TD 1 : reprise en main de Python, méthodes de chiffrement

Les sujets de TD ne sont ni relevés, ni notés. Ils sont volontairement conçus pour ne pas pouvoir être traités en une seule séance ; n'hésitez cependant pas, si vous le souhaitez, à tenter de les finir chez vous.

L'objectif de ce TP est d'implémenter deux algorithmes capables de chiffrement et de déchiffrement de messages. Quelques conseils pratiques :

— Respectez les conventions de style de Python : la PEP 8 et le *Zen of Python* (à chercher sur Google). Prenez dès maintenant ces habitudes.

— Écrivez des tests, c'est-à-dire des fonctions non explicitement demandées dans le sujet vous permettant de tester qu'une de vos implémentations fonctionne. Par exemple, si vous disposez d'un algorithme de chiffrement et d'un algorithme de chiffrement, écrivez une fonction tentant de chiffrer et déchiffrer quelques messages et renvoyant `True` si tout se passe bien. Ceci est *vital* : ainsi, si vous apportez une modification mineure et incorrecte à l'une de vos fonctions, vous pourrez tout de suite trouver quel est le problème.

— Ne vous souciez pas de l'interface dans un premier temps. Écrivez des fonctions à appeler depuis le *shell* Python. Si vous le souhaitez, vous pourrez ultérieurement mettre en place une interface un peu plus *user friendly*.

— **N'utilisez pas les fonctions de ce TD pour échanger des données autres que celles de test.** De façon générale, n'utilisez jamais votre implémentation personnelle d'un algorithme connu : les failles de sécurité potentielles sont multiples, et seul une méthode de chiffrement dont le code est public et ayant fait l'objet d'audits complets présente des garanties suffisantes pour l'échange de données personnelles.

Quelques notions d'arithmétique

Deux nombres entiers x et y sont *congrus modulo* n (avec n entier naturel non nul) si leur différence $x-y$ est un multiple de n. Pour tout nombre x, il existe un unique entier x' compris entre 0 et $n-1$ tel que x et x' soient congrus modulo n : il s'agit du reste de la division euclidienne de x par n. En mathématiques, on utilise la notation $x \mod n$ pour désigner le nombre x'. En Python, on obtient x' en tapant `x % n`.

Dans ce TD, nous travaillerons toujours avec $n = 64$. Raisonner *modulo 64*, c'est ne plus considérer les nombres comme des points sur une droite infinie, mais sur un cercle, une horloge à soixante-quatre positions. Modulo 64, l'on compte : 0, 1, 2, ..., 62, 63, 0, 1, etc.

Les opérations modulo n sont grandement facilitées par le fait que pour tous entiers x et y, on a :

Les propriétés d'associativité, de commutativité et de distributivité vraies dans $\mathbf{Z}$ restent donc valables modulo n.

— $(x+y) \mod n = ((x \mod n) + (y \mod n)) \mod n$;

— $(x-y) \mod n = ((x \mod n) - (y \mod n)) \mod n$;

— $(x \times y) \mod n = ((x \mod n) \times (y \mod n)) \mod n$;

— $x^y \mod n = (x \mod n)^y \mod n$. Attention, il est faux de prétendre que cela est égal à $(x \mod n)^{y \bmod n}$ dans le cas général !

Valentin Melot 2BCPST3
TD 1 15 et 22 septembre 2017

Questions préliminaires

1. Assurez-vous *impérativement* que vous utilisez Python 3. Pour cela, entrez la commande :

   ```
   print "blah"
   ```

   Vous *devez* obtenir une erreur. Si aucune erreur n'apparaît et si le texte « blah » apparaît à l'écran, cela signifie que vous utilisez une version obsolète de Python. Changez-en.

2. Dans ce TP, l'on se limitera à des messages utilisant les lettres de l'alphabet français (incluant les diacritiques) en capitales, les chiffres arabes, l'espace (représentée par la suite par le caractère ␣) et les signes typographiques essentiels que sont le trait d'union, l'apostrophe, la virgule, le point, le point-virgule, le deux-points, le point d'interrogation, le point d'exclamation et les points de suspension. Ces caractères seront codés par les nombres suivants :

A		Z	À	Â	Æ	Ç	É	È	Ê	Ë	Î	Ï	Ô	Œ	Ù	Û
0		25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
Ü	Ÿ	0		9	␣	-	’	,	.	:	:	?	!	…	(	)
40	41	42		51	52	53	54	55	56	57	58	59	60	61	62	63

   En s'aidant du code mis en ligne, écrire deux fonctions `char2int` et `int2char` convertissant l'un des caractères de la liste précédente en nombre entier et réciproquement. Une exception devra être soulevée si le paramètre en entrée est incorrect.

