Projet/TP final – Mathématiques pour l'Informatique.

Ce projet est à rendre le 01/05/2023 sous forme d'un ou plusieurs fichiers Python comportant le code produit, ainsi que les réponses aux quelques questions ouvertes du sujet sous forme de commentaire. Toute trace de travail sera récompensée avec bienveillance.

1 Partie 1 : cryptographie RSA

Le chiffrement RSA (nommé par les initiales de ses trois inventeurs Rivest, Shamir et Adleman) est un système de cryptographie asymétrique encoe très utilisé pour échanger des données confidentielles, notamment dans le commerce électronique. Le principe de la cryptographie asymétrique est d'utiliser une paire de clés pour crypter :

- une clé publique, accessible à tous, qui servira à chiffre un message;
- une clé privée, qui sera nécessaire pour déchiffrer mais dont seule la personne qui doit décoder le message aura accès.

Les deux clés sont générées par une même personne, par exemple Alice, qui souhaite que lui soient envoyées des données confidentielles. Alice rend la clé publique accessible. Cette clé est utilisée par ses correspondants pour chiffrer les données qui lui sont envoyées. La clé privée est quant à elle réservée à Alice, et lui permet de déchiffrer ces données. La clé privée peut aussi être utilisée par Alice pour signer une donnée qu'elle envoie, la clé publique permettant à n'importe lequel de ses correspondants de vérifier la signature.

Une condition indispensable est qu'il soit « calculatoirement impossible » de déchiffrer à l'aide de la seule clé publique, en particulier de reconstituer la clé privée à partir de la clé publique, c'est-à-dire que les moyens de calcul disponibles et les méthodes connues au moment de l'échange (et le temps que le secret doit être conservé) ne le permettent pas.

1.1 Principe du RSA

Le crypto-système RSA utilise le calcul de puissances d'entiers modulo le produit de deux grands nombres premiers. Formellement, on considère deux grands nombres premiers p et q, et leur produit $n = p \times q$. Les entiers p et q sont secrets, alors que n est public. On choisit ensuite deux entiers d et e tels que

$$d \times e \equiv 1[(p-1)(q-1)]$$

ou encore

$$d \times e \equiv 1[\varphi(n)]$$

où φ est la fonction indicatrice d'Euler. Le couple (d, n) est la *clé publique*, tandis que l'entier e est la *clé privée*.

Chiffrer un message. Pour chiffre un message a, on va calculer en utilisant l'algorithme de puissance rapide la valeur

$$b := a^d[n]$$

Cette opération nécessite $O(\ln(e)\ln(n)^2)$ opérations.

Déchiffrer un message. Pour déchiffrer un message b, on va calculer en utilisant l'algorithme de puissance rapide la valeur

$$c := b^e[n]$$

Cette opération nécessite $O(\ln(d)\ln(n)^2)$ opérations.

Dans un crypto-système, on s'attend à ce que lorsqu'on déchiffre un message chiffré, on retombe sur le message original, ce qui revient à dire que les fonctions de chiffrement et de déchiffrement sont des fonctions réciproques l'une de l'autre. C'est le cas ici : en effet, la condition $de \equiv 1[\varphi(n)]$ implique qu'il existe un entier $k \in \mathbb{Z}$ tel que $de = 1 + k\varphi(n)$. Alors, si on part d'un message a, que l'on crypte et l'on décrypte, on obtient

$$\begin{aligned}(a^d[n])^e[n] &= a^{de}[n] = a^{1+k\varphi(n)}[n] \\ &= a \times (a^{\varphi(n)})^k[n] \\ &= a[n]\end{aligned}$$

puisque par le petit théorème de Fermat, on rappelle que $a^{\varphi(n)} \equiv 1[n]$.

Exercice 1.

1. Écrire une fonction Python qui permet de vérifier qu'un nombre passé en paramètre est un nombre premier en testant les divisions successives de ce nombre par tous les nombres de 2 à sa racine.
2. Écrire une fonction Python qui permet de générer un nombre premier aléatoire supérieur à une borne passée en paramètre.
3. Écrire une fonction qui permet de calculer l'inverse modulaire d'un nombre passé en paramètre. Par exemple,

   `inverse_mod(5,7)` renvoie `3`

   puisque $(\overline{5})^{-1} = \overline{3}$ dans $\mathbb{Z}/7\mathbb{Z}$.
4. Générer deux nombres premiers aléatoires p et q, et stocker la partie n de la clé publique. Fixer une variable globale d contenant l'autre partie de la clé publique.
5. Écrire une fonction `chiffrer_rsa` qui prend en paramètre un entier n, et qui renvoie la valeur chiffrée à l'aide de la clé (d, n). On utilisera la fonction de calcul d'exponentiation rapide modulaire vue en cours.
6. Écrire une fonction `dechiffrer_rsa` qui prend en paramètre un entier m, qui calcule la partie privée de la clé (ce qui est difficile en général mais faisable dès lors que les nombres premiers ne sont pas trop grands), puis qui puis qui déchiffre le message m à l'aide de la clé privée.

