

Travaux dirigés et TP n° 4 et 5

Chiffrement

Exercice 1 (Chiffrement par décalage)

note : Tous les chiffreurs héritent de la classe CodeurCA.

Chiffrer par un décalage de 3 le mot "Bonjour"

Écrire le code des méthodes bincode et binDecode d'un chiffreur par décalage :

```
def binCode(self, monBinD: Binaire603) -> Binaire603:
    """
    >> ChiffreurParDecalage(2).binCode(Binaire603([ 0x01, 0x02, 0x03, 0x04, 0xFF]))
    Binaire603([ 0x03, 0x04, 0x05, 0x06, 0x01])
    >> Texte603(ChiffreurParDecalage(1).binCode(Texte603("Bonjour").toBinaire603()))
    Texte603("Cpokpvs")
    """
def binDecode(self, monBinC: Binaire603) -> Binaire603:
    """
    >> ChiffreurParDecalage(2).binDecode(Binaire603([1,2,3,4,255]))
    Binaire603([ 0xff, 0x00, 0x01, 0x02, 0xfd])
    """
```

Exercice 2 (Chiffrement Affine)

La classe ChiffreurAffine n'a pas à être détaillée tant elle est évidente d'après le cours et l'exemple de ChiffreurParDecalage vu au TP précédent.

— Écrire à la main les sorties du script Python suivant :

```
monBin=Binaire603([0x00,0x01,0x02,0x010,0x20,0x40,0x80])
for monCodeur in [ChiffreurAffine(3,5),
ChiffreurAffine(1,1), ChiffreurAffine(1,0),
ChiffreurAffine(2,5)]:
    print(f"Codage avec monCodeur :")
    print(" Bin:",monBin)
    monBinC=monCodeur.binCode(monBin)
    print(" Bin Codé:",monBinC)
    monBinD=monCodeur.binDecode(monBinC)
    print(" Bin Décodé:",monBinD)
    print(" monBinD (décodé) est égal à Monbin
    ?",monBinD==monBin)
```

- Quel est l'espace des clés ? Son cardinal ?
- Comparer le chiffrement par décalage avec le chiffrement affine
- Écrire le code de cette classe : on gagnera à utiliser ElmtZnZ.
- Tester sur un Binaire603 mais aussi un Texte603 courts.
- Écrire un algorithme d'attaque de ce chiffre.

Exercice 3 (Permutations)

Afin de revoir les notions sur les permutations et préparer le chiffreur suivant, exécuter d'abord à la main les tests des codes suivants puis créer la classe `Permutation603` avec les méthodes habituelles dont :

```
def __pow__(self, n):
    """
    >> Permutation603([1,2,3,0])**2
    Permutation603([2, 3, 0, 1])
    >> Permutation603([1, 2, 3, 0])**4
    Permutation603([0, 1, 2, 3])
    >> Permutation603([1, 2, 3, 0])**(-1)
    Permutation603([3, 0, 1, 2])
def ordre(self): """Renvoie l'ordre de la permutation
    >> Permutation603([1,2,3,0]).ordre()
    4
    >> Permutation603([1,0,2,3]).ordre()
    2
def permutAlea(n=6):
    """Renvoie une permutation aléatoire sans utiliser shuffle, seulement randint"""
```

Et pour aller plus loin cet exercice facultatif : programmer les deux fonctions suivantes qui permettent de déterminer des clés de codage par permutation à partir d'un simple entier de $[0..n!]$.

```
def permutKieme(k,n=6):
    """Renvoie la kieme permutation de Sn ce qui permet d'associer des clés aux permutations
    >> lr=[Permutation603.permutKieme(k,n=3) for k in range(6)]
    >> le=[[0,1,2],[0,2,1],[1,0,2],[1,2,0],[2,1,0],[2,0,1]]
    >> sum([1 if Permutation603(l) in lr else 0 for l in le])
    6
def numPermutation(self):
    """Renvoie le numéro de la permutation
    >> Permutation603.permutKieme(100,n=5).numPermutation()
    100
```

Exercice 4 (Chiffrement par Permutation)

Exécuter d'abord à la main les tests des codes suivants puis créer la classe `ChiffreurParPermutation` avec les méthodes habituelles :

```
class ChiffreurParPermutation(CodeurCA):
    """ Chiffreur appliquant une permutation à chaque élément du message à coder
    Evidemment il faudrait que la taille des blocs soient sur 128bits plutôt que sur les 8bits
    de nos exemples
    def __init__(self,p):
        """Chiffreur de clé p où p est une permutation de taille 256
        >> mc=ChiffreurParPermutation(Permutation603([(k*0x51)%256 for k in range(256)]))
        >> mc.binCode(Binaire603([1,2,3,4,5]))
        Binaire603([ 0x51, 0xa2, 0xf3, 0x44, 0x95])
        >> mc.binDecode(mc.binCode(Binaire603([1,2,3,4,5])))
        Binaire603([ 0x01, 0x02, 0x03, 0x04, 0x05])
```

- Comparer le chiffrement par permutation avec le chiffrement affine
- Donner un algorithme d'attaque d'un tel chiffrement
- Réaliser cette attaque sur un long texte codé (idéalement donné par un autre groupe)

Exercice 5 (Chiffrement Vigenere)

Le principe de ce chiffrement est que la clé se présente comme un mot ou une phrase dont chaque caractère donne le décalage à appliquer aux caractères successifs.

- Programmer un chiffreur de Vigenere en utilisant la classe `Texte603`.
- Proposer une attaque pour déterminer la longueur de la clé.
- Réaliser une telle attaque sur une clé de 10 caractères

- Facultatif : Réaliser une attaque sans connaissance de la longueur de la clé.

Exercice 6 (Test des bijections sur les octets)

On se propose de tester différentes bijections sur des octets afin de voir leurs capacités à renvoyer un résultat statistiquement « plat », c'est-à-dire dont les caractéristiques générales du message source seront indétectables.

Pour cela, vous créer des classes, toutes dans un même fichier, héritées de la classe `fBijOctetsCA` définies comme suit :

```
class fBijOctetsCA(object):
    "Une classe abstraite de bijection de [0..255]"
    def __init__(self):
        raise NotImplementedError
    def __repr__(self):
        raise NotImplementedError
    def __call__(self, octet):
        """Renvoie l'image de octet par la bijection"""
        raise NotImplementedError
    def valInv(self, octetC):
        "Renvoie l'antécédent de octetC"
        raise NotImplementedError
```

- Programmer une classe `fBijParDecallage` et ajouter à `fBijOctetsCA` une méthode affichant un graphique.
- On pourra reprendre et adapter le code suivant pour construire un graphique illustrant la diffusion d'une bijection d'octets :

```
import matplotlib.pyplot as plt
lx=[-5+0.1*k for k in range(101)]
ly=[x**2 for x in lx]
plt.plot(lx,ly,"-") # où "." pour un graphique point par point
plt.title("La fonction carré")
plt.show()
```

- Faire le test sur une fonction affine.
- Une fonction appliquant un masque.
- Une fonction appliquant une permutation donnée.
- Une fonction affichant un schéma de Feistel sur un nombre d'itérations données. Les clés successives pourront être générées par `"random.randint"` et initialisées par une graine avec `"random.seed(k)"`.

On pourra aussi essayer de voir si ces fonctions possèdent une caractéristique très importante pour éviter les attaques par analyse différentielle : une légère modification de la clé ou du texte à chiffrer provoque des changements importants dans le texte chiffré.

Exercice 7 (Feistel)

Programmer un chiffreur de Feistel.

Exercice 8 (DES)

Crypter à la main, et sur un seul tour `p=0123456789ABCDEF` avec la clé `133457799BBCDFF1`.

