Chapitre 2 : Cryptographie classique

Le texte des parties I et II de ce chapitre est tiré essentiellement d'un cours du Professeur Evangelos Kranakis (Carleton University of Ottawa) trouvé sur Internet, ce qui explique les exemples avec texte clair en anglais. ftp://ftp.shore.net/member/ws/Support/CryptLect/crypto00.pdf

Petit glossaire (Sécurité informatique avril 1999 N°24 http://www.cnrs.fr/Infosecu/)
chiffrer : transformer à l'aide d'une convention secrète, appelée clé, des informations claires en informations inintelligibles pour des tiers n'ayant pas la connaissance du secret.
Déchiffrer : retrouver les informations claires, à partir des informations chiffrées en utilisant la convention secrète de chiffrement.
Décrypter : retrouver l'information intelligible, à partir de l'information chiffrée sans utiliser la convention secrète de chiffrement.
Par contre, crypter ou encrypter n'a pas de sens clairement défini, mais sont parfois utilisés à tort comme synonymes de chiffrer.

stéganographie : mode de communication secrète obtenu en dissimulant l'existence du message.
Code : on substitue un mot entier
chiffre : on substitue lettre par lettre.

Les chiffrements classiques sont divisés en trois catégories :

- chiffrement monoalphabétique appelé ainsi parce que l'on fait une seule correspondance entre lettres claires et chiffrées.
- Chiffrement polyalphabétique ainsi appelé parce qu'à chaque lettre du texte clair correspond plusieurs lettres chiffrées dépendant de la position de la lettre à chiffrer dans le texte.
- Chiffrement en continu : ainsi appelé parce qu'on génère le texte chiffré en continu.

I) Chiffrement monoalphabétique

1°) Les chiffres de substitution

L'espace des clés est constitué par l'ensemble des substitutions des 26 lettres de l'alphabet. Pour une substitution π donnée :

$$E_\pi(x_1,x_2,x_3,\ldots,x_n) = \pi(x_1)\pi(x_2)\pi(x_3,)\ldots\pi(x_n)$$

et

$$D_\pi(x_1,x_2,x_3,\ldots,x_n) = \pi^{-1}(x_1)\pi^{-1}(x_2)\pi^{-1}(x_3,)\ldots\pi^{-1}(x_n)$$

Exemple 1 : chiffrage par décalage (ou de César)

C'est un chiffrement par substitution avec comme permutation :

x -> p(x) = x+b mod(26)

pour un $0 \leqslant b$ < 26

Exemple 2 : chiffrage linéaire

C'est un chiffrement par substitution avec comme permutation :

x -> p(x) = ax+b mod(26)

pour un $0 \leqslant b, a$ < 26 et PGCD(a,26) = 1

Exemple pratique chiffre de César

Soit le texte codé :

L FDPH L VDZ L FRQTXHUHG

Il y a 26 clés possibles donc on peut les essayer toutes :
on trouve la substitution :