Application : Un professeur P décide d'envoyer ses notes par mail au secrétariat S de l'Université en utilisant un codage RSA. La clé publique de chiffrement est $(d = 3, n = 33 = 3 \times 11)$.

7. Quel message chiffré correspond à la note 13 ?

8. Vérifier que la clé privée de déchiffrement est 7.

9. Si S reçoit le message chiffré "9", à quelle note cela correspond ?

10. Supposons désormais que la clé publique de chiffrement soit $(c = 3, n = 55)$. Quelle est alors la clé privée de déchiffrement ?

2 Partie 2 : Chiffrement affine

Une manière classique de chiffrer un message est d'appliquer une *transformation affine modulaire* sur chaque lettre. Pour ce faire, on va représenter l'alphabet par l'ensemble

$$\mathcal{A} := \{0, 1, \ldots, 25\}$$

avec $A \leftrightarrow 0$, $B \leftrightarrow 1$, $C \leftrightarrow 2$, etc. L'idée est d'appliquer sur chaque lettre une fonction

$$\begin{array}{rccc} f_{a,b\cdot} & : & \mathcal{A} & \to & \mathcal{A} \\ & & x & \mapsto & ax + b[26] \end{array}$$

Par exemple, prenons $a = 3$ et $b = 11$, alors

$$f(0) = 3 \times 0 + 11[26] = 11, \quad f(1) = 3 \times 1 + 11[26] = 14[26], \quad f(2) = 3 \times 2 + 11[26] = 17.$$

Ainsi, la lettre A est envoyée sur la lettre L (indice 11) dans l'alphabet, B est envoyée sur la lettre O (indice 14), C est envoyée sur la lettre R (indice 17), etc. Voici la correspondance obtenue sur tout l'alphabet par cette transformation :

Lettre de départ	Indice de départ	Indice d'arrivée	Lettre d'arrivée
A	0	11	L
B	1	14	O
C	2	17	R
D	3	20	U
E	4	23	X
F	5	0	A
G	6	3	D
H	7	6	G
I	8	9	J
J	9	12	M
K	10	15	P
L	11	18	S
M	12	21	V
N	13	24	Y
O	14	1	B
P	15	4	E
Q	16	7	H
R	17	10	K
S	18	13	N
T	19	16	Q
U	20	19	T
V	21	22	W
W	22	25	Z
X	23	2	C
Y	24	5	F
Z	25	8	I

Par exemple, le mot "Bonjour" sera ainsi transformé par ce système de chiffrement en "Obymbtk". On remarque que la colonne d'arrivée contient toutes les lettres de l'alphabet, et une et une seule fois. Ceci correspond au fait que la fonction de chiffrement est *bijective*, c'est-à-dire que chaque élément de l'ensemble d'arrivée est associé à un et un seul élément d'ensemble de départ.

Globalement, une fonction de la forme $f_{a,b}$ est bijective si et seulement si $\overline{a}$ est inversible dans $\mathbb{Z}/26\mathbb{Z}$, autrement dit **si et seulement si a est premier avec** 26, ce qui est le cas pour 3. Dans ce cas, la fonction $f_{a,b}$ peut bien être utilisée comme fonction de chiffrement. On parle de chiffrement affine sur $\mathbb{Z}/26\mathbb{Z}$. Le couple (a, b) est appelé *clé de chiffrement*.

Notez qu'on considère souvent l'ensemble $\{0, \dots, 28\}$ plutôt que $\{0, \dots, 25\}$ en ajoutant le symbole espace ␣, associé à 0, le symbole virgule associé à 27 et le point associé au 28. Ainsi, puisque 29 est premier, n'importe quel choix de couple (a, b) donnera un clé de chiffrement.

Lorsque la fonction $f_{a,b}$ est bijective, sa fonction réciproque est donnée par

$$(f_{a,b})^{-1} \ : \ x \ \mapsto a^{-1}(x-b)[27]$$

où a^{-1} désigne l'inverse modulaire de $\overline{a}$ dans $\mathbb{Z}/n\mathbb{Z}$, où n est la longueur de l'alphabet considéré, ici 29.

