Crypto : chiffrement affine

Exercice 1 (0 points)

On donne le code ci-dessous :

```
def inverse(a, n):
    i = 1
    while i < n:
        if a*i % n == 1:
            return i
        i += 1
    return 0
```

sort de la fonction (donc de la boucle).

?

a) Combien de fois la boucle `while` figurant dans le code ci-dessus va-t-elle s'exécuter lors de l'appel à la fonction `inverse(5, 8)` ?

b) Et pour l'appel `inverse(2, 6)` ?

c) Et encore pour l'appel `inverse(25, 26)` ?

d) Pourquoi le mot-clef `return` est-il utilisé à deux reprises ?

e) Quel est l'intérêt d'avoir placé le mot-clef `return` dans la boucle while ?

Justifier les deux réponses.

d) La 1ère fois : pour renvoyer l'inverse à l'utilisateur.

La 2ème fois : renvoyer 0 pour signifier qu'il n'y a pas d'inverse.

a) 1 2 3 4 5 ← i

5 | 10 | 15 | 20 | 25 | 30 | 35 | 40

$25 \equiv 1 \mod 8$

La boucle tourne 5 fois.

b) 1 2 3 4 5

2 | 4 | 6 | 8 | 10

Pas d'inverse

Même réponse : 5 fois.

c) $25 \cdot 25 = 625$

25 | 50 | — | — | 625

1 25

625	26
520	24
105	
104	
1	

La boucle est exécutée 25 fois.

e) Pour interrompre le processus dès que l'inverse a été trouvé. (On veut optimiser un peu.)

Exercice 2 (0 points)

On considère la fonction définie ci-dessous :

```
def affine(message, clef):
    a = clef[0]
    b = clef[1]
    chiffre = ""
    i = 0
    while i < len(message):
        c = message[i]
        z = ord(c) - 65
        z = (a*z + b) % 26
        c = chr(z + 65)
        chiffre += c
        i += 1
    return chiffre
```

a) Écrire l'appel à cette fonction qui permet de chiffrer le message `ZUT`.

b) Donner le chiffre correspondant.

c) Combien de fois la boucle aura-t-elle tourné ?

avec la clef (5; 12)

sous forme de texte.

3 fois

a) affine("ZUT"; (5, 12)) appel à la fonction

b) HID

2 6

Z → 25 → 5·25 + 12 ≡ 7 mod 26 → H

T → 19 → 5·19 + 12 ≡ 3 mod 26 → D

Exercice 3 (0 points)

On a chiffré un message m avec le chiffrement affine. Écrire une fonction qui permet de retrouver le message à partir du chiffre et de la clef de chiffrement. On pourra utiliser la fonction `inverse` de l'exercice 1 sans la reprogrammer.

```
def decode_affine(chiffre, clef):
    a = clef[0]
    b = clef[1]
    a_ = inverse(a, 26)
    message = ""
    i = 0
    while i < len(chiffre):
        c = chiffre[i]
        z = ord(c) - 65
        z = a_ * (z - b) % 26
        c = chr(z) + 65
        message += c

        i += 1
    return message
```

Exercice 4 (0 points)

On a chiffré un message avec le code de César. Écrire le code nécessaire pour casser ce chiffre.

On pourra utiliser les fonctions inverse et dech_affine

```
chiffre = "XYZWTQPS"   <- donné
i = 1
while i < 26:
    print(dech_affine(chiffre, (1, i)))
    i += 1
```

Cela permet de trouver visuellement la ligne qui est une phrase en français.

Exercice 5 (0 points)

On a chiffré un message avec le chiffrement affine. Écrire le code nécessaire pour casser ce chiffre.

dech_affine peut être utilisée.

```
chiffre = "ABCDXYZ"
i = 1
while i < 26:
    j = 0
    while j < 26:
        if i != 13:
            print(dech_affine(chiffre, (i, j)))
        j += 1
    i += 2
```