fonction *encipher*(m, N, e) : fonction de chiffrement R.S.A. qui retourne le chiffré du message $m \in (\mathbb{Z}/N\mathbb{Z})^\times$ en utilisant la clé publique (N, e).

3. Fonction *decipher*(c, d, N) : fonction de déchiffrement R.S.A. qui retourne le message clair associé au chiffré $c \in (\mathbb{Z}/N\mathbb{Z})^\times$ en utilisant la clé privée d.