

Thème : Types de bases - CORRECTION

18

Cryptage et décryptage d'un message

1 Application au cryptage d'un message

Cryptage

Transformation d'un message en clair en un message codé
compréhensible seulement par qui dispose du code

L'informatique permet d'automatiser le procédé de cryptage et décryptage d'un message, dans ce notebbok vous allez simuler le fonctionnement d'un logiciel de mail qui crypte/décrypte les messages.

Deux personnes 'Alice' et 'Bob' vont utiliser ce procédé pour communiquer.

Cher agent 'EVE', votre mission sera découvrir le message envoyé par Alice à Bob qui a été intercepté en écoutant le réseau. Nos cryptographes n'ont pas réussi, nous comptons sur vous ! Voici le message :

...

0100110101111010011010010111100101111010011101010010110101110001011010101

...

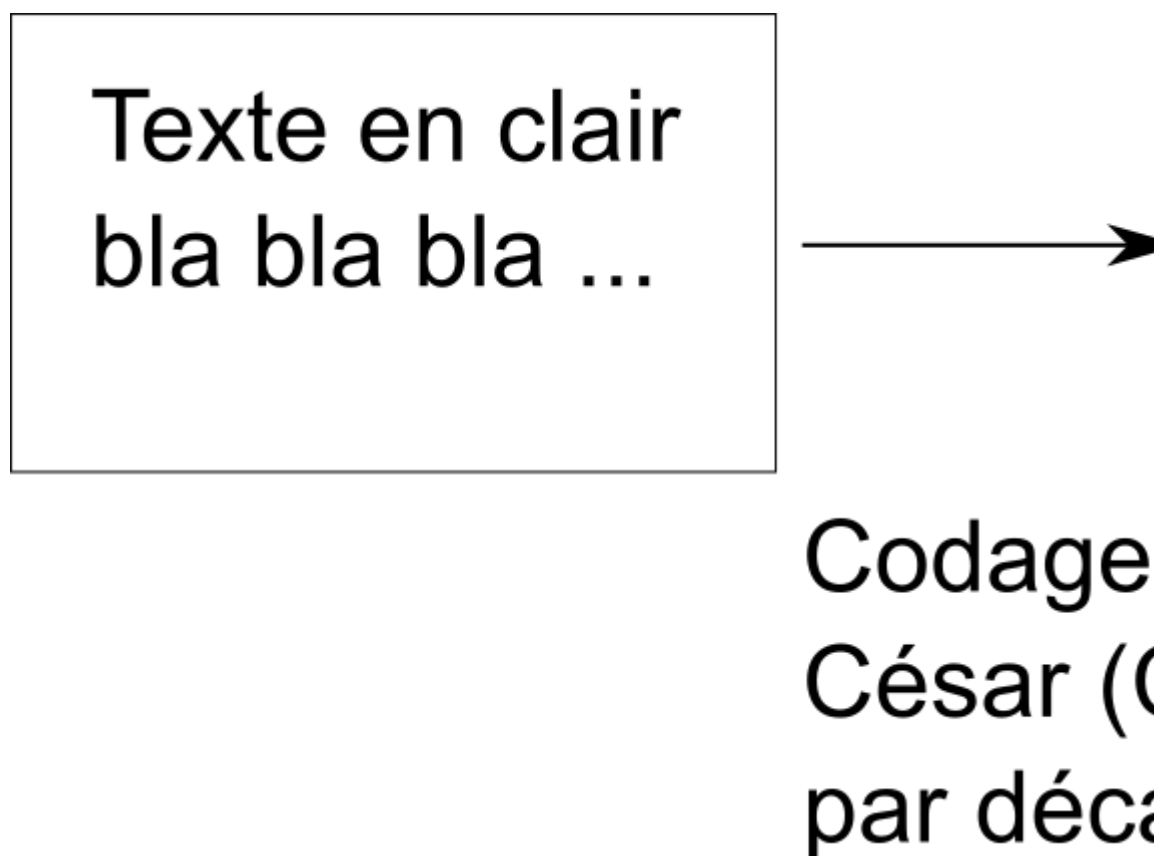
Nos équipes ont mis la main sur la documentation de leur logiciel de messagerie, à vous d'en comprendre le fonctionnement pour décrypter leur message.