3. Écrire deux fonctions convertissant respectivement une chaîne en liste d'entiers et une liste d'entiers en chaîne de caractères selon la table de correspondance précédente. *On pourra utiliser librement le constructeur `list(s)` qui sépare une chaîne `s` en la liste de ses caractères, et la fonction membre `".join(l)`, qui réunit une liste `l` de chaînes en une seule.*

Chiffrement de Vigénère

Le chiffrement de Vigénère est un chiffrement symétrique. Cela signifie qu'avant de s'échanger des messages, l'expéditeur et le destinataire doivent avoir déterminé une clef consistant en une suite de caractères, qui servira à chiffrer et à déchiffrer. Son principe est le suivant :

— Le message originel, de longueur n, est converti en une liste d'entiers de longueur n.

— La clef k est convertie en une liste d'entiers de longueur ℓ, puis répétée et éventuellement tronquée pour obtenir une liste de longueur n également.

— Les entiers des deux listes sont additionnés deux à deux modulo 64, et la liste obtenue est retransformée en une chaîne de caractères.

Par exemple, pour la clef « `BCPST` » et le message « `J’ÉTUDIE SÉRIEUSEMENT.` », le chiffrement est fait de la façon suivante :

Message :	J	’	É	T	U	D	I	E		S	É	R	I	E	U	S
	9	54	30	19	20	3	8	4	52	18	30	17	8	4	20	18
Clef :	B	C	P	S	T	B	C	P	S	T	B	C	P	S	T	B
	1	2	15	18	19	1	2	15	18	19	1	2	15	18	19	1
	10	56	45	37	39	4	10	19	6	37	31	19	23	22	39	19
Résultat :	K	.	3	Œ	Ù	E	K	T	G	Œ	È	T	X	W	Ù	T

Soit un message chiffré valant : `K.3ŒÙEKTGŒÈTXWÙTGÂWÊU;.`.

4. Écrire une fonction qui, à partir d'une clef k de longueur ℓ quelconque et d'un entier n, produit la liste de longueur n correspondante en répétant les caractères de la clef autant de fois que nécessaire.

 Remarques et corrections bienvenues à l'adresse <valentin@melot.tf>.

5. Écrire une fonction qui, à partir de deux listes d'entiers de même longueur, produit la liste des sommes deux à deux modulo 64 des éléments de la liste. L'on soulèvera une exception si les deux listes n'ont pas la même longueur.
6. Écrire une fonction prenant comme paramètre un message et une clef, et chiffrant le message selon la méthode de Vigénère.
7. De même, écrire une fonction capable de déchiffrer un message en connaissant sa clef. L'on utilisera le fait que, $(x + y) - y = x$ reste vrai modulo 64.

Chiffrement RSA

Contrairement au chiffrement de Vigénère, le chiffrement RSA est asymétrique : le destinataire des messages génère une clef privée (qu'il garde secrète) et une clef publique, connue de chaque personne voulant lui écrire. C'est la clef publique qui sera utilisée pour chiffrer, et la clef privée pour déchiffrer. Intuitivement, le destinataire génère des cadenas que chaque personne peut utiliser pour mettre un message dans une boîte, mais seul le destinataire peut ouvrir ce cadenas.

Il repose sur une propriété simple des nombres premiers : il est très simple de multiplier deux nombres entiers, mais très difficile de décomposer un nombre entier en produit de facteurs premiers.

8. Nous aurons besoin pour ce problème de calculer des valeurs de la forme $M^e \mod n$ avec M, e et n particulièrement grands (une vingtaine de chiffres). Constater que l'instruction naïve suivante est beaucoup trop lente :

   ```
   M**e % n
   ```

   L'on propose de mettre en place l'algorithme suivant, récursif :

   — Si `e` vaut 0, le résultat recherché est 1.
   — Si `e` est non nul et pair, calculer $r = M^{n/2} \mod n$; le résultat recherché est $r^2 \mod n$.
   — Si `e` est non nul et impair, calculer $r = M^{(n-1)/2} \mod n$; résultat recherché est $M \times r^2 \mod n$.

   L'implémenter sous la forme d'une fonction Python et constater sa rapidité. Question bonus pour les ex-spé maths : pourquoi cela fonctionne-t-il ?

   En fait, la fonction `pow` de Python fait exactement ce que l'on cherche. En cas de difficultés à faire cette question, n'hésitez pas à passer à la suite avec la fonction `pow`.
9. Le destinataire des messages doit générer deux nombres entiers p et q de grande taille, qui doivent rester secrets. Pour ce TD, vous pouvez vous aider de l'outil en ligne accessible à l'adresse suivante : `<http://fr.numberempire.com/primenumbers.php>`. Choisissez deux nombres premiers de dix chiffres au moins chacun.

   On pose $z = (p-1)(q-1)$. Il est nécessaire de trouver un entier e inférieur strictement à z qui soit premier avec z. L'on propose pour cela de tirer au hasard des nombres entiers dans l'intervalle $[\![2, z-1]\!]$ jusqu'à en trouver un qui convienne. On rappelle qu'il suffit pour cela d'utiliser la fonction `random.randint`. La clef publique est le couple (n, e), elle peut être communiquée à toute personne qui le souhaite. Implémenter une fonction générant la clef publique selon la méthode proposée.

   Pour calculer le PGCD de deux nombres, vous pouvez au choix utiliser la fonction de bibliothèque `math.gcd` ou implémenter vous-même le classique algorithme d'Euclide.
10. Pour chiffrer un message, il est nécessaire pour le destinataire de le découper en blocs de taille au plus n. Avec l'alphabet choisi, un paquet de dix caractères peut être représenté par un nombre compris entre 0 et $2^{60} - 1$, ce dernier nombre étant de l'ordre de $1,15 \times 10^{18}$, donc bien inférieur à $p \times q$. Écrire une fonction `cut_msg`, qui, à partir d'un message renvoie une liste de nombres compris entre 0 et $2^{60} - 1$ correspondant au message initial. On prendra soin de compléter le message initial avec des espaces pour que sa longueur soit un multiple de 10. Écrire également la fonction réciproque.

11. Si le message découpé est la liste `[M_1, ..., M_k]`, le message chiffré est la liste `[M_1**e % n, ..., M_k**e % n]`. Écrire une fonction `rsa_crypt` qui chiffre un message en connaissant la clef publique.

12. Pour la suite, nous aurons besoin de résoudre le problème du calcul de l'inverse modulaire : étant donnés deux entiers a et n premiers entre eux, déterminer un entier b tel que $a \times b \mod n = 1$. Ceci est possible car d'après le théorème de Bézout, il existe u et v entiers tels que $au + nv = 1$, et $b = u$ convient. L'on propose une modification du célèbre algorithme d'Euclide pour calculer u et v :

    — L'on pose $r = a$, $r' = b$, $u = v' = 1$ et $u' = v = 0$.

    — Tant que r' est non nul, l'on pose q le quotient de la division euclidienne de r par r' et :

    $$(r, u, v, r', u', v') \text{ devient } (r', u', v', r - qr', u - qu', v - qv')$$

    — L'algorithme se termine lorsque $r' = 0$.

    Implémenter cet algorithme.

13. La clef privée, qui sert à déchiffrer, est le couple (n, d) où d est l'inverse de e modulo $z = (p-1)(p-1)$. Le déchiffrement repose sur le constat suivant, qui est une conséquence du petit théorème de Fermat : si M est un entier et $N = M^e \mod n$, alors $M = N^d \mod n$. Implémenter une fonction qui déchiffre un message en connaissant la clef privée.

    Cette méthode de chiffrement est plus lente, mais contrairement aux chiffrements symétriques comme celui de Vigénère, il n'est pas nécessaire pour l'expéditeur et le destinataire de se mettre d'accord auparavant sur le choix d'une clef. Les protocoles de communication sécurisée modernes tels que SSH utilisent les deux types d'algorithmes : l'un des utilisateurs génère une clef publique pour RSA qu'il envoie à l'autre, ce dernier génère une clef symétrique qu'il chiffre avec RSA et envoie au premier, et les deux peuvent alors communiquer à l'aide de l'algorithme de chiffrement symétrique puisqu'ils sont en possession de la clef.

14. Nous avons été obligés de rajouter artificiellement des espaces pour grouper les caractères. Comment peut-on remédier à ce désagrément ?

15. Il reste maintenant à générer des nombres premiers de grande taille de façon aléatoire. Une façon de procéder est la suivante : choisir un intervalle dans lequel nous désirons trouver un nombre premier (par exemple $[\![10^9, 10^{10}]\!]$), et y tirer des nombres aléatoirement jusqu'à trouver un premier. Mettre en place une fonction générant deux nombres premiers distincts de cette façon. L'on commencera par un algorithme naïf de recherche de nombres premiers que l'on pourra chercher à améliorer.