Annexe :

Table de conversion de Texte603 <-> Binaire603

| | | | | | | | | | | | | | | | |
|------|------|------|------|-------|------|------|------|------|------|------|------|------|------|------|-------|
| 00:Ä | 01:a | 02:Å | 03:ä | 04:A | 05:a | 06:Ć | 07:ć | 08:Ĉ | 09:ĉ | 0a:Ċ | 0b:ċ | 0c:Č | 0d:č | 0e:Ď | 0f:d' |
| 10:Đ | 11:đ | 12:Ě | 13:ě | 14:Ě | 15:ě | 16:Ě | 17:ě | 18:Ě | 19:ě | 1a:Ě | 1b:ě | 1c:Ĝ | 1d:ĝ | 1e:Ĝ | 1f:ĝ |
| 20: | 21:! | 22:" | 23:# | 24:\$ | 25:% | 26:& | 27:' | 28:(| 29:) | 2a:* | 2b:+ | 2c:, | 2d:- | 2e:. | 2f:/ |
| 30:0 | 31:1 | 32:2 | 33:3 | 34:4 | 35:5 | 36:6 | 37:7 | 38:8 | 39:9 | 3a:: | 3b:; | 3c:< | 3d:= | 3e:> | 3f:? |
| 40:@ | 41:A | 42:B | 43:C | 44:D | 45:E | 46:F | 47:G | 48:H | 49:I | 4a:J | 4b:K | 4c:L | 4d:M | 4e:N | 4f:O |
| 50:P | 51:Q | 52:R | 53:S | 54:T | 55:U | 56:V | 57:W | 58:X | 59:Y | 5a:Z | 5b:[| 5c:\ | 5d:] | 5e:^ | 5f:_ |
| 60:` | 61:a | 62:b | 63:c | 64:d | 65:e | 66:f | 67:g | 68:h | 69:i | 6a:j | 6b:k | 6c:l | 6d:m | 6e:n | 6f:o |
| 70:p | 71:q | 72:r | 73:s | 74:t | 75:u | 76:v | 77:w | 78:x | 79:y | 7a:z | 7b:{ | 7c: | 7d:} | 7e:~ | 7f:f |
| 80:b | 81:B | 82:B | 83:B | 84:b | 85:b | 86:O | 87:C | 88:c | 89:Đ | 8a:D | 8b:Đ | 8c:Đ | 8d:q | 8e:Đ | 8f:Đ |
| 90:E | 91:F | 92:f | 93:G | 94:Y | 95:h | 96:I | 97:İ | 98:K | 99:k | 9a:ı | 9b:İ | 9c:W | 9d:N | 9e:ı | 9f:Θ |
| a0:O | a1:i | a2:ç | a3:f | a4:ı | a5:¥ | a6:ı | a7:S | a8:" | a9:© | aa:a | ab:« | ac:¬ | ad:f | ae:® | af:~ |
| b0:° | b1:± | b2:² | b3:³ | b4:´ | b5:µ | b6:¶ | b7:· | b8:¸ | b9:¹ | ba:º | bb:» | bc:¼ | bd:½ | be:¾ | bf:¿ |
| c0:À | c1:Á | c2:Â | c3:Ã | c4:Ä | c5:Å | c6:Æ | c7:Ç | c8:È | c9:É | ca:Ê | cb:Ë | cc:Ì | cd:Í | ce:Î | cf:Ï |
| d0:Ð | d1:Ñ | d2:Ò | d3:Ó | d4:Ô | d5:Õ | d6:Ö | d7:× | d8:Ø | d9:Ù | da:Ú | db:Û | dc:Ü | dd:Ý | de:Þ | df:ß |
| e0:à | e1:á | e2:â | e3:ã | e4:ä | e5:å | e6:æ | e7:ç | e8:è | e9:é | ea:ê | eb:ë | ec:ì | ed:í | ee:î | ef:ï |
| f0:ð | f1:ñ | f2:ò | f3:ó | f4:ô | f5:õ | f6:ö | f7:÷ | f8:ø | f9:ù | fa:ú | fb:û | fc:ü | fd:ý | fe:þ | ff:ÿ |