1.1 Principe de fonctionnement.

Le texte en clair sera :

- chiffré par Alice en utilisant le code de César, c'est un chiffrement par décalage des lettres. Le texte chiffré s'obtient en remplaçant chaque lettre du texte clair original par une lettre à distance fixe, toujours du même côté, dans l'ordre de l'alphabet. Pour les dernières lettres (dans le cas d'un décalage à droite), on reprend au début. Par exemple avec un décalage de 3 vers la droite, A est remplacé par D, B devient E, et ainsi jusqu'à W qui devient Z, puis X devient A etc...
- chaque caractère sera ensuite converti en binaire sur un octet (8 bits).

C'est ce message binaire qui sera envoyé à Bob qui devra le décoder, Bob connaît la clé qu'Alice a utilisé pour son chiffrement.



1.2 Chiffrement de César.

[Wikipédia : Chiffrement de César](#)

Fonction `ord` :

Renvoie le nombre entier représentant le code Unicode du caractère représenté par la chaîne donnée. Par exemple, `ord('a')` renvoie le nombre entier 97 et `ord('€')` (symbole Euro) renvoie 8364. Il s'agit de l'inverse de `chr()`.

Fonction `chr()` :

Renvoie la chaîne représentant un caractère dont le code de caractère Unicode est le nombre entier `i`. Par exemple, `chr(97)` renvoie la chaîne de caractères 'a', tandis que `chr(8364)` renvoie '€'. Il s'agit de l'inverse de `ord()`. L'intervalle valide pour cet argument est de 0 à 1114111 (0x10FFFF en base 16). Une exception `ValueError` sera levée si `i` est en dehors de l'intervalle.

```
In [4]: def chiffrement_caractere(carac,cle):
        """
        Chiffre un caractère par la méthode de César, les lettres
        les autres caractères ne sont pas modifiés (accents, tirets, etc.)
        Entrées :
            carac (str) un caractère
            cle (int) la clé de codage (classiquement entre 0 et 25)
        Sortie :
            (str) Le caractère décalé par la méthode de César
        """
        if 65 <= ord(carac) <= 90:      # Pour les majuscules
            numero = ord(carac)
            numero = numero - 65 # on se ramène entre 0 et 26 qui est la longueur de l'alphabet
            numero = numero + cle # on effectue le décalage
            numero = numero % 26 # on revient au début lorsqu'on dépasse la fin de l'alphabet
            numero = numero + 65 # on retourne dans la table ASCII
            carac_code = chr(numero)
        elif 97 <= ord(carac) <= 122:   # pour les minuscules
            carac_code = chr((ord(carac) - 97 + cle) % 26 + 97) # même principe que pour les majuscules
        else :
            carac_code = carac
```

```

        carac_code = carac    # on ne change pas les autres ca
    return carac_code

```

```

In [5]: # Tests, si pas d'erreurs continuez.
assert chiffrement_caractere('A',3) == 'D'
assert chiffrement_caractere('T',5) == 'Y'
assert chiffrement_caractere('w',7) == 'd'
# Gestion accent ...
assert chiffrement_caractere('é',10) == 'é'
assert chiffrement_caractere('ç',15) == 'ç'

```

```

In [6]: def chiffrement_mot(mot,cle):
        """
        Chiffre un mot (chaîne de caractères sans espace) par la
        Entrées :
            mot (str)
            cle (int) la clé de codage
        Sortie :
            (str) Le mot décalé par la méthode de césar
        """
        mot_code = ''
        for lettre in mot:
            mot_code += chiffrement_caractere(lettre,cle)
        return mot_code

```

```

In [7]: # Tests, si pas d'erreurs continuez.

assert chiffrement_mot('Bonjour',7) == 'Ivuqvby'
assert chiffrement_mot('Décodage',16) == 'Tésetqwu'
assert chiffrement_mot('Bonjour',33) == 'Ivuqvby'
assert chiffrement_mot('Anticonstitutionnellement',4) == 'Er>

```

```

In [8]: def decoupage(texte):
        """
        Découpe un texte suivant ses espaces
        Entrée : texte (str)
        Sortie : (lst) une liste de str constitué des mots du tex
        """
        texte_decoupe = texte.split()
        return texte_decoupe

```

```

In [9]: # Tests, si pas d'erreurs continuez.
assert decoupage('Ceci est un test.') == ['Ceci', 'est', 'un', 'test']
assert decoupage(' Un texte avec des espaces !!') == ['Un', 'texte', 'avec', 'des', 'espaces', '!!']

