UNIVERSITE DE CAEN
IUT Caen
Département Réseaux et Télécommunications

TP cryptographie R408 – TP 1

***

Exercice 0 Créez un répertoire $TPcrypto\_ < votre\_nom >$ c'est dossier que vous rendre zippé sur sur ecampus. Il contiendra tout vos fichiers python, les fichiers décryptés et les réponses aux questions...cryptées.

Exercice 1 Vous allez coder l'algorithme d'euclide qui permet de calculer le pgcd entre deux nombres. Créez une fonction $getPGCD(nb1, nb2)$

- Elle prend en argument deux entiers, $nb1$ et $nb2$
- Elle effectue les calculs des des division euclidiennes successives.
- Elle affiche les calculs.
- Si $nb1 == nb2$ alors elle retourne nb1
- Si nb1 ou nb2 vaut 1 elle retourne 1
- Si nb1 ou nb2 est négatif alors elle lève une erreur.
- Elle retourne le PGCD entre les deux nombres

```
>> print(PGCD(16, 24))
8
>> print(PGCD(17, 24))
1
```

Exercice 2 Le code César est un code qui fonctionne grâce à la substitution. Dans premier temps vous coderez le chiffre puis vous verrez les deux manières de le craquer.

1. Créez la fonction $decal\_lettre(lettre, x)$:
   (a) Elle prend en argument un caractère $lettre$, et un int x
   (b) Elle calcul le caractère voulu en décalant de x le caractère mis en entrée
   (c) Elle gère le dépassement, c'est dire que le décalage de 'z' de 1 doit donner 'a' avec un calcul de modulo "%".
   (d) Elle conserve la casse (une majuscule doit rester une majuscule).
   (e) Elle garde la ponctuation.

(f) Elle retourne le caractère modifié.

(g) Vous ne traiterez pas les accents.

```
>> print(decal_lettre("a", 3))
d

>> print(decal_lettre("z", 3))
c
>> print(decal_lettre("z", 26))
z

>> print(decal_lettre("E", 5))
J
```

2. Créez la fonction $chiffre\_Caesar(message, decalage, chiffre)$. Un pangramme est un phrase qui contient toutes les lettres de l'alphabet. Vous testerez l'exécution de cette fonction à l'aide des pangrammes proposés dans le fichier $test_phrase.txt$.

   (a) Elle prend en argument une string $message$, un int x, un booléen $chiffre$.

   (b) Si $chiffre$ est vrai elle appelle la fonction $decal\_lettre(lettre, x)$ pour chiffrer en ajoutant x à chaque caractère de la string.

   (c) Sinon, elle utilise la fonction $decal\_lettre(lettre, x)$ pour déchiffrer en soustrayant x à chaque caractère de la string.

   (d) Enfin, elle retourne la string chiffrée ou déchiffrée.

```
>> code = "Utwyje hj anjzc bmnxpd fz ozlj gqtsi vzn kzrj"
>> (chiffre_cesar(code, 5, chiffre= False))
"Portez ce vieux whisky au juge blond qui fume"

>> message = "Portez ce vieux whisky au juge blond qui fume"
>> print(chiffre_cesar(message, 23))
"Mloqbw zb sfbru tefphv xr grdb yilka nrf crjb"
```

3. La manière la plus simple de casser le code César est de testé toutes les clefs possible. Facile, il n'y en a que 25. Créez la fonction $bruteForce\_Caesar(message)$:

   (a) Elle prend en entrée une string $message$.

   (b) Elle affiche les 25 possibilités de déchiffrement

```
>>  craquer_ceasar_BF("Kpzfvy, jwsf, gbujhvf, mf ofa rvj qjrvf,
   Dmpxo Ibsz tljf ebot m'pncsf")
Joyeux, ivre, fatigue, le nez qui pique, Clown Hary skie dans l'
   ombre
Inxdtw, huqd, ezshftd, kd mdy pth ohptd, Bknvm Gzqx rjhd czmr k'
   nlaqd
```

```
Hmwcsv, gtpc, dyrgesc, jc lcx osg ngosc, Ajmul Fypw qigc bylq j'
    mkzpc
Glvbru, fsob, cxqfdrb, ib kbw nrf mfnrb, Ziltk Exov phfb axkp i'
    ljyob
Fkuaqt, erna, bwpecqa, ha jav mqe lemqa, Yhksj Dwnu ogea zwjo h'
    kixna
Ejtzps, dqmz, avodbpz, gz izu lpd kdlpz, Xgjri Cvmt nfdz yvin g'
    jhwmz
Disyor, cply, zuncaoy, fy hyt koc jckoy, Wfiqh Buls mecy xuhm f'
    igvly
Chrxnq, bokx, ytmbznx, ex gxs jnb ibjnx, Vehpg Atkr ldbx wtgl e'
    hfukx
Bgqwmp, anjw, xslaymw, dw fwr ima haimw, Udgof Zsjq kcaw vsfk d'
    getjw
Afpvlo, zmiv, wrkzxlv, cv evq hlz gzhlv, Tcfne Yrip jbzv urej c'
    fdsiv
Zeoukn, ylhu, vqjywku, bu dup gky fygku, Sbemd Xqho iayu tqdi b'
    ecrhu
Ydntjm, xkgt, upixvjt, at cto fjx exfjt, Radlc Wpgn hzxt spch a'
    dbqgt
Xcmsil, wjfs, tohwuis, zs bsn eiw dweis, Qzckb Vofm gyws robg z'
    capfs
Wblrhk, vier, sngvthr, yr arm dhv cvdhr, Pybja Unel fxvr qnaf y'
    bzoer
Vakqgj, uhdq, rmfusgq, xq zql cgu bucgq, Oxaiz Tmdk ewuq pmze x'
    ayndq
Uzjpfi, tgcp, qletrfp, wp ypk bft atbfp, Nwzhy Slcj dvtp olyd w'
    zxmcp
Tyioeh, sfbo, pkdsqeo, vo xoj aes zsaeo, Mvygx Rkbi cuso nkxc v'
    ywlbo
Sxhndg, rean, ojcrpdn, un wni zdr yrzdn, Luxfw Qjah btrn mjwb u'
    xvkan
Rwgmcf, qdzm, nibqocm, tm vmh ycq xqycm, Ktwev Pizg asqm liva t'
    wujzm
Qvflbe, pcyl, mhapnbl, sl ulg xbp wpxbl, Jsvdu Ohyf zrpl khuz s'
    vtiyl
Puekad, obxk, lgzomak, rk tkf wao vowak, Iruct Ngxe yqok jgty r'
    ushxk
Otdjzc, nawj, kfynlzj, qj sje vzn unvzj, Hqtbs Mfwd xpnj ifsx q'
    trgwj
Nsciyb, mzvi, jexmkyi, pi rid uym tmuyi, Gpsar Levc womi herw p'
    sqfvi
Mrbhxa, lyuh, idwljxh, oh qhc txl sltxh, Forzq Kdub vnlh gdqv o'
    rpeuh
Lqagwz, kxtg, hcvkiwg, ng pgb swk rkswg, Enqyp Jcta umkg fcpu n'
    qodtg
```

4. Une autre manière d'attaquer le code César est d'utiliser l'apparition statistique des

lettres. En français on sait que le E est la lettre le plus fréquente. On récupère la clef de décalage en calculant l'écart entre la lettre la plus fréquente du message codé et E. Pour tester votre fonction vous utiliserez le fichier "$Caesar\_crypted.txt$" fourni en annexe. Le fichier décrypté sera à rendre à la fin du TP avec les autre fichiers produits. Créez la fonction $attaque\_stat\_Caesar(filename)$:

(a) Elle prend en argument une string $filename$, le nom du fichier.

(b) Elle calcule le caractère le plus fréquent du fichier "filename.txt". Elle le compare à "e" pour extraire la clef.

(c) Elle produit un nouveau fichier texte "¡filename¿_¡clef¿_decrypted.txt" avec le message déchiffré.

(d) Elle ne renvoie rien

5. Vous rendrez un fichier texte $reponse\_Caesar\_ < votre_clef >.txt$ chiffré le code César répondant aux questions suivantes:

   - Qui est l'auteur du texte "$Caesar\_crypted.txt$" ?
   - Que représente l'objet découvert par le protagoniste de l'histoire ?

Exercice 3 Le code de Vigenère agit comme un code de César, mais tous les caractères ne sont pas décalés de la même valeur. Les décalages utilisés dépendent d'une clef, en général donnée par un mot ou une phrase.

Nous avons vu au premier cours que le chiffre de Vigenère était sensible au attaques fréquentielle par utilisation du test de Kasiski ou l'indice de coïncidence. Aujourd'hui nous allons plutôt utiliser une méthode plus ancienne, la méthode de Charles Babbage.

Vous utiliserez la fonction $decal\_lettre(lettre, x)$ de l'exercice précédent.

1. Créez la fonction $vigenere(message, clef, chiffre)$

   (a) Elle prend en argument une string $message$, une string $clef$ et un booléen $chiffre$

   (b) Elle transforme $clef$ en liste de nombre

   (c) Elle duplique la clef pour que la longueur de la clef corresponde à la longueur de $message$.

