Algorithme de Chiffrement et de cryptanalyse

TP

Cryptologie

Objet de l'étude

La cryptographie peut se rencontrer sous forme de différents algorithmes.

Il s'agit d'en implanter quelques-uns en Python pour mieux les appréhender.

Outils utilisés

- De quoi prendre des notes personnelles : voir les détails dans les sections suivantes.
- Un ordinateur personnel (de bureau).
- L'environnement de programmation EduPython.

Organisation

- Penser à prendre des notes personnelles en tenant compte des indications suivantes…
- Conserver une copie des documents fournis pour ce TP (via Moodle) et des documents numériques qui auront été produits durant la séance (fichiers personnel, scripts, etc.).
- Conserver les notes informatiques produites (éditeur, documents textes, documents graphiques, etc.), en les enregistrant, soit dans un dossier de votre espace de stockage personnel (lecteur H) associé à votre compte utilisateur au lycée (dénommé « compte Kwartz »), soit sur la zone de stockage des documents personnels de votre espace Moodle.
- Gardez aussi toutes ses productions sur une clef de stockage (ou sur un espace de stockage en ligne fiable) pour un travail personnel hors séance ou en cas d'impossibilité d'accéder aux données du serveur. Cette pratique de la sauvegarde de données est très importante.
- Poursuivre les travaux en dehors de la séance en travaillant à son rythme.

Documents de référence ou complémentaires

- L'utilisation des diaporamas de cours donnés en ressource sur Moodle est à la fois nécessaire et particulièrement encouragée.
- L'aide de l'EDI EduPython, accessible directement dans son interface.
- Les documents du site EduPython peuvent apporter des compléments ou une aide précieuse : Cliquer ici, ainsi que d'autres, à voir sur le site.

Conventions typographiques

Les variables et le texte codé (scripts) sont notés en caractères `courier`.

Le symbole 🛠 indique il faut effectuer une manipulation dans le logiciel ou dans l'environnement.

I. Introduction

De tout temps l'être humain a cherché des moyens de protéger la confidentialité et l'authenticité de ses communications. Il y a généralement trois acteurs : un expéditeur, un destinataire, et un intercepteur potentiel sur le chemin. La cryptographie consiste à rendre le message inintelligible à toute personne autre que le destinataire. La cryptanalyse consiste à tenter de déchiffrer un message sans avoir les informations données au destinataire.

Le plus ancien document chiffré est une recette secrète de poterie qui date du XVIe siècle av. J.-C., qui a été découverte en Mésopotamie (dans l'actuelle Irak). La plus ancienne analyse cryptographique connue est une description de l'analyse fréquentielle par Al-Kindi au IXe siècle.

Le chiffrement d'un message peut être un jeu — par exemple la célèbre correspondance entre George Sand et Alfred de Musset, qui relève plutôt du message caché — ou une affaire d'État : chiffre de Marie Stuart, ou Enigma[1] pendant la seconde guerre mondiale. Plus récemment les affaires de violation de la vie privée révélées par Snowden, mettent en jeu des questions de cryptographie.

II. Chiffre de César

De nos jours, Jules César est le premier personnage connu ayant utilisé le chiffrement par décalage. Selon Suétone, César utilisait un décalage de 3 lettres pour sa correspondance secrète, en particulier militaire.

Soit n un entier relatif. On souhaite écrire un programme qui code un mot en décalant chaque lettre de l'alphabet de n lettres. Par exemple pour n = +3, le décalage sera le suivant :

Alphabet clair		**a**	**b**	**c**	**d**	**…**	**x**	**y**	**z**
Alphabet crypté		**D**	**E**	**F**	**G**	**…**	**A**	**B**	**C**

Le mot prepatsi devient ainsi SUHSDWVL

Q.1 🛠 Analyser et tester les lignes de code suivante:

```
alp_clair = [chr(i) for i in range(97,123)]
alp_crypt = [chr(i) for i in range(65,91)]
```

la commande inverse de chr() est ord()

Q.2 🛠 Écrire le code de la fonction suivante :

```
def cesar(n : int, texte : str) -> str
```

qui prend un entier n et une chaîne de caractères texte, et retourne la chaîne codée par la méthode décrite ci-dessus. On ignorera (en les recopiant) les caractères qui ne sont pas des minuscules non accentuées chaque caractère étant calculé en utilisant les fonctions chr() et ord().

Q.3 🛠 Écrire le code de la fonction suivante :

```
def cesar2(n : int, texte : str) -> str
```

Qui fonctionne comme la précédente mais qui utilise la méthode index et les listes définies à la question 1

La méthode .lower() permet de passer un texte quelconque en minuscule.

Q.4 🛠 La « clé n » étant connue, comment mettre en œuvre les fonctions précédentes pour décoder un texte crypté ? Tester et valider cet usage des fonctions

La cryptanalyse consiste à casser le code. Le texte qui suit a été codé par le chiffre de César :

« GFLITIRTBVILEKVOKVGRIWFITVSILKVZCWRLKKVJKVIKFLKVJCVJGFJJZSZCZKVVJ »

[1] On fera attention à ne pas prendre pour vérité historique le dernier film sur le sujet

Q.5 🛠 Combien y a-t-il de clés de décalage possibles ? Proposer une méthode de résolution, et casser le code. Il est vivement conseillé de faire une boucle.

III. Autre chiffrement mono-alphabétique

Q.6 🛠 La fonction de numpy random.randint(a,b) tire un entier au hasard (selon une loi uniforme) dans range(a,b) À l'aide de cette fonction, coder une fonction :

```
def tirage(n : int) -> lst
```

qui prend en argument un entier *n* et qui retourne une liste de *n* entiers tirés au hasard dans l'intervalle [0,n-1], **sans répétition**. Utiliser une boucle while.

Q.7 Combien y a-t-il de telles listes pour un *n* donné ?

Q.8 🛠 créer une fonction :

```
def creer_cle () -> lst
```

qui fournit une clé sous forme d'une liste de caractère majuscule de 26 caractères ordonnés de façon aléatoire. Réutiliser pour cela tirage et chr().

Exemple de résultat : ['K', 'L', 'U', 'X', 'F', 'J', 'E', 'V', 'I', 'R', 'G', 'S', 'Y', 'P', 'C', 'D', 'W', 'B', 'A', 'M', 'Z', 'T', 'H', 'N', 'O', 'Q']

Q.9 🛠 créer une fonction :

```
def chiffrer (cle : lst , texte : str) -> str
```

qui crypte ici de telle sorte que a -> K ; b -> L etc.

Il est possible pour cela d'utiliser alp_clair et la méthode .index() vus précédemment.

Q.10 🛠 Mettre en œuvre le décodage lorsque l'on connait la clé ; écrire la fonction :

```
def decrypt (cle : lst , texte : str) -> str
```

Q.11 Peut-on casser le chiffrage par force brute, par une méthode analogue à celle utilisée pour casser le code de César ? En supposant qu'il suffise d'une milliseconde pour tester chaque possibilité, combien faut-il de jours (ou d'années) pour tester toutes les possibilités.

IV. Analyse fréquentielle

Les fréquences d'apparition des différentes lettres ne sont pas les mêmes, dans un texte en français suffisamment long. Par exemple, le « e » est la lettre (en moyenne[2]) la plus fréquente.

Si le texte a été crypté par permutation monoalphabétique, la lettre la plus fréquente sera probablement celle qui correspond au « e ».

Q.12 🛠 Écrire une fonction

```
def frequences (texte : str) -> lst
```

qui retourne une liste des fréquences des 26 lettres dans le texte fournit en argument. texte est en majuscules

Q.13 🛠 tracé de l'histogramme

Inclure le code suivant :

[2] Sauf dans *la disparition*

```
def trace(freq):
    x   = np.array(range(26))
   w = 0.3
   plt.bar(x,freq,w,0)
    plt.xticks(x,alp_crypt)
   plt.show()
```

La fréquence des lettres en français est donnée ici :

[9.42, 1.02, 2.64, 3.39, 15.87, 0.95, 1.04, 0.77, 8.41, 0.89, 0.00, 5.34, 3.24, 7.15, 5.14, 2.86, 1.06, 6.46, 7.90, 7.26, 6.24, 2.15, 0.00, 0.30, 0.24, 0.32]

Q.14 🛠Affichez sur une même figure les deux listes de fréquences. Conjecturer quelle lettre correspond au « e ».

Q.15 Pour aller plus loin :

Les lettres n'ont pas toutes les mêmes voisines, avec les mêmes fréquences. Par exemple les lettres doublées (bi-gramme) sont, par fréquence décroissante : e, s, l, t, n, m, r, p, f, c (et d'autres plus rares). Écrire une fonction qui classe par fréquence décroissante les lettres doubles du texte

Écrire un programme qui • affiche les histogrammes de la question 14 et le texte crypté, • demande à l'utilisateur (input()) de rentrer une lettre chiffrée et sa traduction en clair. • effectue le remplacement dans le texte (avec les vérifications ad hoc). • recommence au premier point. Grâce à ce programme (et aux lettres doublées), craquer le code du texte.

V. Chiffre de Vigenère

Nous venons de voir que les chiffrements par substitution sont vulnérables à l'analyse fréquentielle. Le chiffrement de Vigenère fonctionne sur le principe suivant : on dispose d'une clé qui est une chaîne de caractères (en général un mot), et on décale la première lettre du texte d'un rang correspondant à celui de la première lettre de la clé dans l'alphabet et ainsi de suite. Par exemple, si la clé est « python », la première lettre sera décalée de 15 (p est la 15ème lettre de l'alphabet, en partant de 0), le deuxième de 24 etc. On revient au début de la clé si le texte à crypter est plus long que la clé).

Ce chiffrement, décrit au XVIe siècle, a été considéré comme incassable de sa popularisation au XVIIe siècle jusqu'au milieu du XIXe siècle.

Exemple :

texte clair	u	n	e	x	e	m	p	l	e	d	e	c	h	i	f	f	r	e	m	e	n	t
clé	p	y	t	h	o	n	p	y	t	h	o	n	p	y	t	h	o	n	p	y	t	h
décalage	15	24	19	7	14	13	15	24	19	7	14	13	15	24	19	7	14	13	15	24	19	7
texte crypté	J	L	X	E	S	Z	E	J	X	K	S	P	W	G	Y	M	F	R	B	C	G	A

Q.16 🛠Écrire une fonction :

```
def chiffre_vig (cle , texte : str) -> str
```

prenant comme argument une clé cle et le texte à crypter texte et effectuant le chiffrement.

Q.17 🛠Comment déchiffrer le texte en connaissant la clé ? écrire la fonction :

```
def decrypt_vig (cle , texte : str) -> str
```

Analyse :

Grâce à la clé, une même lettre peut être codée par différentes lettres dans le texte chiffré Donc ce code n'est pas vulnérable à une analyse fréquentielle simple.

Q.18 En fonction de la longueur N de la clé, donner le nombre de tests à effectuer par une méthode « force brute ». Que constate-t-on ?.

VI. Chiffrement RSA

Le chiffrement RSA est un algorithme de cryptographie asymétrique [WIKI06], très utilisé dans le commerce électronique, et plus généralement pour échanger des données confidentielles sur Internet. Cet algorithme a été décrit en 1977 par Ronald Rivest, Adi Shamir et Leonard Adleman, ce qui, par la reprise des initiales des noms, donne la dénomination « RSA ». Cet algorithme a été breveté aux États-Unis par le Massachusetts Institute of Technology (MIT) en 1983. Le brevet ayant expiré le 21 septembre 2000, il fait désormais partie du « domaine public ».

Avant d'étudier les différentes fonctions Python permettant le chiffrement et le déchiffrement d'un message, quelques informations sur la cryptographie vont être apportées.

Éléments sur le chiffrement

Le chiffrement d'un message est un procédé de cryptographie qui assure que la compréhension d'un document est rendue impossible à toute personne qui n'a pas l'information permettant le déchiffrement, appelée clé de déchiffrement.

Beaucoup de méthodes assurent le chiffrement/déchiffrement plus ou moins efficace, et avec des complexités variables, qui permet la confidentialité des informations transmises. C'est souvent le nombre de cas à traiter pour découvrir la teneur du message crypté qui permet de garantir la confidentialité du message : plus le traitement est long, plus le décrypteur illégitime renonce à son entreprise.

Pour crypter un message, deux types de chiffrement sont employés :

- Le chiffrement symétrique quand la même clé est utilisée pour chiffrer et déchiffrer un message. Dans ces conditions la découverte de la clé de cryptage permet d'accéder plus aisément au message.
- Le chiffrement asymétrique emploie deux clés différentes. La première est la clé publique, utilisée pour le chiffrement, et l'autre est la clé privée, tenue secrète pour mener à bien les opérations de déchiffrement. Cette organisation publique/privée est d'autant plus efficace qu'elle utilise de grands nombres aux propriétés particulières pour garantir au mieux l'impossibilité calculatoire de déduire la clé privée de la clé publique.

Une des méthodes la plus connue est le RSA pour le chiffrement asymétrique.

Description de l'algorithme RSA

Tous les calculs se font modulo un nombre entier n qui est le produit de deux nombres premiers. Le petit théorème de Fermat joue un rôle important dans la conception du chiffrement.

Les messages clairs et chiffrés sont des entiers inférieurs à l'entier n. Les opérations de chiffrement et de déchiffrement consistent à élever le message à une certaine puissance modulo n (c'est l'opération d'exponentiation modulaire).

La seule description des principes mathématiques sur lesquels repose l'algorithme RSA n'est pas suffisante. Sa mise en œuvre concrète demande de tenir compte d'autres questions qui sont essentielles pour la sécurité. Par exemple le couple (clé privée, clé publique) doit être engendré par un procédé vraiment aléatoire qui, même s'il est connu, ne permet pas de reconstituer la clé privée. Les données chiffrées ne doivent pas être trop courtes, pour que le déchiffrement demande vraiment un calcul modulaire, et complétées de façon convenable.

Eléments des algorithmes utilisés

Le principe :

Le récepteur utilise sa clé privée pour décrypter c'est lui qui a fabriqué les clés il a envoyé sa clé publique en clair à son interlocuteur, il est le seul à pouvoir décoder le message car il connait la clé privée. Pour crypter un message l'émetteur à juste besoin de la clé publique de son destinataire. Il existe des annuaires de clés publiques où l'on peut déposer sa clé publique

Le renouvellement des clés n'intervient que si la clé privée est compromise, ou par précaution au bout d'un certain temps (qui peut se compter en années).

Création des clés

a) Choisir p et q, deux nombres premiers distincts ; (normalement il faut de très grands nombre premiers)

b) calculer leur produit n = pq, appelé module de chiffrement ;

c) calculer φ(n) = (p - 1)(q - 1) (c'est la valeur de l'indicatrice d'Euler en n) ;

d) choisir un entier naturel e premier avec φ(n) et strictement inférieur à φ(n), appelé exposant de chiffrement ;

e) calculer l'entier naturel d, inverse de e modulo φ(n), et strictement inférieur à φ(n), appelé exposant de déchiffrement ; d peut se calculer efficacement par l'algorithme d'Euclide étendu.

Comme e est premier avec φ(n), d'après le théorème de Bachet-Bézout il existe deux entiers d et k tels que ed = 1 + kφ(n), c'est-à-dire que ed ≡ 1 (mod φ(n)) : e est bien inversible modulo φ(n).

Le couple (n, e) est la clé publique du chiffrement, alors que sa clé privée est définie comme le couple (d, n)

Q.19 Analyser la fonction euclide(a,b) qui renvoie le PGCD de a et b.

Deux nombres étant premiers entre eux lorsque leur PGCD est égal à 1

Q.20 Analyser choix_e(phi) qui recherche un nombre e < φ(n) premier avec φ(n). Noter que φ(n) est un très grand nombre. Le principe est de choisir des nombres au hasard compris entre 1 et φ(n) jusqu'à en trouver un premier avec φ(n) et le fournir en retour de cette fonction. Cette fonction sera utilisée à l'étape d)

Le script de la fonction définie ci-dessous permet de déterminer l'élément de la clé privée : c'est à dire : l'entier naturel d, inverse de e modulo φ(n), et strictement inférieur à φ(n)

```
def euclide_etendu(a,b):
    """Entrée : a, b entiers (naturels) Sortie : r entier (naturel)
       et u, v entiers relatifs tels que r = pgcd(a, b) et r = a*u+b*v
    u est l'inverse de a modulo b"""
    r , r1 , u , v , u1 , v1 = a, b, 1 , 0 ,0,  1

    while r1 != 0 :
       q ,rs,us,vs,r,u,v = r//r1,r,u,v,r1,u1,v1
       r1 = rs - q*r1
       u1 = us - q*u1
       v1 = vs - q*v1
       return r,u,v
```

Pour analyser le fonctionnement de l'algorithme on se place dans un cas trivial ou p=11 et q = 5

Q.21 Déterminer la valeur de n et de φ(n) dans ce cas.

Q.22 montrer que e = 33 convient pour répondre à la problématique du d)

Q.23 écrire la ligne de programme permettant de déterminer d en utilisant les variables de l'énoncé : d, e , phi (représentant φ(n))

Q.24 écrire le tableau d'avancement des variables temporaires : `q, r, r1, u, v, u1, v1 us, vs lorsque l'on exécute x,y,z = euclide_etendu(33,40)` jusqu'à la sortie de la boucle ? Quelle est alors la valeur *d* qui constitue la clé privée ?

Chiffrement du message

Si M est un entier naturel strictement inférieur à n représentant un message, alors le message chiffré sera représenté par

$$C \equiv M^e \ (mod\ n)$$

l'entier naturel C étant choisi strictement inférieur à n.

Déchiffrement du message

Pour déchiffrer C, on utilise d, l'inverse de e modulo φ(n), et l'on retrouve le message clair M par

$$M \equiv C^d \ (mod\ n)$$

Construction des fonctions de cryptage et de décryptage

Les clés sont fabriquées grâce à la fonction suivante :

```
def cree_cles(p,q) :
    """p,q sont deux nombres premiers"""
    n=p*q
    phi=(p-1)*(q-1)
    d=-1
    r=0
    while d <0 :
            e = choix_e(phi)
            #e et phi sont premiers entre eux
            r,d,v = euclide_etendu(e,phi)
            # si d < 0 c'est que e ne convient pas
    cle_pub =  e , n
    cle_priv = d , n
    return cle_pub , cle_priv
```

Cette fonction utilisée une fois pour toute par le destinataire renvoie un tuple de tuple

Q.25 Ecrire en python la fonction **crypt (M , Key)** qui renvoie le nombre crypté et la fonction **def decrypt(C , Key)** qui effectue le décryptage. (Utiliser la fonction pow() décrite en annexe)

Q. .26 Le message M est un entier qui doit être inferieur à n. Ecrire une fonction qui calcule le nombre d'octet que peut contenir le Message M **nboctets (n)**

Pour la suite on prend

p= 900000015247

q=pow(2,521)-1

p et q sont deux nombres premiers (connus uniquement du destinataire)

Q.27 Quel est l'intérêt de choisir de grands nombres premiers ?

Q.28 Quelle est la longueur du message pouvant être crypté avec ces deux nombres ?