```

```

In [10]: def cesar(texte,cle):
        """

```

```

Chiffre un texte par la méthode de césar. Le texte ne co
Entrées :
    texte (str)
    cle (int) la clé de codage
Sortie :
    (str) Le texte décalé par la méthode de césar
"""
texte_decoupe = decoupage(texte)
texte_code = ''
for mot in texte_decoupe:
    texte_code += chiffrement_mot(mot,cle)+' '
# Enlève le dernier espace.
texte_code = texte_code[:-1]
return texte_code

```

```

In [217]: # Tests, si pas d'erreurs continuez.
assert cesar('Bonjour',7) == 'Ivuqvyb'
assert cesar("WIKIPEDIA ENCYCLOPEDIE LIBRE",9) == 'FRTRYNMF
assert cesar("Le texte chiffré s'obtient en remplaçant cha

```

1.3 Conversion en binaire

Compléter les fonction de conversion entre base 2 et base 10.

```

In [24]: def dec_vers_bin(nombre):
    """
    Convertir un nombre entier naturel vers la base 2 sans p
    Entrées :
        nombre (int) : entier naturel
    Sortie :
        (str) Le nombre écrit en base 2
    """
    s = ''
    while nombre > 0:
        # Attention, concaténer à gauche
        s = str(nombre % 2) + s
        nombre //= 2
    return s

```

```

In [ ]: def dec_vers_bin2(chaine_binaire):
    rep=0
    n=len(chaine_binaire)-1
    for i in chaine_binaire:
        rep+=int(i)*2**n

```

```

    n-=1
    return rep

```

```

In [22]: def dec_vers_bin3(nombre):
          return format(nombre, '08b')
          nbe = randint(3,1000)

          dec_vers_bin(nbe)

```

Out[22]: '111010011'

```

In [25]: from random import randint
          # Tests, si pas d'erreurs continuez.
          assert dec_vers_bin(1) == '1'
          assert dec_vers_bin(2) == '10'
          assert dec_vers_bin(1) == '1'
          assert dec_vers_bin(17) == '10001'
          # test aléatoire
          nbe = randint(3,1000)
          assert dec_vers_bin(nbe) == bin(nbe)[2:]

```

```

In [30]: def bin_vers_dec(binaire):
          """
          Convertir un nombre binaire vers la base 10 sans passer
          Entrées :
              binaire (str) : un nombre binaire
          Sortie :
              (int) Le nombre écrit en base 10
          """
          s = 0
          exposant = len(binaire) - 1
          for a in binaire:
              s += int(a) * 2 ** exposant
              exposant -= 1
          return s

```

```

In [31]: # Tests, si pas d'erreurs continuez.
          assert ( bin_vers_dec('1111011') == 123 )
          assert ( bin_vers_dec('101111011') == int('0b101111011',2) )
          assert ( bin_vers_dec('1001101') == int('0b1001101',2) )
          # test aléatoire
          nbe = randint(30,1000)
          binaire = bin(nbe)[2:]
          assert bin_vers_dec(binaire) == nbe

```

1.4 Codage et décodage du message

```
In [42]: def codage_message(texte,cle):
        """
        Chiffre un texte par la méthode de César puis code les caractères en binaire.
        Le texte ne commence, ni ne se termine par des espaces.
        Entrées :
            texte (str)
            cle (int) la clé de codage
        Sortie :
            (str) Le texte codé
        """
        texte_code = ''
        texte_apres_cesar = cesar(texte,cle)
        for carac in texte_apres_cesar:
            carac_binaire = dec_vers_bin(ord(carac))
            # Remplissage avec zéros si nécessaire (pour obtenir 8 bits)
            while len(carac_binaire) != 8:
                carac_binaire = '0'+carac_binaire
            texte_code += carac_binaire
        return texte_code
```

```
In [33]: # Tests, si pas d'erreurs continuez.  
assert codage_message('Coucou le monde',5) == '010010000111010101010101'  
assert codage_message('César',10) == '0100110111101001011000'
```

```
In [34]: def decode_message(binaire, cle):
        """
        Déchiffre un message binaire codé avec la méthode du not
        Le texte ne commence, ni ne se termine par des espaces.
        Entrées :
            binaire (str)
            cle (int) la clé de décodage
        Sortie :
            (str) Le texte décodé
        """
        # Extraire un octet et enlever du texte codé
        texte_decode = ''
        while binaire:
            octet = binaire[:8]
            binaire = binaire[8:]
            caractere_code = bin_vers_dec(octet)
            caractere_decode = chiffrement_caractere(chr(caractere_code), cle)
            texte_decode += caractere_decode
        return texte_decode
```

```
In [35]: # Tests, si pas d'erreurs continuez.
```



