

DROIT ET CONDUITE DE PROJET . 2020-2021



FACULTÉ DES SCIENCES ET TECHNIQUES
MASTER 1 - MATHS. CRYPTIS

COMPTE-RENDU

À l'attention de :

M. CRESPIN
M. DUSART
M. CONCHON

Rédigé par :

PIARD A.
JACQUET R.

Table des matières

1	Introduction et organisation	2
1.1	Introduction	2
1.2	Organisation	2
2	Premières réactions	3
3	Analyse des sujets	3
3.1	Premier sujet	3
3.1.1	Premier exercice	3
3.1.2	Second exercice	3
3.2	Second sujet	4
3.2.1	Premier exercice	4
3.2.2	Second exercice	4
3.2.3	Troisième exercice	4
4	Comparaisons entre étudiants	5
4.1	Sujet n°1	5
4.1.1	Premier exercice	5
4.1.2	Second exercice	7
4.2	Sujet n°2	10
4.2.1	Premier exercice	10
4.2.2	Second exercice	15
4.2.3	Troisième exercice	16
4.3	Conclusions	17
4.3.1	Conclusion de Raphaël	17
4.3.2	Conclusion d'Arthur	17
4.3.3	Conclusion générale	18

1 Introduction et organisation

1.1 Introduction

Le but de notre projet était d'entraîner et de tester les compétences, en terme de programmation, de nos collègues du Master 1 Maths CRYPTIS. Comme énoncé dans notre fiche d'avant-projet, deux *Competitive Programming Test* étaient prévus, avec les quatre mêmes participants pour chacun d'eux, et nous avions deux protocoles organisationnelles qui s'offraient à nous en vue de la condition sanitaire actuelle. Le protocole ayant été retenu est celui du distanciel.

Le premier sujet a été traité le 1er février 2021 et le second le 8 février 2021.

1.2 Organisation

Le « recrutement » des participants s'est fait à distance. Depuis le début de l'année nous avons un groupe Messenger et un serveur Discord pour la promo master 1 maths CRYPTIS 2020/2021. Nous nous sommes orientés à quasiment 100% sur le Discord, ce qui est bien plus pratique. C'est d'ailleurs Arthur qui avait pris l'initiative de créer ce serveur. C'est donc par cet intermédiaire que nous avons fait une première requête de participants aux alentours du 8 octobre 2020. Nous avons lors de cette requête transmis notre fiche d'avant-projet ainsi qu'un appel à candidatures.

Dans les faits, nous n'avons pas eus énormément de retours mais tout de même quelques intéressés. Pour ceux-là et pour d'autres personnes désireuses mais ne s'étant pas prononcé, nous avons organisé un vocal, toujours sur le Discord, pour répondre à leurs éventuels questions et notamment pour les rassurer car certaines personnes n'étaient pas confiantes vis à vis de leurs compétences en programmation. Lorsque nous avons connus les noms de certains participants, Raphaël a créé un autre serveur Discord dans lequel il y a deux groupes de personnes : un groupe pour les participants et un groupe pour les administrateurs, c'est à dire nous. Sur ce Discord il y avait différents salons. Un salon pour rappeler le but de notre projet, un salon pour prévoir d'un jour et d'un créneau horaire satisfaisant le plus de personnes possibles, un salon pour poster les sujets, un salon pour qu'ils postent leur travail en fin de session, un salon pour qu'ils nous posent des questions, etc, ainsi que deux salons vocaux, présents pour les débloquer s'ils avaient un souci lors de l'épreuve.

Enfin, pour la rédaction du rapport, nous avons encore une fois fait la majorité du travail en appel sur Discord. Cependant, après chaque fin de session, nous publions l'avancée du rapport sur **GitHub** de manière à ce que lorsque l'un d'entre nous souhaitait avancer de son côté, il ait la bonne version du rapport. Cela aurait aussi pu nous permettre de retourner sur une version plus ancienne du rapport si nous avions fait des modifications puis changé d'avis.

2 Premières réactions

À l'issue du premier *Competitive Programming Test*, nous nous sommes rendus compte que les étudiants ont mis beaucoup plus de temps que prévu sur les exercices, ce qui fait qu'aucun d'eux n'a terminé le premier sujet. Il se trouve que les exercices n'étaient pas triés par ordre de difficulté mais qu'ils ont tous choisis de commencer par le premier exercice.

Nous les avons légèrement questionnés et aidés à comprendre ce qui n'allait pas, en vue du second *Competitive Programming Test* qui était prévu la semaine suivante. Celui-ci s'est bien mieux passé. Les étudiants n'ont pas terminés le sujet mais ils ont tous traités au moins un exercice. La plupart des étudiants s'étant intéressés à deux exercices.

3 Analyse des sujets

3.1 Premier sujet

3.1.1 Premier exercice

L'objectif principal de ce premier exercice était de mettre l'étudiant face à un chiffré dont la méthode de chiffrement est inconnue. Il a été utilisé deux clés pour chiffrer le message. L'exercice en lui-même n'était pas difficile mais il fallait que l'étudiant tente diverses choses pour comprendre comment le message avait été chiffré. Le chiffrement en question est un chiffrement par substitution (de type César) pour lequel les clés étaient 3 et 13. Malgré les indications, la majorité des étudiants a perdu beaucoup de temps sur cet exercice, ce qui a influé sur l'ensemble du sujet. En effet un étudiant n'a pas pu effectuer l'exercice 2 et n'a pas trouvé comment résoudre l'exercice 1.

3.1.2 Second exercice

L'objectif de ce second exercice était de développer la méthode de déchiffrement d'un chiffrement précisément décrit. On rappelle que le choix du langage de programmation n'était pas imposé et que suivant le langage utilisé cela est plus ou moins rapide à écrire. Les étapes de l'algorithme étaient bien décomposées mais malgré tout nous n'avons pas eu beaucoup de retours, en raison du temps mal géré par les étudiants lors de l'exercice 1.

3.2 Second sujet

3.2.1 Premier exercice

L'objectif de cet exercice était de programmer l'algorithme de chiffrement écrit en français dans l'énoncé. Les seules difficultés étaient la maîtrise des opérateurs logiques et la conversion de type. Contrairement à nos attentes, cet exercice n'a pas très bien été réussi. Étant donné que l'épreuve a été conduite en distanciel, les étudiants avaient accès à toutes les ressources possibles. Malgré tout, certains n'ont pas su trouver comment effectuer un ET Logique.

3.2.2 Second exercice

L'objectif de cet exercice était d'écrire la méthode de déchiffrement à partir de celle qui a permis le chiffrement. Les opérations étaient donc à inverser. Si les étudiants possédaient la bonne fonction, il leur était alors possible de déchiffrer le message. Ici aussi de nombreuses difficultés se sont présentées aux étudiants, comme le passage de tableaux de bits à une chaîne de caractères.

3.2.3 Troisième exercice

L'objectif de cet exercice était de déchiffrer un fichier binaire donné. Afin de déchiffrer ce fichier, il fallait trouver la clef de chiffrement qui était écrite sur l'image dont l'extension n'est pas la bonne. Il fallait donc au préalable trouver le bon format d'image avec la commande type xxd, ou exiftool, ou autre. Une fois la clef récupérée il fallait la concaténer autant de fois que nécessaire pour obtenir une taille de clef similaire à la taille du fichier binaire. Enfin il suffisait d'effectuer un XOR entre le fichier binaire et la clef pour obtenir l'image finale. Cet exercice était volontairement légèrement plus difficile et comme attendu il n'a pas été traité.

4 Comparaisons entre étudiants

4.1 Sujet n°1

4.1.1 Premier exercice

```
1 import re
2
3 chiffre = "IPLU'QV\Đ 'Ç' [VP3'SL'TLZZHNL' JHJOD'LZ['A'HGT1GXJ' ("
4
5 n = len(chiffre)
6 print(n)
```

Ici il est clair que l'étudiant n'a eu aucune idée de comment traiter l'exercice. Du moins il n'en a pas laissé de traces.

```
1 #!/usr/bin/python3
2
3 #EXERCICE 1
4 alph="abcdefghijklmnopqrstuvwxyzâçéèù"
5 echantillonfr="Quand, ..."
6 chif="IPLU'QV\Đ 'Ç' [VP3'SL'TLZZHNL' JHJOD'LZ['A'HGT1GXJ' ("
7
8 def minuscule(texte):
9     tmin=texte.lower()
10    return [c for c in tmin if c in alph]
11
12 def anfreq(chaine):
13     dicoccur={}
14     for c in chaine:
15         if c in dicoccur:
16             dicoccur[c]=dicoccur[c]+1
17         else:
18             dicoccur[c]=1
19     listocc=list(dicoccur.items())
20     listocc=sorted(listocc,key=lambda t:t[1])
21     listocc.reverse()
22     return listocc
23
24 def replaUN(chaine,dicodéco):
25     ecart=ord(dicodéco[0][1])-ord(dicodéco[0][0])
26     for x in (0,len(chaine)):
27         chaine[x]=chr(ord(chaine[x])+ecart)
28     return chaine
29
```

```

30 # Appels des fonctions (main())
31 statsfr=anfreq(minuscul(echantillonfr))
32 statsch=anfreq(chif)
33 llcl=[t[0] for t in statsfr]
34 llch=[t[0] for t in statsch]
35 dicodage=list(zip(llch,llcl))
36 #print(dicodage)
37 chclUN=replaUN(chif,dicodage)

```

Cet étudiant a eu la bonne idée de faire de l'analyse fréquentielle depuis un texte écrit en français. Grâce à cette méthode, il peut trouver une clef et reconstituer une partie du message. Ensuite, il peut essayer de trouver la seconde clef, compte-tenu du fait qu'il connaît déjà une partie du message, en faisant le déchiffrement à la main. Cet étudiant utilise de nombreuses fonctions Python élémentaires, il maîtrise la gestion de liste, les dictionnaires et les boucles.

```

1  #!/usr/bin/python3
2  alphabet = 'abcdefghijklmnopqrstuvwxyz'
3  # La fréquence d'apparition des lettres varie bien évidemment en fonction
4  # de la langue et du type de texte
5  # Pour un texte rédigé en français, on a :
6  liste_frequence=['e',' ','a','s','i','t','u','r','n'] #quelques lettres
7  #par ordre décroissant
8
9  chaine=input('chiffré : ')
10 chaine=chaine.lower()
11 chaine=[c for c in chaine if c in alphabet]
12 dico_occurences = {}
13
14 def occurence(chaine):
15     for c in chaine:
16         if c in dico_occurences:
17             dico_occurences[c] = dico_occurences[c] + 1
18         else:
19             dico_occurences[c] = 1
20     print(dico_occurences)
21
22     liste_elements = list(dico_occurences.items())
23     liste_elements = sorted(liste_elements,key=lambda t:t[1])
24     print(liste_elements)
25     dico=dict(liste_elements)
26     liste_key=list(dico.keys())
27
28     n=alphabet.index(liste_key[len(liste_key)-1])
29     m=alphabet.index('e')

```

```

30         if n >= m :
31             decalage=n-m
32         else:
33             decalage=m-n
34         return decalage
35
36 texte_clair=''
37 decalage=occurence(chaine)
38 decalage=26-decalage #pour déchiffrer
39 alphabet_dechiffrement = alphabet[decalage:] + alphabet[:decalage]
40 associations = dict(zip(alphabet,alphabet_dechiffrement.upper()))
41 for c in chaine:
42     if c in associations :
43         texte_clair += associations[c]
44 print("#####")
45 print("le message clair : ")
46 print(texte_clair.lower())
47 # je parviens pas à continuer !!!!!

```

Cet étudiant a suivi le même raisonnement que le précédent. Son raisonnement arrive au même résultat et en allant un peu plus loin il aurait pu arriver au résultat final. Il maîtrise quand même lui aussi les listes, dictionnaires et les fonctions élémentaires de python.

On voit que seulement deux personnes sur quatre ont traités cet exercice ce qui est moins que ce que nous attendions.

4.1.2 Second exercice

```

1  #!/usr/bin/python3
2  import re
3  A = """155c04970e1747172e4e070e044e17776e076526
4  0417474e27330474074e2767072e4e070e9726044f2f776e0407e704272f
5  0f042e76074e970e4f27047623"""
6  n = len(A)
7
8  def hexad(n): #fonction qui prend transforme un nombre en haxa et
9  # le passe en entier
10     h = 0
11     for i in range(len(n)):
12         if n[i] == 'a':
13             h += 10 * 16**(1-i)
14         elif n[i] == 'b':
15             h += 11 * 16**(1-i)
16         elif n[i] == 'c':

```



```
17         h += 12 * 16**(1-i)
18     elif n[i] == 'd':
19         h += 13 * 16**(1-i)
20     elif n[i] == 'e':
21         h += 14 * 16**(1-i)
22     elif n[i] == 'f':
23         h += 15 * 16**(1-i)
24     else:
25         h += int(n[i]) * 16**(1-i)
26     return h
27
28 def bbinaire(x):
29     b = str(bin(x))
30     b = b[2:]
31     n = len(b)
32     if n<8:
33         b = (8-n)*'0' + b
34     return b
35
36 def echange(n):
37     if n< 10000000 and n%10 == 1:
38         n= n + 10000000 -1
39     elif n>1000000 and n%10 == 0:
40         n = n - 1000000 +1
41     return n
42
43 B = []
44 m = len(B)
45
46 for i in range(0,n-1,2):
47     B.append(bbinaire(hexad(A[i:i+2])))
48
49 #for j in range(m):
50 #mon but était de faire le chemin inverse avec mes fonctions
51
52 print(B)
```

Ici notre étudiant a créé des fonctions qui n'ont pas réellement d'intérêt puisqu'il existe des fonctions élémentaires en python qui permettent, par exemple, de passer d'un entier à sa représentation hexadécimale et vice-versa. La fonction `bbinaire` permet d'utiliser des octets. Quant à la fonction `echange`, elle, n'a aucun intérêt. L'étudiant a quand même essayé de faire quelque chose mais ça reste très loin du résultat final.

```

1  #!/usr/bin/python3
2  import sys , re
3  liste1 = []
4  liste2 = []
5  message = """155C04970E1747172E4E070E044E17776E0
6  765260417474E27330474074E2767072E4E070E9726044F2F776e040
7  7e704272f0f042e76074e970e4f27047623"""
8
9  p = re.compile('\w\w')
10 liste1 = p.findall(message)
11
12 for lettre in liste1 :
13     liste2 += bin(int(lettre, 16))
14
15 print (liste2)

```

Cet étudiant a utilisé les expressions régulières pour récupérer deux à deux les caractères du chiffré. Il les convertit ensuite en binaire.

```

1  #!/usr/bin/python3
2  message = input("Entrez le message à envoyer? : ")
3  l=len(message)
4  liste=[]
5  def gest_caractere(message):
6      blocs=[[message[2*i:2*i+2]] for i in range(0,l//2)]
7      #on construit des blocs de 2 elements
8      print(blocs)
9
10     for i in range(0,len(blocs)):
11         liste.append(blocs[i][0])
12     print(liste)
13     liste_bin=[bin(int(liste[i], 16)) for i in range(0,len(liste))]
14     print(liste_bin)
15     #le temps ??????
16 gest_caractere(message)

```

Cet étudiant a eu la bonne idée de découper le message en blocs de deux caractères, il les stocke dans une liste qu'il convertit ensuite en une liste de nombres binaires. Les lignes 10 à 12 n'ont aucun intérêt. Il pouvait totalement travailler avec la liste blocs. Toutefois il n'aura effectué que deux opérations sur la dizaine demandée.

L'objectif de cet exercice était pourtant clair, mais il a dû susciter beaucoup d'incompréhension vis à vis de nos étudiants qui ont utilisés beaucoup de fonctions inutiles.

4.2 Sujet n°2

4.2.1 Premier exercice

```
1  #!/usr/bin/python3
2  import os, sys
3  phrase = 'Il fait beau ce matin.'
4  phrase = phrase.upper()
5  PH = []
6  for u in phrase:
7      PH.append(ord(u))
8  def premf(A):
9      for i in range(len(A)):
10         if A[i]%3 == 0:
11             A[i] += (i+13)%i
12     return A
13  premf(PH)
14  ch = ""
15  for i in range(len(PH)):
16      ch += chr(PH[i])
17
18  CH = bytes(ch, 'utf-8')
19  ME = []
20  for i in CH:
21      if i%2 ==0:
22          ME.append(i^5)
23      elif i%3==0:
24          ME.append(i|6)
25      elif i%5==0:
26          ME.append(i&2)
27      else:
28          ME.append(i)
29
30  for i in range(len(ME)):
31      ME[i] = chr(ME[i])
32
33  mes_fin = ''.join(ME)
34  print(mes_fin)
35
36  # j'obtiens II%CI%GI%CW%MaI[+
```

Cet étudiant a bien compris le but de cet exercice. Il utilise dans le bon ordre les différentes fonctions et a compris de manière générale les étapes à suivre. Toutefois il a fait des erreurs. Il n'a pas utilisé les bons opérateurs logiques pour obtenir le bon chiffré.

```
1  #!/usr/bin/python3
2  #Exercice 1
3  message = 'Il fait beau ce matin.'
4  msg = message.upper()
5  L = []
6  for lettre in msg:
7      L.append(ord(lettre))
8  print(L)
9  L_prime = []
10 for nombre in L:
11     if int(nombre)%3 == 0:
12         L_prime.append((int(nombre)+13)%int(nombre))
13     else:
14         L_prime.append(nombre)
15 print(L_prime)
16 #subsidaire
17 #chaîne = ''
18 #for nombre in L_prime:
19 #     c = chr(nombre)
20 #     print(c)
21 #     chaîne += c
22 #subsidaire
23 L_bin = []
24 print(bin(67))
25
26 for nombre in L_prime:
27     b = bin(nombre)
28     if int(nombre)%2 == 0:
29         L_bin.append(b[2:len(b)]^101)
30     elif int(nombre)%3 == 0:
31         L_bin.append(b[2:len(b)] or 110)
32     elif int(nombre)%5 == 0:
33         L_bin.append(b[2:len(b)] and 10)
34     else:
35         L_bin.append(b[2:len(b)])
36
37 print(L_bin)
38 # NON FINI
```

Cet étudiant a lui aussi bien suivi l'ordre de l'algorithme défini, mais il n'a pas terminé son code. Il avait bien compris le but de l'exercice et connaissait ses opérateurs logiques. Toutefois il a fait des erreurs comme la modification des octets suivant la parité de l'index du tableau.

```
1  #!/usr/bin/python3
2  import sys
3  message = "Il fait beau ce matin."
4  message = message.upper()
5  tableau=[]
6
7  for i in range(0, len(message)) :
8      tableau.append(ord(message[i]))
9      if(tableau[i]%3==0):
10         tableau[i]+= (13+i)%i
11  for i in range(0, len(message)-1) :
12      if(i%2==0):
13         tableau[i] = tableau[i]^5
14      if(i%3==0):
15         newl=""
16         l = bin(tableau[i])
17         for j in range(0, len(l)-3):
18             newl += l[j]
19         newl += "11"
20         newl += l[len(l)-1]
21         tableau[i] = int(newl, 2)
22      if(i%5==0):
23         newl=""
24         l = bin(tableau[i])
25         for j in range(0, len(l)-3):
26             newl += l[j]
27         newl += "0"
28         newl += l[len(l)-2]
29         newl += "0"
30         tableau[i] = int(newl, 2)
31  l = ""
32  for i in range(0, len(message)-1) :
33      l += chr(tableau[i])
34
35  print(l)  #JL%FDHV BO@U'CW"HAfIZ
```

Cet étudiant a presque le bon résultat. Sa méthodologie est bonne mais ses opérations logiques sont mal effectuées. En effet, il faut les faire avec des bits. La chaîne "110" n'est pas le bit 110. Avec un peu plus de rigueur et d'attention, cet étudiant aurait obtenu le bon chiffré. Il est quand même important de souligner que tout le reste est correct.

```
1  #!/usr/bin/python3
2
3  message='il fait beau ce matin'
4  message=message.upper()
5  #print(message)
6  tableau=[ord(i) for i in message]
7  print(tableau)
8  for i in range(0,len(tableau)):
9      if tableau[i]%3==0:
10         tableau[i]=tableau[i]+(13+i)%i
11 #print(tableau)
12 tableau_caract=[chr(i) for i in tableau]
13 print(tableau_caract)
14 chaine=''.join(tableau_caract)
15 print(chaine)
16 tableau_oct=[bin(ord(i))[2:] for i in chaine]
17 print(tableau_oct[0])
18 for i in range(0,len(tableau_oct)):
19     if tableau[i]%2==0:
20         tableau[i]=tableau[i]^int('101',2)
21     if tableau[i]%3==0:
22         tableau[i]=tableau[i]|int('110',2)
23     if tableau[i]%5==0:
24         tableau[i]=tableau[i]&int('010',2)
25 #print(tableau)
26 chaine_f=[chr(i) for i in tableau]
27 chaine_f=''.join(chaine_f)
28 print(chaine_f)
```

Cet étudiant a une excellente méthodologie, il a bien suivi l'énoncé. La seule erreur notable est la mauvaise écriture des opérateurs logiques en python. En dehors de cette erreur, le code est totalement correct.

```
1  #!/usr/bin/python3
2  #EXERCICE 1
3  txt="Il fait beau ce matin."
4  #1
5  TXT=txt.upper()
6  #2 et 3
7  tab=[]
8
9  for i in range(0,len(txt)):
10     tab.append(ord(txt[i]))
11     if tab[i]%3==0:
```

```
12         tab[i]=tab[i]+((13+i)%i)
13     #4
14     rst=""
15
16     for i in range(0,len(tab)):
17         rst=rst+chr(tab[i])
18     #5
19     loct=list(map(bin,rst.encode('utf-8')))
20     #6
21     def tabmodif(ta):
22         for i in range(0,len(ta)):
23             if i%2==0:
24                 ta[i]=ta[i]^101
25             if i%3==0:
26                 ta[i]=ta[i] (ou inclusif) 110
27             if i%5==0:
28                 ta[i]=ta[i] (et) 010
29         return ta
30     #7
31     #Si fait avec la liste du 3)
32     rslt=""
33
34     for i in range(0,len(tab))
35         rslt=rslt+chr(tab[i])
36
37     #Si fait avec liste du 5)
38     rsltat=""
39     for i in range(0,len(loct)):
40         rsltat=rsltat+bytes([int(loct[i],2)]).decode('utf-8')
```

Cet étudiant était sur la bonne piste mais il n'a pas été au bout de la réflexion et a manqué de rigueur sur la rédaction du code. On remarque qu'il ne connaît ni les opérateurs logiques ni la conversion chaîne de caractères vers bits. Même après "correction" du code, le résultat n'est pas du tout celui attendu. Les légères corrections apportées étaient nécessaires pour pouvoir exécuter le programme. Autrement, les lignes de code de l'étudiant n'ont pas été modifiées.

4.2.2 Second exercice

```
1  #!/usr/bin/python3
2  import os, sys
3  ph="FC?E^FK?YZ^J?XZ?B^KFM-"
4  S = bytes(ph,'utf-8')
5  A = []
6  for u in S:
7      A.append(chr((u^21)-10))
8  phrase = ''.join(A)
9  print(phrase)
10 # résultat = IL FAIT BEAU CE MATIN.
```

Cet étudiant a réussi l'exercice. Il a même prit l'initiative d'utiliser le xor entre entiers, ce qui fonctionne avec python. Il obtient le bon résultat ce qui est satisfaisant.

```
1  #!/usr/bin/python3
2  #Exercice 2
3  message = 'FC?E^FK?YZ^J?XZ?B^KFM-'
4  L = []
5  for lettre in message:
6      L.append(ord(lettre) + 10)
7  print(L)
8  msg = ''
9  for i in L:
10     msg += chr(i)
11  print(msg)
12 # NON FINI
```

Cet étudiant n'a effectué que les trois premières étapes, sûrement par faute de temps. Il était toutefois sur la bonne voie. Les premières lignes de codes sont justes.

```
1  #!/usr/bin/python3
2  import sys
3  message = "FC?E^FK?YZ^J?XZ?B^KFM"
4  dechiffre = ""
5  for i in range(0, len(message)) :
6      dechiffre += chr((ord(message[i])^21)-10)
7  print(dechiffre) #IL FAIT BEAU CE MATIN
```

Cet étudiant a également réussi l'exercice et de façon encore plus compacte. Il n'y a pas grand chose à redire, sa maîtrise du XOR entre entiers est intéressante.


```
1  #Chiffrement:
2  chif='FC?E^FK?YZ^J?XZ?B^KFM-'
3  tabl=[]
4  chaine=""
5  for i in range(0,len(chif)):
6      tabl.append(ord(chif[i])+10)
7      chaine=chaine+chr(tabl[i])
8  lisoct=list(map(bin,chaine.encode('utf-8')))
9  for i in range(0,len(lisoct)):
10     lisoct[i]=lisoct[i]^10101
11  retout=""
12  for i in range(0,len(lisoct)):
13     retour=retour+bytes([int(lisoct[i],2)]).decode('utf-8')
14  #Déchiffrement
15  byt=list(map(bin,chif.encode('utf-8')))
16  inter=""
17  clair=""
18  for i in range(0,len(byt)):
19     byt[i]=byt[i]^10101
20     inter=inter+bytes([int(byt[i],2)]).decode('utf-8')
21  t=[]
22  for i in range(0,len(inter)):
23     t.append(ord(inter[i]))
24     t[i]=t[i]-10
25     clair=clair+chr(t[i])
```

Cet étudiant a essayé de coder la méthode de chiffrement, ce qui était inutile. Peut-être espérait-il que ça l'aiderait. Ceci étant, sa méthode de chiffrement ne marche pas. En effet, il a tenté d'effectuer un XOR entre deux types différents à la ligne 10.

Pour le déchiffrement il a effectué la même erreur, c'est à dire un XOR entre deux types différents (à la ligne 19) donc sa méthode ne marche pas. Cela dit, hormis cette erreur importante, le reste du programme semble correct.

4.2.3 Troisième exercice

A notre plus grand regret cet exercice n'a pas été traité. Cela aurait été un bon moyen de départager les étudiants et c'est celui qui ressemblait le plus à un CTF (Capture The Flag).

4.3 Conclusions

4.3.1 Conclusion de Raphaël

Le choix de mon binôme était assez naturel. Nous nous connaissons bien avec Arthur et j'ai le sentiment que nous réussissons à travailler ensemble. L'idée du **Competitive Programming Test** nous est venue assez rapidement. Celle-ci ayant émergée principalement en raison de mon attrait pour la programmation qui s'est un petit peu plus développé suite aux séances de travaux pratiques du module Réseau et Système, et suite à la découverte des **Capture The Flag**.

Nous avons donc chacun créés des exercices de programmation relativement basiques mais utilisant des notions de cryptographie. Arthur en avait fait en Java et de mon côté en Python. Nous avons chacun essayé de ré-écrire les programmes de l'autre, dans le langage que nous avons utilisé, pour voir approximativement le temps nécessaire à leur écriture, et ainsi pour faire des sujets cohérents avec le temps imparti pour ces derniers.

Nous nous échangeons les fichiers par notre conversation personnelle sur Discord. Pour l'écriture des sujets, nous avons profité de ma présence sur Limoges pour se voir et donc les rédiger. Bien que les exercices étaient déjà prêts. Étant parti de Limoges après les examens de janvier, nous avons arrangé les détails par appel sur Discord en partage d'écran.

Enfin, pour l'organisation avec les autres étudiants, nous avons lancé un appel à candidatures sur le serveur Discord de notre promotion, qui avait été créé par Arthur au début de l'année. Une fois que nous avons eu suffisamment de participants, nous avons créés un nouveau serveur Discord pour que l'on se retrouve tous et que la préparation des épreuves et leur déroulement y soit simplifié.

Pour conclure, c'est un projet qui s'est bien déroulé. Autant au niveau de notre binôme qu'avec les autres étudiants qui semblent avoir apprécié participer et progressé. Le partage des tâches et notre bonne entente ont permis de bien mener ce projet, sachant que la plupart de nos sessions de travail se sont déroulées en appel et partage d'écran.

4.3.2 Conclusion d'Arthur

Pour ma part j'ai apprécié faire ce projet dirigé conjointement avec Raphaël. Mon intérêt pour la programmation diverse et les diverses sessions de travaux pratique lors du premier semestre a fait naître cette idée de confronter des étudiants à des exercices de programmations divers et variés utilisant la programmation pure comme la réflexion cryptographique. La création des sujets a été une partie assez intéressante et ludique. Il fallait trouver le bon équilibre sur la difficulté. Raphaël et moi avons fait les corrections pour sa part en **Python** et moi-même en **Java**. Cela permettait de voir les avantages et éventuellement inconvénients sur ces différents langages. Malgré les circonstances qui n'étaient très clairement pas optimales, nous avons pu nous débrouiller grâce à la plateforme **Discord** et organiser ces épreuves à distance. Je suis un peu déçu du peu de résultat pour quelques sujets mais je pense que le stress et/ou une mauvaise organisation du temps peuvent mener quelques fois à des échecs. Malgré tout j'étais très content d'avoir pu participer à la création de ce projet et les retours des étudiants testés étaient bons donc je peux considérer que ce projet s'est bien déroulé.

4.3.3 Conclusion générale

Les prémices de ce projet ont été de choisir des étudiants du **Master Maths. CRYPTIS** exclusivement. Notre choix s'est porté sur des étudiants avec des profils très différents et chacun avec leurs spécificités. Il aurait été trop facile de choisir des étudiants de la filière informatique du **Master CRYPTIS**.

Il est clair que les étudiants ont été moins productif que ce nous attendions. Mais, a posteriori, lorsque l'on compare ce qu'ils ont fait en ce début de semestre 2 et aujourd'hui avec les travaux pratiques auxquels nous assistons, il y a une différence notable. En effet, leur niveau est pour la majorité à la hausse. Cela s'explique par un conditionnement et une pratique plus régulière de l'algorithmique générale. La matière **Sécurité TIC** qui utilise les notions de **Python** vues au premier semestre et l'utilisation de nouvelles bibliothèques comme **OpenSSL** n'est pas un problème pour eux. Ils arrivent beaucoup mieux à suivre les étapes et à traduire les énoncés en lignes de code. Nous avons beaucoup discutés avec les étudiants testés après les **CPT**.

En leur expliquant les subtilités et les cheminements d'idées nécessaire à la bonne résolution de tel ou tel exercice, nous pensons fortement avoir augmenté leur niveau en programmation, ce qui était le but recherché.

Nous avons apprécié ce type de projet en autonomie. Avec la situation sanitaire plus que complexe durant ce second semestre nous avons dû nous organiser autrement et faire les épreuves en distanciel. Cela n'a pas été idéal pour nous comme pour les étudiants mais nous sommes satisfait du résultat. Comme conclusion plus globale nous pouvons tirer comme enseignement que les étudiants ayant fait peu voir jamais d'informatique (orientée programmation) pendant leur cycle de Licence seront nécessairement plus en difficulté dans le cursus **Maths. CRYPTIS**. Nous recommandons donc aux futurs promotions de travailler, seul, ou bien avec toutes les ressources qu'Internet propose, leurs bases en programmation.