# STAGE IREM

3- La machine enigma

Après quelques incidents de frontières (voir " le sceptre d' Ottokar" par Hergé) la Bordurie et la Syldavie ont des relations tendues. Les services secrets Syldaves viennent de vous recruter pour une mission (ultra-secrète) : *briser le code* utilisé par les militaires Bordures.

## Les permutations

La majorité de l'équipe de déchiffrement est virtuose dans l'art de résoudre les takuzus (et autres énigmes). Cependant quelques rappels mathématiques pourront lui être utiles.
Une *permutation* d'un ensemble $E$ est une bijection de $E$ dans $E$. La composée de deux permutations $f, g : E \to E$ est définie par :

$$f \circ g : e \mapsto f(g(e)).$$

On note $\mathrm{Id}_E$ l'application $e \mapsto e$ (qui est bien une permutation), et $f^{-1}$ l' inverse (ou encore la réciproque) de la permutation $f$ :

$$f \circ f^{-1} = f^{-1} \circ f = \mathrm{Id}_E.$$

La loi $\circ$ est associative i.e. pour toutes permutations $f, g, h$

$$f \circ (g \circ h) = (f \circ g) \circ h$$

Pour $E = [0, 25]$, l'ensemble des entiers de 0 à 25, on représente une permutation $p$ par une liste `p` de longueur 26 telle que, pour tout $i \in [0, 25]$, `p[i]` a pour valeur $p(i)$. Par exemple $p : x \mapsto (3*x+2) \pmod{26}$ est représentée par la liste [2,5,8,11,14,17,20,23,0,3,6,9,12,15,18,21,24,1,4,7,10,13,16,19,22,25].
**Exercice 2.1** A quelle condition sur $a, b \in \mathbb{N}$ est-il vrai que $f : [0, 25] \to [0, 25]$ définie par

$$x \mapsto (a * x + b) \pmod{26}$$

est une permutation ?
**Exercice 2.2** Compléter le module de manipulation de permutations esquissé dans le fichier `aide_crypto.py` :

```
##retourne l'inverse de p
def inv(p):
    ...

##teste que p1==p2
##retourne un booleen
def equal(p1,p2):
    ...

##renvoie True ssi p est une permutation
def permut(p):
    ...

##renvoie True ssi p a un point fixe
def apointfixe(p):
    ...
```

## La machine Enigma

**Description** C'est une machine électromécanique (voir par exemple le site www.apprendre-en-ligne.net/crypto/Enigma/) que l'on peut schématiser comme suit [1]. On rentre un message $m = m[0]m[1]\cdots m[i]\cdots m[\ell-1]$ qui est une suite de lettres majuscules dans {'A' ,'B', ...,'Z'}. La lettre $m[i]$ est transformée en une lettre $f_i(m[i])$ par une permutation $f_i$ de {'A' ,'B', ...,'Z'}. La machine renvoie le message chiffré :

$$m' := f_0(m[0])f_1(m[1])\cdots f_i(m[i])\cdots f_{\ell-1}(m[\ell-1]).$$

La machine comporte 3 "rotors" $(\mathsf{RT}_0, \mathsf{RT}_1, \mathsf{RT}_2)$ et un "réflecteur" $\mathsf{REF}$.
La traversée (par un symbole) d'un rotor induit une permutation et la traversée du réflecteur induit une permutation qui est une *involution* ($\mathsf{REF} = \mathsf{REF} \circ \mathsf{REF}$) *sans point fixe* (i.e. $\forall x, \mathsf{REF}(x) \neq x$).
Dans le "réglage" de base de la machine le symbole $x$ en position 0 ($x := m[0]$)
passe à travers le premier rotor, est transformé en $\mathsf{RT}_0(x)$,
qui passe à travers le second rotor, est transformé en $\mathsf{RT}_1(\mathsf{RT}_0(x))$,
qui passe à travers le troisième rotor, est transformé en $\mathsf{RT}_2(\mathsf{RT}_1(\mathsf{RT}_0(x)))$,
qui passe à travers le réflecteur , est transformé en $\mathsf{REF}(\mathsf{RT}_2(\mathsf{RT}_1(\mathsf{RT}_0(x))))$,
qui retraverse, en ordre inverse les 3 rotors, et ressort transformé en le symbole :

$$\mathsf{RT}_0^{-1}(\mathsf{RT}_1^{-1}(\mathsf{RT}_2^{-1}(\mathsf{REF}(\mathsf{RT}_2(\mathsf{RT}_1(\mathsf{RT}_0(x))))))))$$

Autrement dit :

$$f_0 := \mathsf{RT}_0^{-1} \circ \mathsf{RT}_1^{-1} \circ \mathsf{RT}_2^{-1} \circ \mathsf{REF} \circ \mathsf{RT}_2 \circ \mathsf{RT}_1 \circ \mathsf{RT}_0$$

Puis le symbole en position 1 ($x := m[1]$) est traité de la même manière à une nuance près : le premier rotor a "tourné" d'un vint-sixième de tour ; du coup le caractère en sortie est : $\mathsf{RT}_0(x+1)$ etc... et il ressort le symbole :

$$\mathsf{RT}_0^{-1}(\mathsf{RT}_1^{-1}(\mathsf{RT}_2^{-1}(\mathsf{REF}(\mathsf{RT}_2(\mathsf{RT}_1(\mathsf{RT}_0(x+1))))))) - 1$$

À chaque fois que l'on s'apprête à encrypter une nouvelle position $i+1$ du message $m$ , le premier rotor tourne de 1, le second tourne de 1 (si le premier revient à 0) et le troisième tourne de 1 (si le second revient à 0). Cela revient à dire que, si on note $t$ la permutation $x \mapsto x+1 \pmod{26}$, et si $i \equiv a_2 26^2 + a_1 26 + a_0 \pmod{26^3}$

$$f_i := t^{-a_0} \circ \mathsf{RT}_0^{-1} \circ t^{-a_1} \circ \mathsf{RT}_1^{-1} \circ t^{-a_2} \circ \mathsf{RT}_2^{-1} \circ \mathsf{REF} \circ \mathsf{RT}_2 \circ t^{a_2} \circ \mathsf{RT}_1 \circ t^{a_1} \circ \mathsf{RT}_0 \circ t^{a_0} \qquad (1)$$

(on considère ci-dessus que l'on encrypte des entiers de 0 à 25 plutôt que des caractères ; on passe des entiers aux caractères et réciproquement au moyen des bijections `num_to_lett` et `lett_to_num`).
Si la machine est dans la configuration : liste des rotors $=(r_0, r_1, r_2)$ et décalage des rotors$=(d_0, d_1, d_2)$, alors la permutation $f_i$ utilisée pour encrypter le nombre $m[i]$ est

$$f_i := t^{-a_0-d_0} \circ \mathsf{RT}_{r_0}^{-1} \circ t^{-a_1-d_1} \circ \mathsf{RT}_{r_1}^{-1} \circ t^{-a_2-d_2} \circ \mathsf{RT}_{r_2}^{-1} \circ \mathsf{REF} \circ \mathsf{RT}_{r_2} \circ t^{a_2+d_2} \circ \mathsf{RT}_{r_1} \circ t^{a_1+d_1} \circ \mathsf{RT}_{r_0} \circ t^{a_0+d_0} \qquad (2)$$

1. Nous considérons ici une machine simplifiée qui ne comporte pas de *tableau de connexions*

**Simulation** Les Syldaves ont promptement écrit une fonction `crypt(lrot,lpos,mes)` qui simule la machine enigma (voir le fichier `aide_crypto.py`)

```
##cryptage simple
##renvoie le message mes (litteral) crypte par enigma
##avec la suite de rotors lrot,
##et les suite de positions initiales lpos (numeriques)
def crypt(lrot,lpos,mes):
    ...
    return mesk
```

**Exercice 2.3** On affecte :

```
lrot=[1,0,2]
lpos=[23,1,13]
mes="LUNDISEIZEFEVRIERDEUXMILLEQUINZEBULLETINMETEONEIGEAPARTIRDE\\
MILLESIXCENTMETRES"
```

Que renvoie l'appel `crypt(lrot,lpos,crypt(lrot,lpos,mes))` ? Comment expliquez-vous ce phénomène ?
Est-il dû au choix particulier des rotors ? de leurs positions ? du message ? du réflecteur ?
Est-il possible de trouver un message $m$ et une position $i \in [0, |m| - 1]$ telle que son image cryptée $m'$ ait la *même lettre* en position $i$ , i.e. $m[i] = m'[i]$ (pour un certain choix des rotors et de leurs positions) ?

## Propriétés d'un texte

Les Bordures envoient aussi bien des bulletins météo que des informations de mouvements de troupes avec le même système de cryptage. On peut donc espérer détecter dans un message (une fois décrypté) certains mots particuliers comme :
"BULLETINMETEO" ou "MERCREDIDIXHUITFEVRIERDEUXMILLEQUINZE" (en préfixe) ou "VIVEPLECSZYGLADZ" (en suffixe). On développe donc un module de recherche de motifs dans les mots.
**Exercice 2.4** En vous inspirant de la fonction `prefixe(motif,mot)`, écrire des fonctions :

```
##detection d'un mot dans un message (bulletin meteo, date, nom de ville etc...)
##retourne True si motif est facteur de mot, False sinon
def facteur(motif,mot):
    ...

##detection d'un mot en fin de message (vive Plecszy Gladz)
##retourne True si motif est suffixe de mot, False sinon
def suffixe(motif,mot):
    ...
```

## Cryptanalyse 1 : recherche exhaustive

**Trouver la liste des rotors** On suppose ici que l'on connaît des messages cryptés avec enigma dans une configuration (lrot,lpos) inconnue. La première idée qui vient à l'esprit pour trouver "la clé" i.e. le couple (lrot,lpos) est de tester tous les réglages de enigma.
**Exercice 2.5** 1- Compléter la fonction esquissée par l'équipe Syldave.

```
##attaque force-brute
##on a un message crypte mesk (litteral)
##enumeration de toutes les configurations possibles,
##jusqu'a trouver une propriete du message decode qui rend plausible le decodage
##date correcte, signature, un mot-cle , etc ...
##critere est une fonction des messages vers les booleens
##retourne une liste  [[lrot1,lpos1],...,[lroti,lposi],...] (possiblement vide)
def force_brute(NROTORS,mesk,critere):
    ...
```

2- Testez cette fonction et chronométrez-la sur des exemples de messages.

**Le protocole Bordure** L'ensemble des agents Bordures utilisent une liste de rotors du jour `LROTJOUR` qui est une permutation de $[0, 2]$ et une une clé du jour `CLEJOUR` qui est une suite de 3 nombres dans $[0, 25]$.
Chaque agent Bordure, qui souhaite crypter un message `mes`, opère suivant le protocole suivant :
P1- Il choisit (en principe aléatoirement) une clé du message ,`clem`, qui est une suite de 3 nombres dans $[0, 25]$ (elle ne sera utilisée, a priori, que pour ce message)
P2- Il crypte avec `LROTJOUR` et `CLEJOUR` le mot de longueur 6 `clem.clem` (le "carré" de la clé du message), le résultat est le mot `entete` (une suite de 6 caractères entre 'A' et 'Z').
P3- Il crypte avec `LROTJOUR` et `clem` le message `mes` : le résultat est `mesk`.
P4 : Il envoie le mot `entete`+`mesk`

**Exercice 2.6** Le protocole Bordure est simulé par la fonction `crypt_s(lrot,lpos,clem,mes)`. Ecrire la fonction de décryptage associée

```
##decryptage du message mesk (crypte selon le procole officiel, litteral)
##retourne False si l'entete est incorrecte
##retourne le message decode (litteral) sinon
def decrypt_s(lrot,lpos,mesk):
    ...
```

**Trouver la clé du jour** Après avoir déchiffré un seul message Bordure, on connait la liste des rotors du jour `LROTJOUR`. Il ne reste plus qu'à connaître `CLEJOUR` pour décrypter n'importe quel message. Vous connaissez, grace à la fonction `force_brute(NROTORS,mesk,critere)` appliquée à quelques messages de contenu partiellement prévisible, une (petite) liste de couples `[message_en_clair,message_crypte]`.

**Exercice 2.7** Compléter les fonctions esquissées dans le fichier `aide_crypto.py` :

```
##attaque force-brute, connaissant lrot (liste des rotors)
##on a une liste_paires de [cle_message,cle_cryptee]
##enumeration de toutes les configurations possibles,(lrot,lpos)
##jusqu'a interpoler tous les couples
##retourne une liste  [lpos1,...,lposi,...] (possiblement vide)
def force_brute_clrot(lrot,liste_paires):
    ...
```

2- La tester sur une liste de quelques couples `[clem, cryptage de clem]` obtenus en appliquant `force_brute` à des messages Bordures.

## Cryptanalyse 2 : cycle des indicateurs

On s'appuie ici sur la structure particulière de la suite des six premiers symboles de chaque message crypté (cette suite est l'*indicateur* du message). Notons $m$ le carré de la clé du message

(vu comme une liste de nombres), $m'$ son cryptage (vu aussi comme une liste de nombres) et $f : [0, 25] \to [0, 25]$ la permutation utilisée en position 0 i.e.

$$f := t^{-d_0} \circ \mathsf{RT}_{r_0}^{-1} \circ t^{-d_1} \circ \mathsf{RT}_{r_1}^{-1} \circ t^{-d_2} \circ \mathsf{RT}_{r_2}^{-1} \circ \mathsf{REF} \circ \mathsf{RT}_{r_2} \circ t^{d_2} \circ \mathsf{RT}_{r_1} \circ t^{d_1} \circ \mathsf{RT}_{r_0} \circ t^{d_0} \quad (3)$$

On suppose que $0 \leq d_0 \leq 22$.
On remarque que $m'[0]$ (resp. $m'[3]$) encodent le même nombre $m[0]$, l'un au moyen de la permutation $f$, l'autre au moyen de la permutation $t^{-3} \circ f \circ t^3$. On a donc

$$m'[3] + 3 \equiv f \circ t^3 \circ f^{-1}(m'[0]) \pmod{26}$$

En lisant les indicateurs des messages chiffrés Bordures, on peut collecter suffisamment de couples $(m'[0], m'[3] + 3)$, pour connaitre la permutation $f \circ t^3 \circ f^{-1}$.
**Exercice 2.8** 1- Montrer que $g := f \circ t^3 \circ f^{-1}$ est un "cycle" d'ordre 26 i.e. une permutation telle que, pour tout $x \in [0, 25]$ et tout $n \in \mathbb{N}$,

$$g^n(x) = x \Leftrightarrow n \equiv 0 \pmod{26}$$

2- Montrer que $g$ et $f(0)$ déterminent entièrement $f$

On en déduit un algorithme de calcul de $f$ à partir de quelques centaines d'indicateurs (200 indicateurs en général suffisent) :
- on calcule $g$ (que nous appelons le "cycle des indicateurs")
- puis on fait la liste de toutes les valeurs possibles de $f(0)$ (il n'y en a que 26) - pour chacune on ne retient que les $f$ qui sont involutives et sans point fixe.

**Exercice 2.9** Compléter les fonctions esquissées par l'équipe Syldave.

```
##liste_indics une liste de mots de la forme crypt(cle.cle)
##retourne le "cycle des indicateurs"
##c'est une permutation de [0,25]
##si f est la fonction de cryptage sur la lettre de numero 0
##le cycle est la permutation: f \odot trans(3) \odot f^{-1}
def cycle(liste_indics):
    ...

##calcul de la fonction de cryptage (du jour) f: [0,25] |-> [0,25]
##f0 est la valeur de f sur l'argument 0; tau est le "cycle des indicateurs"
##retourne une permutation de [0,25]
def pcrypt(f0,tau):
    ...

##cyclindics est le "cycle des indicateurs"
##mesk_s est le message crypte , suivant la procedure officielle (crypt_s)[litte
##critere est une fonction des messages vers les booleens
##retourne une valeur [lrot,permjour] satisfaisant le critere
##(applique au message en clair)
def perm_du_jour(cyclindics,mesk_s,critere):
    ...
```

Comment peut-on trouver le couple `[LROTJOUR,CLEJOUR]` avec les fonctions ci-dessus et la donnée d'environ 200 messages Bordures cryptés ? Mettez en oeuvre cette méthode et chronométrez-la. Les Bordures vont-ils attaquer ? où et quand ?

**Épilogue** Il se trouve que ce jour-là, $d_0 \in [0, 22]$.
Qu'aurait-on pu faire dans le cas où $d_0 \in \{24, 25\}$ ? peut-on adapter immédiatement la méthode du cycle des indicateurs ?
Et dans le cas où $d_0 = 23$ ? ?