0 Wdzi ejpé, qjpn vqzu yéxyé gz hznvzbz, qjpn êozn yzn xcvhkd
jin. Gv XDV qjpn vooziy !!
1 Vcyh dioé, piom upyt xéwixé fy gymmuay, piom ênym xym wbugjc
ihm. Fu WCU piom unnyhx !!
2 Ubxg chné, ohnl toxs wévhwé ex fxllttx, ohnl êmxl wxl vatfib
hgl. Et VBT ohnl tmmxgw !!
3 Tawf bgmé, ngmk snwr véugvé dw ewkksyw, ngmk êlwv vwv uzseha
gfk. Ds UAS ngmk sllwfv !!
4 Szve aflé, mflj rmvq uétfué cv dvjjrxv, mflj êkvj uvj tyrdgz
fej. Cr TZR mflj rkkveu !!
5 Ryud zeké, leki qlup téseté bu cuiiqwu, leki êjui tui sxqcfy
edi. Bq SYQ leki qjjudt !!
6 Qxtc ydjé, kdjh pkto sérsé at bthhpvt, kdjh êith sth rwpbex
dch. Ap RXP kdjh piitcs !!
7 Pwsb xcié, jcig ojsn réqcré zs asggous, jcig êhsg rsg qvoadw
cbg. Zo QWO jcig ohhsbr !!
8 Ovra wbhé, ibhf nirm qépbqé yr zrffntr, ibhf êgrf qrf punzcv
baf. Yn PVN ibhf nggraq !!
9 Nuqz vagé, hage mhql péoapé xq yqeemsq, hage êfqe ppe otmybu
aze. Xm OUM hage mffqzp !!
10 Mtpy uzfé, gzfd lgpk oénzoé wp xpddlrp, gzfd êepd opd nslxa
tzyd. Wl NTL gzfd lepyo !!
11 Lsox tyeé, fyec kfoj némyné vo wocckqo, fyec êdoc noc mrkwz
syxc. Vk MSK fyec kddoxn !!
12 Krnw sxdé, exdb jeni mélxmé un vnbbjpn, exdb êcnb mnb lqjvy
rxwb. Uj LRJ exdb jccnwm !!
13 Jqmv rwcé, dwca idmh lékwlé tm umaaiom, dwca êbma lma kpiux
qwva. Ti KQI dwca ibbmvl !!
14 Iplu qvbé, cvbz hclg kéjvké sl tlzzhnl, cvbz êalz klz johtw
pvuz. Sh JPH cvbz haaluk !!
15 Hokt puaé, buay gbkf jéiujé rk skyygm, buay êzky jky ingsv
outy. Rg IOG buay gzzktj !!
16 Gnjs otzé, atzx faje iéhtié qj rjxxflj, atzx êyix iix hmfru
ntsx. Qf HNF atzx fyyjsi !!
17 Fmir nsyé, zsyw ezid hégshé pi qiwweki, zsyw êxiw hiw gleqt
msrw. Pe GME zsyw exxirh !!
18 Elhq mrxé, yrxc dyhc géfrgé oh phvvdjh, yrxc êwhv ghv fkdps
lrqv. Od FLD yrxc dwwhqg !!
19 Dkgp lqwé, xqwu cxgb féeqfé ng oguucig, xqwu êvgu fgu ejcor
kqpu. Nc EKC xqwu cvvgpf !!
20 Cjfo kpvé, wpvt bwfa éedpé mf nfttbhf, wpvt êuft eft dibnq
jpot. Mb DJB wpvt buufœ !!
21 Bien joué, vous avez décodé le message, vous êtes des champ

ions. La CIA vous attend !!
22 Ahdm inté, untr zudy cébncé kd ldrzfd, untr êsdr cdr bgzlo
hnmr. Kz BHZ untr zssdmc !!
23 Zgcl hmsé, tmsq ytcx béambé jc kcqqyec, tmsq êrcq bcq afykn
gmlq. Jy AGY tmsq yrrclb !!
24 Yfbk glré, slrp xsbw aézlaé ib jbppxdb, slrp êqbp abp zexjm
flkp. Ix ZFX slrp xqqbka !!
25 Xejf fkqé, rkqo wrav zéykezé ha iaoowca, rkqo épao zao ydwil
ekjo. Hw YEW rkqo wppajz !!

1.5 Réponse :

clé : 21 et message : Bien joué, vous avez décodé le message, vous êtes des champions. La CIA vous attend !!

In []:

