La Cryptanalyse des méthodes de substitutions alphabétiques

TIPE 2017-2018
Milieux : Interfaces, Interactions, Homogénéité, Ruptures

Problématique

- Comment, à partir de cryptanalyses de méthodes de chiffrement plus ou moins avancées, trouver un chiffrement plus sûr par déduction des failles des précédents ?

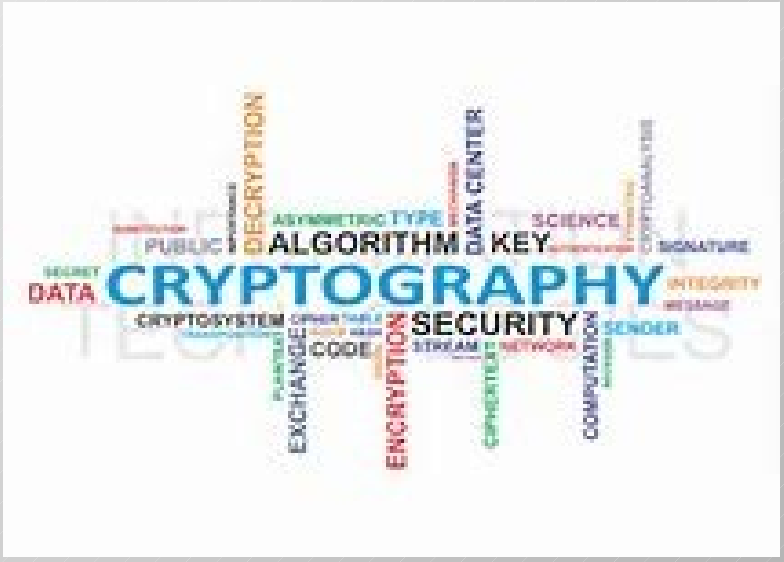
- Reproduire sur quelque chose de très élémentaire un schéma de pensée que pourrait avoir un scientifique lors de la création d'une méthode de chiffrement

Organisation de la présentation

- *Introduction*
- *Analyse de méthodes connues*
- *Déduction des failles permettant le cassage*
- *Présentation de nouvelles méthodes de chiffrement alphabétique*
- *Critique des méthodes apportées*
- *Conclusion*
- *Annexe (* Catalogue des programmes et fonctions présentées *)*

Introduction

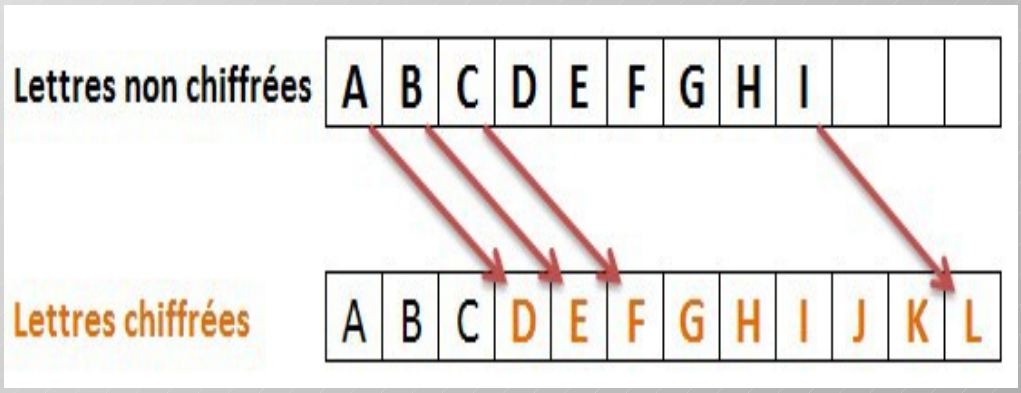
- Notions abordées : chiffrement, méthode de chiffrement, clef de chiffrement, cassage de méthodes
- Intérêt de la cryptographie
- On restera dans un cadre modeste



Études de méthodes connues

Méthode de César :

- Décalage simple dans l'alphabet
- Simple à mettre en œuvre, allant de pair avec la simplicité du cassage
- Clef de chiffrement très simple
- Vulnérable à l'analyse des fréquences