   (d) Si $chiffre$ est vrai elle utilise la fonction $decal\_lettre(lettre, x)$ pour chiffrer $message$. Sinon, elle utilise la fonction $decal\_lettre(lettre, x)$ pour déchiffrer $message$.

```
>> print(vigenere("Portez ce vieux whisky au juge blond qui fume
   ", "Crypto", chiffre = True))
Rscvka gp bjgyi cikwva bw uwmf fwqte ufk gwqp
>> print(vigenere("Isevyu vi xczlq jjcnbr nw elzi dfjew dwc wnqr
   ", "graphie", chiffre = False))
Portez ce vieux whisky au juge blond qui fume
```

2. La lettre la plus fréquente en français est la lettre E. Nous l'avons vu, cette information permet facilement de casser le code de César en calculant le décalage entre la lettre la plus fréquente du message codé et 'E' . Mais cette même méthode ne marchera pas pour casser le code de Vigenère qui est un peu plus solide. Nous allons contourner cet obstacle en étudiant la fréquence des groupes de trois lettres (méthode Babbage).

   Principe : Un groupe de trois lettres consécutives a toutes les chances, à chaque fois qu'il apparaît dans le message chiffré, d'être la conséquence du chiffrement des mêmes lettres du message avec les mêmes lettres de la clef. Pour un groupe de quatre lettres, c'est encore plus probable. L'espacement entre deux mêmes groupes de lettres chiffrées est un multiple de la longueur de la clef.

   Exemple: Si la répétition d'un groupe est espacée de 30 lettres, puis celle d'un autre de 25, le plus grand diviseur commun de 25 et 30 est 5. La clef possède donc dans ce cas 5 lettres.

   Vous validerez votre fonction avec le fichier "Vigenere_crypted.txt" fourni avec les documents du cours. Le fichier décrypté sera à rendre à la fin du TP avec les autre fichiers produits. Vous testerez l'exécution de cette fonction à l'aide des pangrammes proposés dans le fichier $test_phrase.txt$.

   Une fois que la longueur de la clef est récupérée il suffit de regrouper les lettres par selon l'intervalle donnée.
   Exemple :

E	V	I	D	E	M	M	E	N	E
B	C	B	C	B	C	B	C	B	C
F	X	J	F	F	O	N	G	O	G

   On aura le premier sous set composé de F,J,F,N,O.
   Le deuxième sous set sera composé de X,F,O,G,G.

   Dans le premier set c'est la lettre F qui est la plus fréquente, l'écart avec E est de 2, ce qui correspond à la lettre B.
   Dans le deuxième set c'est la lettre G qui est la plus fréquente, l'écart avec E est de 3, ce qui correspond à lettre C.
   Le mot de chiffre est bien "BC", il ne reste plus qu'à appliquer le décalage sur le chiffre pour retrouver le texte d'origine.

   (a) Utilisez la fonction pgcd développée dans l'Exercice 1

   (b) Créez la fonction $longueurClef(message, taille\_motif)$ qui va déterminer la longueur de la clef

   - Elle prend en argument une string $message$ et une string $taille\_motif$ initialisé à 3.

- Elle parcourt le message pour recenser toutes les motifs de taille $taille\_motif$
- Elle collecte toutes les distances entre deux occurrences du même motif de $taille\_motif$ lettres.
- Elle calcul la longueur probable de la clef à l'aide de la fonction $getPGCD$ si besoin.
- Si la longueur est supérieur à 5, le résultats à des chances d'être bon, la fonction retourne la longueur probable de la clef
- Sinon on relance l'algorithme avec une taille de motif plus grand. (attention au condition d'arrêt)

(c) Créez la fonction $getClef(message, taille_clef)$ qui détermine la clef du message.

- Elle prend en argument une string $message$ et un entier $taille_clef$
- Elle regroupe les lettres du message codé avec la même lettre (voir exemple) en sous set
- Pour chaque sous set elle détermine la lettre la plus fréquente qui correspond en clair à la lettre "e" et en déduit la lettre du correspondante du chiffre
- Elle renvoie la une string $clef$

(d) Créez la fonction $attaque\_Babagge(filename)$

- Elle prend en argument une string $filename$, le nom du fichier.
- Elle calcule la taille de la clef avec $longueurClef(message, mot)$
- Elle détermine la clef avec $getClef(message, l)$
- Elle utilise $vigenere(message, clef, chiffre)$ pour décoder le texte.
- Elle produit un nouveau fichier texte "¡filename¿_¡clef¿_decrypted.txt" avec le message déchiffré.
- Elle ne renvoie rien

3. Vous rendrez un fichier txt $reponse\_Vigenere\_ < votre_clef > .txt$ chiffré avec le code Vigenère répondant aux questions suivantes:

   - De qui parle le texte ? (vous pouvez cherchez sur Wikipédia la page de cette personne pour être plus à l'aise pour les questions suivantes)
   - Dans quel domaine cette personne a été la précurseure ?
   - Avec qui a-t-elle notamment travaillé ?