Exercice 2. On prend $n = 29$ et on code l'alphabet de la façon suivante :

$$\text{␣} \leftrightarrow 0, \quad A \leftrightarrow 1, \quad B \leftrightarrow 2, \quad \dots \quad Z \leftrightarrow 26, \quad \text{virgule} \leftrightarrow 27, \quad \cdot \leftrightarrow 28$$

1. Écrire une fonction `chiffrer_aff` qui prend en paramètres deux entiers a et b (la clé de chiffrement), et une chaîne de caractère, et qui renvoie le texte dont chaque lettre a été chiffrée individuellement.

2. En utilisant la clé $(4, 21)$, coder la phrase : « la clef est dans le coffre ».

3. Écrire une fonction `dechiffrer_aff` qui prend en paramètres deux entiers a et b (la clé de chiffrement), et une chaîne de caractère, et qui renvoie la chaîne originale, c'est-à-dire le texte qui a permis d'obtenir la chaîne en paramètre après chiffrement.

4. Déchiffrer le message «v.vlukyu,fwtfyooyn.ws» obtenu à partir de la clé $(4, 21)$.

Remarque : Afin de déchiffrer un message dont on ne connaît pas la clé de chiffrement, on essaye souvent de passer par une analyse de fréquence des lettres dans notre texte, en lien avec les fréquences d'apparition de lettres dans la langue française. Cette fréquence indique que le E est la lettre apparaissant le plus dans la langue française : on peut donc supposer que le lettre associée au E est la lettre la plus présente dans le texte. Par exemple, si la lettre la plus présente est la C, on a une information stipulant que E est envoyé sur C, d'où

$$f_{a,b}(5) = 3 \quad \Leftrightarrow a \times 5 + b = 3[27].$$

Ceci n'est bien sûr pas suffisant pour trouver a et b, mais si on arrive à trouver une autre lettre (par exemple l'espace, aussi assez fréquent dans un texte, ou le A...), on peut alors réussir à retrouver a et b !

Exercice 3.

1. Écrire une fonction Python qui détermine les fréquences d'apparition des lettres d'un texte passé en paramètre dans une liste.

2. Écrire une fonction qui prend en paramètre une chaîne de caractères puis qui, à partir de la liste des fréquences d'apparition des caractères dans cette chaîne, associe E et ␣ aux deux caractères les plus fréquents. Ceci donne deux possibilités de clé (a_1, b_1), (a_2, b_2), afficher les textes déchiffrés à l'aide de ces deux clés.

3. Déterminer la clé qui a servi à obtenir le message codé

« akdyne.vxnkbijdju.dfodjoujhrajdcjd.jyboigfjudgfid␣jnhj..jo.dcjdybiqqnjndj.dc
jdcjybiqqnjndfodhjuukxjsdcvo.dajdyvo.jofdojdcvi.dj.njdyvoofdgfjdcjduvodj,
jci.jfndj.dcjduvodcju.iok.kinjz »

4. Chiffrer le message « vive la programmation » par une clé choisie par vos soins. Pouvez-vous utiliser la technique précédente pour déchiffrer le message obtenu sans utiliser la clé ? Pourquoi ?

3 Partie 3 : Cryptographie de Hill

Le chiffrement de Hill est une extension du modèle de chiffrement affine de la Section 2. Au lieu de coder caractère par caractère, on va les regrouper par paquets de taille fixée p, et coder chaque bloc de taille p à l'aide d'une transformation

$$X' = AX + B[27]$$

où A est une *matrice* de taille $p \times p$, B est un vecteur de p valeurs entières, X est un vecteur contenant les p caractères à chiffrer représentés en colonne, X' est le vecteur correspondant aux p caractères chiffrés. En pratique, on considérera $p = 2$ et on regroupera les données à chiffrer par paquets de 2 éléments. On rappelle qu'une matrice 2×2 est un tableau de 4 éléments disposés selon 2 lignes et 2 colonnes comme suit :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

où a, b, c et d sont des nombres réels. Nous représenterons nos matrices par une liste de deux listes à deux éléments en Python. On peut additionner des matrices coefficient par coefficient (voir ci-dessous), et la formule pour multiplier des matrices est donnée ci-dessous :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} + \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} = \begin{pmatrix} a+a' & b+b' \\ c+c' & d+d' \end{pmatrix}, \qquad \begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} = \begin{pmatrix} aa'+bc' & ab'+bd' \\ ca'+dc' & cb'+dd' \end{pmatrix}$$

On peut aussi multiplier une matrice par un vecteur à 2 éléments via :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax+by \\ cx+dy \end{pmatrix}$$

Exemple : On code les lettres de A à Z par les nombres de 1 à 26, l'espace ␣par 0, la virgule par 27 et le point · par 28. On va donc raisonner modulo 29, puisque l'ensemble des caractères considéré possède 29 éléments. Pour coder, on va utiliser la matrice et le vecteur

$$A = \begin{pmatrix} 5 & 7 \\ 1 & 15 \end{pmatrix}[29], \qquad B = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Ainsi, pour coder la paire de caractères « EX » représentée par le vecteur $\begin{pmatrix}5\\24\end{pmatrix}$ (car E est codé par 5, X est codé par 24), on va calculer le résultat de

$$\begin{aligned}\begin{pmatrix}5 & 7\\1 & 15\end{pmatrix}\begin{pmatrix}5\\24\end{pmatrix}+\begin{pmatrix}1\\2\end{pmatrix}[29] &= \begin{pmatrix}5\times 5+7\times 24\\1\times 5+15\times 24\end{pmatrix}+\begin{pmatrix}1\\2\end{pmatrix}[29]\\ &= \begin{pmatrix}194\\367\end{pmatrix}[29]\\ &= \begin{pmatrix}20\\19\end{pmatrix}[29]\end{aligned}$$

Le vecteur $\begin{pmatrix}20\\19\end{pmatrix}$ représente la paire de caractères « TS ».

Exercice 4.

1. Écrire en Python des fonctions qui permettent :
   - de multiplier deux matrices 2×2 entre elles,
   - de multiplier une matrice par un vecteur de taille 2,
   - de multiplier tous les coefficients d'une matrice par un nombre x passé en paramètre,
   - de faire la somme de deux vecteurs de taille 2.
2. Écrire une fonction `decouper_texte` qui prend en paramètre une chaîne de caractères, et qui renvoie le texte découpé sous forme d'une liste de listes de deux caractères. On commencera à découper par la droite, de sorte que si la longueur du texte est impaire, le premier caractère se retrouvera groupé avec un espace `_`.
3. Écrire une fonction `rassembler_texte` qui permet de faire le regroupement des lettres dans l'autre sens.
4. Écrire une fonction `chiffrer_hill` qui prend en paramètre une chaîne de caractère, une matrice A et un vecteur B de taille 2 (qui représentent la clé) et qui permet d'afficher le texte chiffré en utilisant la clé (A, B).
5. En reprenant le même codage de l'alphabet et la même clé de chiffrement que dans l'exemple ci-dessus, coder le message «vive les vacances ».

La fonction $F_{A,B} : X \mapsto AX+B$ est bijective sous une certaine condition sur la matrice A, qui est que cette matrice est inversible modulo 29 (la taille de l'alphabet). Pour une matrice 2×2, il suffit de calculer son déterminant, et vérifier que ce n'est pas un multiple de 29. La transformation permettant de décrypter est alors donnée par :

$$(f_{A,B})^{-1} \ : \ Y \mapsto A^{-1}(Y-B)[29]$$

où A^{-1} est l'inverse de la matrice A. On rappelle que pour calculer l'inverse d'une matrice $A=\begin{pmatrix}a & b\\c & d\end{pmatrix}$, on a

$$\det\left(\begin{pmatrix}a & b\\c & d\end{pmatrix}\right)=ad-bc, \qquad A^{-1}=\frac{1}{\det(A)}\begin{pmatrix}d & -b\\-c & a\end{pmatrix}$$

Cette formule est également valable lorsque l'on raisonne modulo un nombre entier, avec la seule différence que le nombre $\frac{1}{\det(A)}$ représente l'inverse modulaire de $\det(A)$ modulo ce nombre. D'où

$$A^{-1} = (\det(A))^{-1} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \text{ [29]}$$

Exercice 5.

1. Écrire une fonction qui détermine si une matrice donnée est inversible modulo 29 (la taille de l'alphabet), et si oui, qui calcule son inverse modulaire.

2. Écrire une fonction `dechiffrer_hill` qui prend en paramètre une chaîne de caractère, une matrice A et un vecteur B de taille 2 (qui représentent la clé) et qui permet d'afficher le texte déchiffré, c'est à dire le texte qui a permis d'obtenir la chaîne en paramètre après chiffrement.

3. **(Bonus – plus difficile)** Généralement, j'utilise pour crypter mes messages une matrice générée aléatoirement et un vecteur $B = (0, 0)$. Par ailleurs, je termine tous mes messages en signant «␣ bd.» (4 caractères) Écrire une fonction qui permet de retrouver la matrice permettant de déchiffrer le message.

   Indication : Ceci indique que le couple $(0, 2)$ (respectivement $(4, 28)$) correspondant respectivement aux paires «␣b» et «d.» sont envoyés sur les couples correspondant aux 4 dernières lettres de notre texte. Ceci donne un système d'équations à partir des quelles on peut retrouver sans problème les 4 coefficients de la matrice...