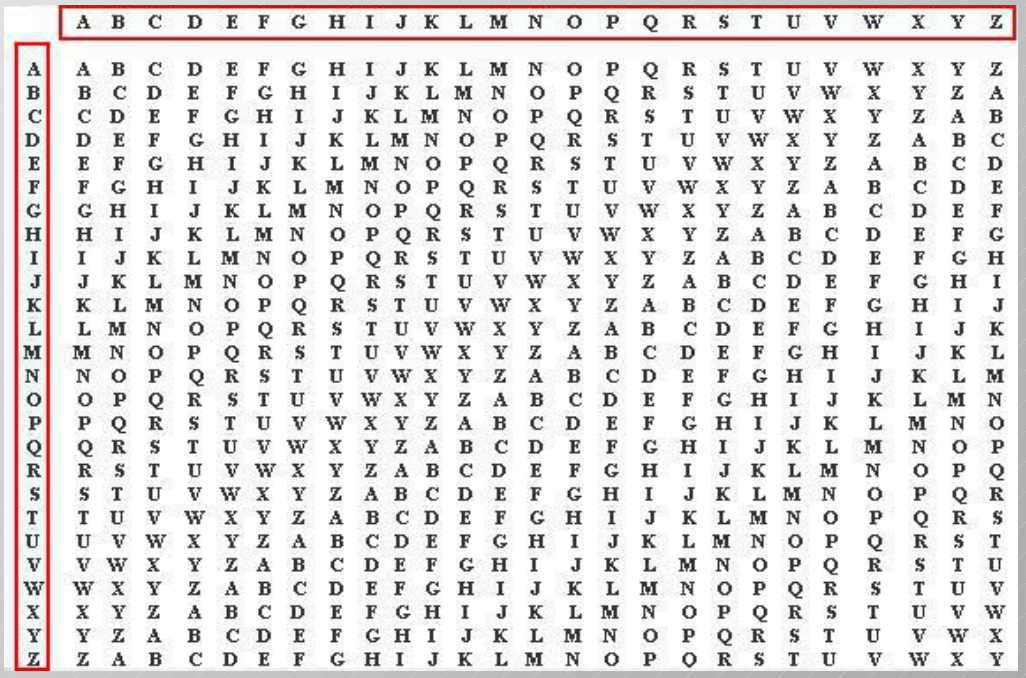



```
>>> cesar_x(7,Exemple2,'chiffrement')
'UL ZVTIYL WHZ KVBJLTLUA KHUZ JLAAL KVBJL UBPA'

>>> cesar_x(7,'UL ZVTIYL WHZ KVBJLTLUA KHUZ JLAAL KVBJL UBPA','dechiffrement')
'NE SOMBRE PAS DOUCEMENT DANS CETTE DOUCE NUIT'
```

Études de méthodes connues

Méthode de Vigenère :

- Tableau de chiffrement
- La redondance donne accès à la longueur de la clef
- « Amélioration » de la méthode de César



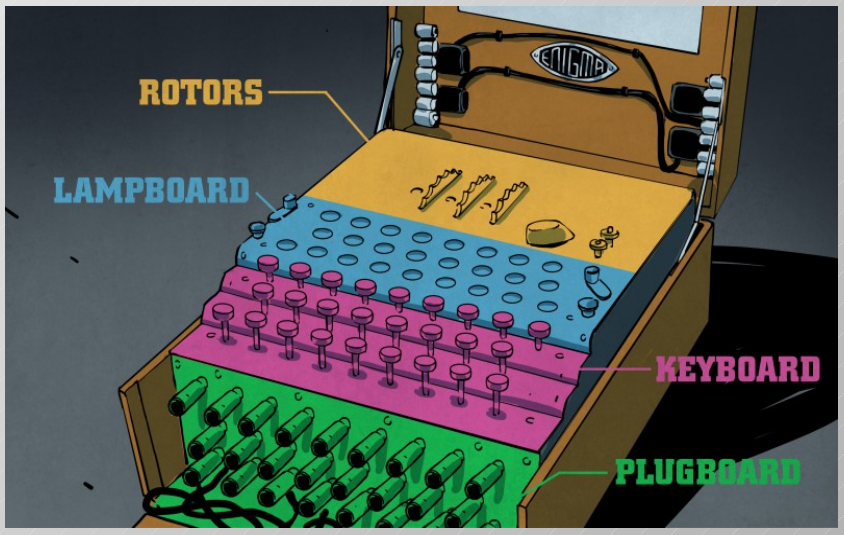
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

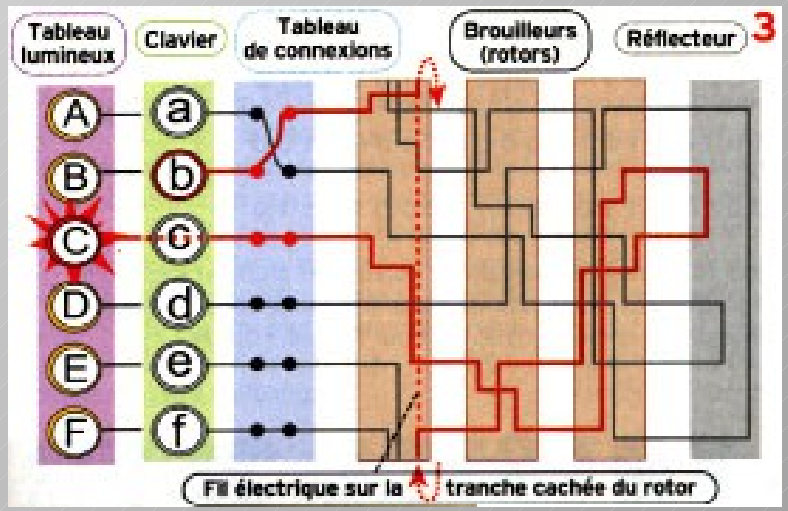
```
>>> vigenere('MUSIQUE','NE SOMBRE PAS DOUCEMENT DANS CETTE DOUCE NUIT')
'ZY AEGFDY XQM PIMKUGIZN LQHW WWBJY PIMKU RGCL'

>>> DecryptVigenere('MUSIQUE','ZY AEGFDY XQM PIMKUGIZN LQHW WWBJY PIMKU RGCL')
'NE SOMBRE PAS DOUCEMENT DANS CETTE DOUCE NUIT'
```

Études de méthodes connues

Méthode d'Enigma :

- Méthode qui a tenu bon face aux mathématiciens
- Clef de chiffrement se modifiant tout au long du chiffrement
- Seule une erreur humaine a permis le cassage





Synthèse des écueils à éviter

- Il est nécessaire d'éviter les redondances, premier angle d'attaque du cryptanalyste
- « 1 lettre = 1 lettre » ou « 1 lettre = 1 symbole »
- Clef de chiffrement restant la même tout au long du chiffrement, la rendant plus apparente
- Clef de chiffrement trop « simple », le fait qu'elle soit courte implique ici une relative simplicité

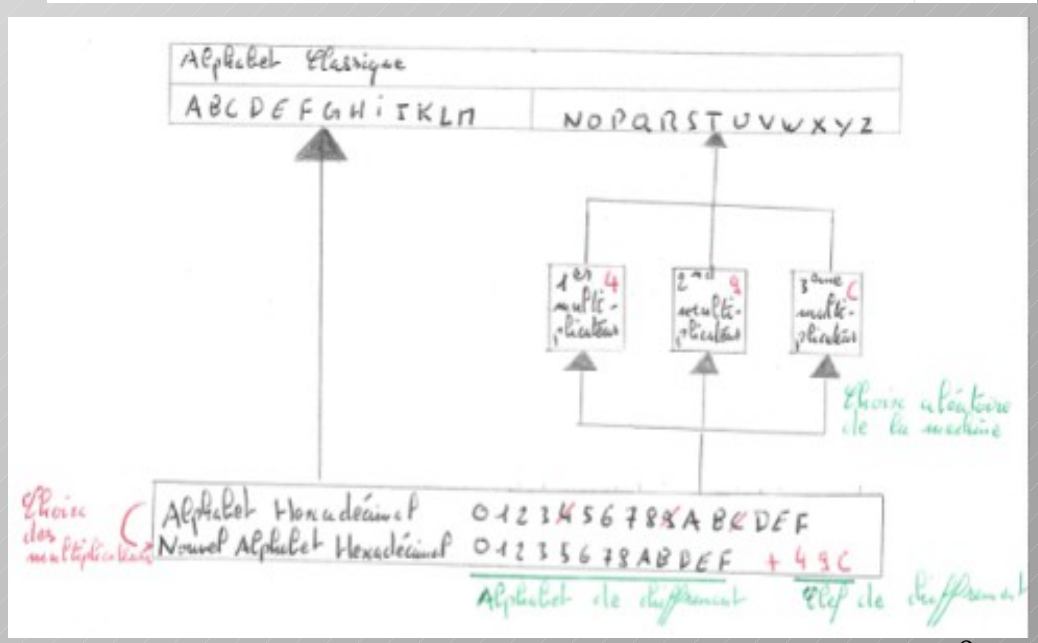
Propositions de nouvelles méthodes

HexaEncrypt :

- **Construction de la méthode**

```
>>> transpose_texte_h(Exemple2,'4','9','C')
'9059641F1C55C20463419825F5909730904625 47C7534148254098AC7'

>>> Dechiffre_Hexa('9059641F1C55C20463419825F5909730904625 47C7534148254098AC7','4','9','C')
'NESOMBREPASDOUCEMENTDANSCETTEDOUCENUIT'
```

- Choix de 3 « multiplicateurs », la clef de chiffrement
- Découpage de l'alphabet en 2 parties de 13 caractères
- Rôle du multiplicateur



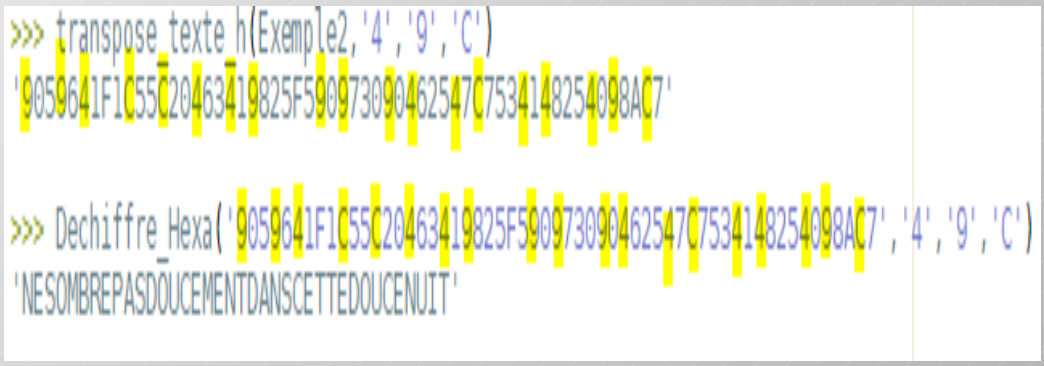
Proposition d'une nouvelle méthode

HexaEncrypt :

- Alphabet Hexadécimal peu ou pas utilisé dans la cryptographie
- Permet de transcender le classique « 1 lettre = 1 symbole »
- Concept de « multiplicateurs » permettant le déchiffrement et de briser l'égalité des longueurs de chaînes de caractères

Présentation et analyse des résultats obtenus

- Visibilité des multiplicateurs et donc de la clef
- Quasi équivalence de place dans les alphabets
- Faible utilité des multiplicateurs



- 19 multiplicateurs sur 57 caractères : 33%
- 3 multiplicateurs sur 16 dans l'alphabet héxadécimal : 18,5%

Proposition d'une nouvelle méthode

HexaEncryptFinal :

- Rajout de la variation en temps réel des multiplicateurs
- Correction du problème de la visibilité de la clef

```
while i<len(T) :
    if i%modulo == 0 :
        Crypt=Crypt + transpose_texte_h(T[i-modulo:i],m1,m2,m3)
        n=len(Crypt)
        m1=Crypt[n-1]
        m2=Crypt[n-2]
        m3=Crypt[n-3]
    i+=1
```

```
>>> transpose_final_texte_h(Exemple2,'4','9','C',8)
'40546C1F195592096391982 4F68029508047364979654 20A46A1A9B08'

>>> Dechiffre_final_Hexa('40546C1F195592096391982 4F68029508047364979654 20A46A1A9B08','4','9','C',8)
'NESOMBREPASDOUCEMENTDANSCETTEDOUCENUIT'
```

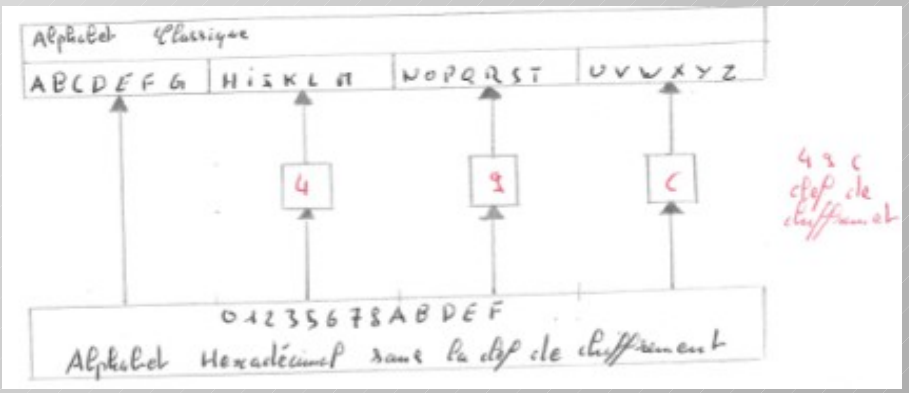
Améliorations possibles

HexaEncrypt :

- Création d'un alphabet à ordre aléatoire
- Trouver une meilleure utilité aux multiplicateurs

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

import random as rd

def SupprimerDansListe(i,L):
    Ll=[]
    for j in range(len(L)):
        if j != i :
            Ll.append(L[j])
    return(Ll)

def AlphabetAleatoire():
    Alphabet=Alpha
    AlphaAleat=[]
    i=len(Alpha)
    while i!=1:
        i=len(Alphabet)
        r=rd.randint(0,i - 1)
        AlphaAleat.append(Alphabet[r])
        Alphabet=SupprimerDansListe(r,Alphabet)
    return(AlphaAleat)
```

```
>>> AlphabetAleatoire()
['H', 'D', 'X', 'Z', 'S', 'N', 'E', 'B', 'M', 'P', 'K', 'Y', 'L', 'G', 'F', 'V', 'I', 'U', 'W', 'C', 'T', 'J', 'A', 'Q', 'R', 'O']
```

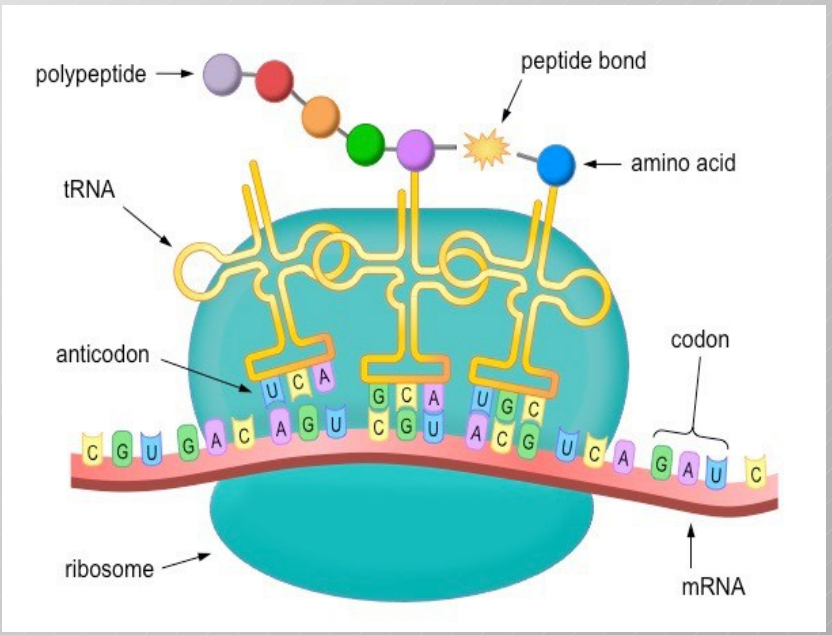


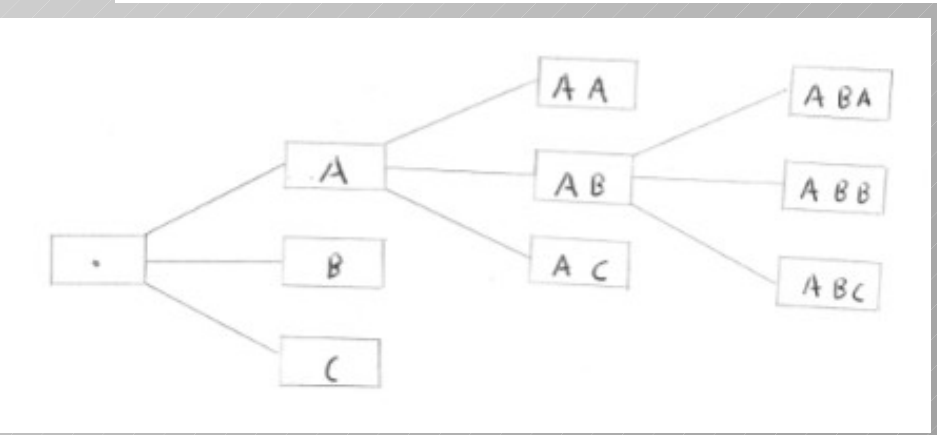
Propositions de nouvelles méthodes

CodonScript :

- **Construction de la méthode**

- Principe d'action de l'ARNt sur les codons de nucléotides
- Clef de chiffrement sous forme de matrice
- Arbre de complétion





Proposition d'une nouvelle méthode

CodonScript :

- Permet de transcender le classique « 1 lettre = 1 symbole »
- Passage de 26 possibilités à 26^3 possibilités par caractère
- Clef de chiffrement sous forme de matrice difficile à casser de force au vu du poids de celle ci (taille,temps,...)
- Construction de la clef via le parcours aléatoire de l'arbre rendant la clef quasi unique

Présentation et analyse des résultats obtenus

CodonScript :

- La clef de chiffrement ne se modifie pas au cours du temps , ce que l'on tentait d'éviter
- La clef de chiffrement est excédemment volumineuse, pour une estimation du nombre de groupes utiles faible

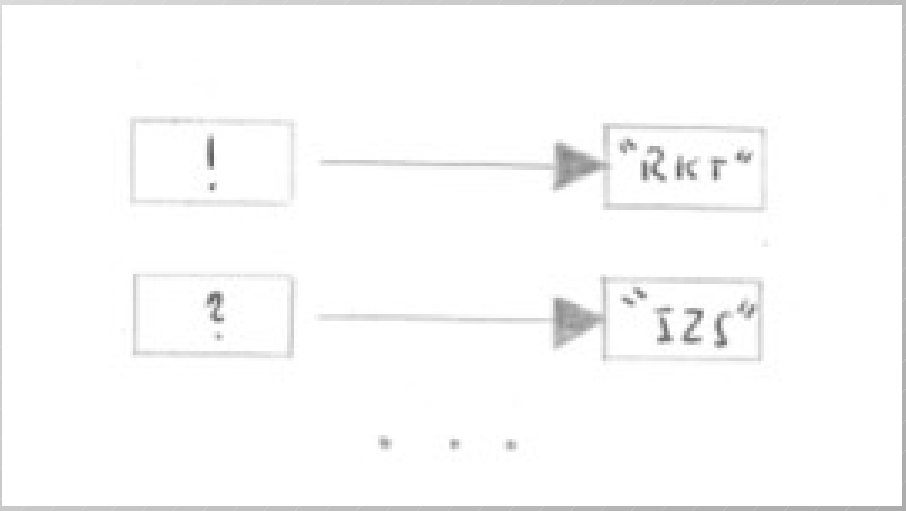
- Pas de résultats significatifs à cause de la clef très difficilement manipulable de par sa taille

- Taille réelle : 17 576 groupes de 3 lettres (26^3)

Améliorations possibles

CodonScript :

- Associer des groupes de lettres non utiles à la ponctuation
- Trouver comment faire varier la clef de chiffrement au cours du dit chiffrement
- Trouver comment réduire la clef pour la rendre manipulable



Conclusion

- Trouver des méthodes de chiffrement n'est pas simple et la déduction de failles ne peut permettre que d'améliorer des méthodes imaginées, et non d'en créer de nouvelles
- Il faut garder à l'esprit qu'il existe des méthodes bien meilleures que la substitution alphabétique (Hachage elliptique , RSA)

Annexe

- Cesar

```
def cesar_x(x,mess,sens): # 'x' est le décalage
    y=x%26
    N,O,j,z='',[],0,0
    if sens=='dechiffrement':
        z+=-1
    if sens=='chiffrement':
        z+=1
    for i in range(0,len(mess)):
        rang=0
        for j in range(0,len(Alpha)):
            if mess[i]==Alpha[j]:
                rang+=j
        if mess[i]==' ':
            rang+=30
        O.append(rang)
    for k in range (0,len(O)):
        if O[k]==30:
            N+=' '
        else:
            O[k]+=z*y
            N+=Alpha[O[k]%26]
    return N
```

Annexe

- Vigenere

```
def matrice_encodage():
    L,Li=[],[]
    for i in range(26):
        Li=[]
        for j in range(26):
            Li.append(cesar_x(i,Alpha[j],"chiffrement"))
        L.append(Li)
    return(L)

def rang_lettre(element):
    for i in range(26):
        if Alpha[i]==element:
            return(i)
    return(-1)

def indice_liste(element,L):
    for i in range(len(L)):
        if L[i]==element:
            return(i)

def vigenere(Clef,T):
    crypt=''
    SuiteClef=''
    n=len(Clef)
    matrice=matrice_encodage()
    for i in range(len(T)):
        if T[i]==' ':
            SuiteClef=SuiteClef + ' '
        else:
            SuiteClef=SuiteClef + Clef[i % n]
    for i in range(len(T)):
        if T[i]==' ':
            crypt=crypt + ' '
        else:
            crypt=crypt + matrice[rang_lettre(SuiteClef[i])][rang_lettre(T[i])]
    return(crypt)
```

```
def DecryptVigenere(Clef,T):
    decrypt=''
    matrice=matrice_encodage()
    n=len(Clef)
    SuiteClef=''
    for i in range(len(T)):
        if T[i]==' ':
            SuiteClef=SuiteClef + ' '
        else:
            SuiteClef=SuiteClef + Clef[i % n]
    for i in range(len(T)):
        if T[i]==' ':
            decrypt=decrypt + ' '
        else:
            decrypt=decrypt + Alpha[indice_liste(T[i],matrice[rang_lettre(SuiteClef[i])])]
    return(decrypt)
```

Annexe

- HexaEncrypt

```
Alphabet=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
Hexadecimal=['0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F']
A=Alphabet
H=Hexadecimal

#Le "modulo" doit être supérieur ou égal à 5 pour un bon fonctionnement , lorsqu'il est inférieur à 5 , on tombe sur des problèmes d'indices propres à chaque
"modulo" compris entre 1 et 4

#Ecrire ce que le programme doit chiffrer sous forme d'une chaine de caractères , en majuscules , sans espaces ni caractères n'appartenant pas à 'Alphabet'
ci-dessus

Exemple2='NESOMBREPASDOUCEMENTDANSCETTEDOUCENUIT'
m1='3'
m2='9'
m3='D'

import random as rd
```

```
def position(a,L):
    j=0
    while j<len(L) and L[j]!=a :
        j+=1
    return(j)

def newHexa(m1,m2,m3):
    nH=[]
    for i in range (0,len(H)):
        if H[i]!=m1 and H[i]!=m2 and H[i]!=m3:
            nH.append(H[i])
    return(nH)

def transpose_a_h(a,m1,m2,m3):
    K=newHexa(m1,m2,m3)
    j=position(a,A)
    if j <= 12 :
        return(K[j])
    else :
        j=j-13
        r=rd.random()
        if r<(1/3) :
            return(m1+K[j])
        else :
            if r<(2/3) :
                return(m2+K[j])
            else:
                return(m3+K[j])

def transpose_texte_h(T,m1,m2,m3):
    Crypt=''
    for i in range (0,len(T)):
        h=transpose_a_h(T[i],m1,m2,m3)
        Crypt=Crypt + h
    return(Crypt)
```

Annexe

- HexaEncrypt

```
def Dechiffre_Hexa(T,m1,m2,m3):
    i=0
    nH=newHexa(m1,m2,m3)
    Decrypt=''
    while i<len(T):
        if T[i]==m1 or T[i]==m2 or T[i]==m3 :
            Decrypt=Decrypt + Alphabet[13+position(T[i+1],nH)]
            i+=2
        else:
            Decrypt=Decrypt + Alphabet[position(T[i],nH)]
            i+=1
    return(Decrypt)

def transpose_final_texte_h(T,m1,m2,m3,modulo):
    Crypt=''
    i=1
    while i<len(T) :
        if i%modulo == 0 :
            Crypt=Crypt + transpose_texte_h(T[i-modulo:i],m1,m2,m3)
            n=len(Crypt)
            m1=Crypt[n-1]
            m2=Crypt[n-2]
            m3=Crypt[n-3]
        i+=1
    k=len(T)%modulo
    Crypt=Crypt + transpose_texte_h(T[len(T)-k:],m1,m2,m3)
    return(Crypt)
```

```
def Decoupage_String(T,m1,m2,m3,modulo,i):
    Decoupe=''
    j=i
    compteur=0
    while compteur < modulo:
        if T[j]!=m1 and T[j]!=m2 and T[j]!=m3:
            Decoupe=Decoupe + T[j]
            compteur+=1
        else:
            Decoupe=Decoupe + T[j]
        j+=1
    return(Decoupe)
    #String de longueur i sans compter les modulos

def Nombre_De_Decoupages_Possibles(T,m1,m2,m3,modulo):
    compteur=0
    for i in range(len(T)):
        if T[i]!=m1 and T[i]!=m2 and T[i]!=m3:
            compteur+=1
    return(compteur//modulo)

def Dechiffre_final_Hexa(T,m1,m2,m3,modulo):
    j=0
    Decrypt=''
    for i in range(Nombre_De_Decoupages_Possibles(T,m1,m2,m3,modulo) - 1):
        U=Decoupage_String(T,m1,m2,m3,modulo,j)
        j+=len(U)
        Decrypt=Decrypt + Dechiffre_Hexa(U,m1,m2,m3)
        m1=U[len(U)-1]
        m2=U[len(U)-2]
        m3=U[len(U)-3]
    Decrypt=Decrypt + Dechiffre_Hexa(T[j:],T[j - 1],T[j - 2],T[j - 3])
    return(Decrypt)
```

Annexe

- PartiesEnsemble (Utile Pour CodonScript)

```
def SupprimerDansListe(i,L):
    L1=[]
    for j in range(len(L)):
        if j != i :
            L1.append(L[j])
    return(L1)


def PartiesEnsembleAux(L):
    if len(L)==1:
        return([L])
    else :
        L1=[]
        for i in range(len(L)):
            L1=L1 + [SupprimerDansListe(i,L)] + PartiesEnsembleAux(SupprimerDansListe(i,L))
    return(L1)


def EstDansListe(element,L,i,j):
    for i in range(i,j):
        if L[i]==element:
            return(True)
    return(False)


def SupprimerDoublonsDansListe(L):
    L1=[]
    for i in range(len(L)):
        if not(EstDansListe(L[i],L,0,i)):
            L1.append(L[i])
    return(L1)
```

```
def ExtraireListesDeMemeLongueurs(longueur,L):
    L1=[]
    for i in range(len(L)):
        if len(L[i])==longueur:
            L1.append(L[i])
    return(L1)


def MinimumListe(L):
    marqueur=(L[0],0)
    for i in range(1,len(L)):
        (element,indice)=marqueur
        if L[i]<element:
            marqueur=(L[i],i)
    return(marqueur)


def TriNaif(L):
    L1=[]
    L2=L
    for i in range(len(L)):
        (element,indice)=MinimumListe(L2)
        L1.append(element)
        L2=SupprimerDansListe(indice,L2)
    return(L1)


def MaxLongueurListe(L):
    maximum=0
    for i in range(len(L)):
        if len(L[i])>maximum:
            maximum=len(L[i])
    return(maximum)
```

```
def TriInsertionPourPartiesEnsemble(L):
    L1=[]
    for i in range(MaxLongueurListe(L)+1):
        L1.append(TriNaif(ExtraireListesDeMemeLongueurs(i,L)))
    return(L1)


def PartiesEnsemble(L):
    L1=TriInsertionPourPartiesEnsemble(SupprimerDoublonsDansListe(PartiesEnsembleAux(SupprimerDoublonsDansListe(TriNaif(L)))))
    L1[0]=[[]]
    L1.append([TriNaif(L)])
    return(L1)


def MixerEnsemble(L):
    if len(L)==2:
        return([[L[0],L[1]],[L[1],L[0]]])
    else :
        L1=[]
        for i in range(len(L)):
            L1=L1+[MixerEnsemble(SupprimerDansListe(i,L))[0]+[L[i]],MixerEnsemble(SupprimerDansListe(i,L))[1]+[L[i]]]
        return(TriNaif(L1))

        #Permet d'avoir toutes les parties de l'ensemble dans tous les ordres (fonctionnel uniquement pour 3 éléments)
```

Annexe

- CodonScript (Non achevé à cause de la complexité spatiale)

```
def PartiesAlphabetA3Parties():
    L=[]
    for i in range(len(Alphabet)):
        L1=[]
        for j in range(len(Alphabet)):
            L2=[]
            for k in range(len(Alphabet)):
                L2.append([Alphabet[i],Alphabet[j],Alphabet[k]])
            L1.append(L2)
        L.append(L1)
    return(L)


def SupprimerDansListe(i,L):
    L1=[]
    for j in range(len(L)):
        if j != i :
            L1.append(L[j])
    return(L1)


def MinimumListe(L):
    marqueur=(L[0],0)
    for i in range(1,len(L)):
        (element,indice)=marqueur
        if L[i]<element:
            marqueur=(L[i],i)
    return(marqueur)
```

```
def TriNaif(L):
    L1=[]
    L2=L
    for i in range(len(L)):
        (element,indice)=MinimumListe(L2)
        L1.append(element)
        L2=SupprimerDansListe(indice,L2)
    return(L1)


def CreerTenseur():
    L=(PartiesAlphabetA3Parties())
    return(L)


def ArbreDeCompletion():
    L=[]
    for i in range(len(Alphabet)):
        L1=[Alphabet[i],26**2]
        L2=[]
        for j in range(len(Alphabet)):
            L2.append([Alphabet[i],Alphabet[j],26])
            L3=[]
            for k in range(len(Alphabet)):
                L3.append([Alphabet[i],Alphabet[j],Alphabet[k]])
            L2.append(L3)
        L1.append(L2)
        L.append(L1)
    return(L)
```