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
defghijklmnopqrstuvwxyzabc
```

qui donne :

I came I saw I conquered

Le chiffre de César se casse facilement avec une attaque « texte chiffré seul »

2°) Méthodes d'attaque

- attaque texte chiffré seul (cyphertext-only) : l'attaquant possède une chaîne chiffrée y.
- attaque texte clair connu (known plaintext) : l'attaquant possède une copie du texte clair x et une copie du texte chiffré y.
- attaque texte clair choisi (chosen plaintext): l'attaquant possède temporairement un accès à la machine de chiffrement. Il peut choisir librement un texte x et le chiffrer en y.
- attaque texte chiffré choisi (chosen cyphertext) : l'attaquant possède temporairement un accès à la machine de déchiffrement et peut choisir des textes codés y et les décoder.
- Attaque par force brute : on essaie toutes les clés possibles.
- Attaque par devinette : supposons un espace de clé défini. Si un utilisateur utilise un sous espace seulement (par exemple seulement des mots français) on peut facilement faire une recherche sur les mots français. En calculant l'entropie de ces clés on peut montrer que cet espace des mots français n'est pas exceptionnel.

3°) Les types de sécurité

Il y a typiquement deux manières de sécuriser un code :

Sécurité inconditionnelle : peu importe la puissance de l'ordinateur dont vous disposez le chiffre ne peut pas être cassé.

Sécurité informatique (Computational Security) : elle signifie une des deux choses suivantes : Étant donné les ressources informatiques limitées (c'est à dire que le temps de calcul est de l'ordre de grandeur de l'âge de l'univers) le chiffre ne peut pas être cassé.

Prouvé sécurisé (provably secure) : le problème algorithmique peut se réduire à un problème réputé algorithmiquement complexe

4°) Analyse des fréquences

Dans tous les langages les lettres apparaissent avec des fréquences différentes : par exemple le e est la lettre la plus employée du français et de l'anglais.
Nous présentons ci-dessous les fréquences des lettres et des groupes de lettres. Ces fréquences sont calculées à partir d'articles de journaux ou de livres.

Français			
A	9,42%	N	7,15%
B	1,02%	O	5,14%
C	2,64%	P	2,86%
D	3,39%	Q	1,06%
E	15,87%	R	6,46%
F	0,95%	S	7,90%
G	1,04%	T	7,26%

Anglais			
E	12,70%	TH	THE
T	9,10%	HE	ING
A	8,20%	IN	AND
O	7,50%	ER	HER
I	7,00%	AN	ERE
N	6,70%	RE	ENT
S	6,30%	ED	THA

Français			
H	0,77%	U	6,24%
I	8,41%	V	2,15%
J	0,89%	W	0,00%
K	0,00%	X	0,30%
L	5,34%	Y	0,24%
M	3,24%	Z	0,32%

Anglais			
H	6,10%	ON	NTH

5°) Cryptanalyse des chiffres affines

Nous ne nous intéressons qu'aux attaques texte chiffré seulement. Considérons le texte chiffré suivant :

FMNVEDKAPHFERBNDKRX
RSREFMORUDSDKDVSHVU
FEDKAPRKDLYEVLRHHRH

En regroupant les fréquences d'apparition par lettres, nous obtenons par ordre décroissant :

Lettre	*Nombre d'occurrences*
R	8
D	6
E	5
H	5
K	5
V	4
F	4

Ensuite on fait une conjecture. Pour chaque conjecture on choisit deux lettres candidates potentiels et essayons de montrer que cette conjecture amène du sens. Utilisant notre conjecture nous calculons a et b. Cela implique la résolution d'un système linéaire constitué de deux congruences et de deux inconnues a et b.
Première conjecture : R -> e et D ->t , c'est à dire $E_k(19)=5$ et $E_k(4)=17$ où $E_k(x)=ax+b \bmod 26$
On doit donc résoudre : $4a+b=17 \bmod 26$ et $19a+b=5 \bmod 26$ qui nous donne a=6 et b=19, solution impossible à garder car PGCD(13,26) > 1.
Deuxième conjecture : R -> e et E -> t. On obtient de même a=13 qui est illégal car PGCD(13,26)>1.
Troisième conjecture : R -> e et H -> t. La résolution donne a=3 et b=5. La fonction d'encryption est donc $E_k(x) = 3x+5 \bmod 26$
L'opération de déchiffrement peut s'en déduire facilement : $D_k(y) = 9y-19 \bmod 26$. Si nous essayons sur le texte chiffré nous obtenons bien le texte clair.

6°) Cryptanalyse des chiffrements de substitution

Nous ne nous intéressons qu'aux attaques texte chiffré seulement. On regroupe les fréquences d'apparition par lettres puis on fait la même chose sur les groupes de deux puis de trois lettres. On conjecture que la lettre la plus fréquente est 'e' puis on utilise une analyse exaustive par essais et erreurs. On regarde ensuite les motifs de deux lettres puis de trois pour voir si c'est consistant avec nos choix précédents. A chaque étape on vérifie si notre supposition est consistante et si elle aboutit à un texte ayant un sens.
Même s'il y a 26! ($=2.10^{26}$) permutations possibles cette analyse des fréquences permet une cryptanalyse même sur des textes courts (200 caractères).

7°) La force de la cryptanalyse

Nous allons montrer comment déchiffrer le texte :

```
YIFQFMZRWQFYVECFMDZPCVMRZWNMDZVEJBTXCDDUMJ
NDIFEFMDZCDMQZKCEYFCJMYRNCWJCSZREXCHZUNMXZ
NZUCDRJXYYSMRTMEYIFZWDYVZVYFZUMRZCRWNZDZJJ
XZWGCHSMRNMDHNCMFQCHZJMXJZWIEJYUCFWDJNZDIR
```

Lettre	*Fréquence*	*Lettre*	*Fréquence*	*Lettre*	*Fréquence*	*Lettre*	*Fréquence*
A	0	B	1	C	15	D	13
E	7	F	11	G	1	H	4
I	5	J	11	K	1	L	0
M	16	N	9	O	0	P	1
Q	4	R	10	S	3	T	2
U	5	V	5	W	8	X	6
Y	10	Z	20				

Le tableau des fréquences nous permet de conjecturer que Z -> e. Les autres caractères apparaissant plus de dix foix sont C, D, F, J, M, R, Y. Nous supposons qu'ils peuvent se déchiffrer en un sous-ensemble de t,a,o,i,n,s,h,r.
Ensuite nous cherchons les motifs -Z et Z-. En les comptant nous trouvons que DZ et ZW arrivent quatre fois chacun et NZ et ZU trois fois. En regardant la fréquence des motifs de deux lettres, comme ZW arrive quatre fois mais pas WZ et que W arrive moins que n'importe quel autre caractère nous supposons que W->d. Comme DZ arrive quatre fois et ZD deux fois on peut supposer que D->r,s,t mais il n'est pas clair lequel de ces trois doit-on prendre.
En supposant Z -> e et W -> d nous regardons de nouveau le message codé et remarquons que ZRW et RZW arrivent près du début et RW un peu plus tard dans ce message. Comme RW arrive fréquemment et nd est courant nous conjecturons que R -> n.
A ce point nous pouvons regarder ce que cela donne :

```
------end---------e----ned---e------------
YIFQFMZRWQFYVECFMDZPCVMRZWNMDZVEJBTXCDDUMJ
--------e----e---------n--d---en----e----e
NDIFEFMDZCDMQZKCEYFCJMYRNCWJCSZREXCHZUNMXZ
-e---n------n------ed---e---e--ne-nd-e-e--
NZUCDRJXYYSMRTMEYIFZWDYVZVYFZUMRZCRWNZDZJJ
-ed------n----------e----ed-------d---e--n
XZWGCHSMRNMDHNCMFQCHZJMXJZWIEJYUCFWDJNZDIR
```

Comme NZ est un motif courant et que ZN ne l'est pas une autre supposition que l'on fait est N->h.
En supposant que cela est correct le texte ne-ndhe suggère que C -> a. Nous obtenons donc :

```
------end-----a---e-a--nedh--e------a-----
YIFQFMZRWQFYVECFMDZPCVMRZWNMDZVEJBTXCDDUMJ
h-------ea---e-a---a---nhad-a-en--a-e-h--e
NDIFEFMDZCDMQZKCEYFCJMYRNCWJCSZREXCHZUNMXZ
he-a-n------n------ed---e---e--neandhe-e--
NZUCDRJXYYSMRTMEYIFZWDYVZVYFZUMRZCRWNZDZJJ
-ed-a---nh---ha---a-e----ed-----a-d--he--n
XZWGCHSMRNMDHNCMFQCHZJMXJZWIEJYUCFWDJNZDIR
```

Considérons maintenant M, la deuxième lettre la plus fréquente. La correspondance KNM -> nh- suggère que h- commence un mot. Ainsi probablement M est une voyelle, c'est à dire M -> i, o. Comme ai est plus vraissemblable que ao le motif codé CM suggère M->i. Cela donne :

```
-----iend-----a-i-e-a-inedhi-e------a---i-
YIFQFMZRWQFYVECFMDZPCVMRZWNMDZVEJBTXCDDUMJ
h-----i-ea-i-e-a---a-i-nhad-a-en--a-e-hi-e
NDIFEFMDZCDMQZKCEYFCJMYRNCWJCSZREXCHZUNMXZ
he-a-n-----in-i----ed---e---e-ineandhe-e--
NZUCDRJXYYSMRTMEYIFZWDYVZVYFZUMRZCRWNZDZJJ
-ed-a--inhi--hai--a-e-i--ed-----a-d--he--n
XZWGCHSMRNMDHNCMFQCHZJMXJZWIEJYUCFWDJNZDIR
```

Quelle lettre chiffre le o ? Puisque o est une lettre commune nous supposons que cela peut être une des lettres D, F, J, Y. Pour éviter les chaînes de voyelles il apparaît que Y -> o est le meilleur choix. Sur les tettres restantes nous conjecturons D, F, J -> r, s, t. Nous supposons F -> r et J -> t. A ce point nous en sommes à :

```
o-r-riend-ro--arise-a-inedhise--t---ass-it
```

```
YIFQFMZRWQFYVECFMDZPCVMRZWNMDZVEJBTXCDDUMJ
hs-r-riseasi-e-a-orationhadta-en--ace-hi-e
NDIFEFMDZCDMQZKCEYFCJMYRNCWJCSZREXCHZUNMXZ
he-asnt-oo-in-i-o-redso-e-ore-ineandhesett
NZUCDRJXYYSMRTMEYIFZWDYVZVYFZUMRZCRWNZDZJJ
-ed-ac-inhischair-aceti-ted--to-ardsthes-n
XZWGCHSMRNMDHNCMFQCHZJMXJZWIEJYUCFWDJNZDIR
```

et finalement :
our friend from Paris examined his empty glass with surprise, as if evaporation had taken place while he wasn't looking i poured some more wine and he settled back in his chair face tilted up towards the sun.

8°) Chiffre de Beaufort

Inventé par Sir francis Beaufort, c'est le précurseur du chiffre de Vigenère. Le principe est d'arranger l'alphabet anglais (ou français) dans un carré 27x27. La clé est dérivée d'un nom, d'un poème et satisfait les deux utilisateurs et est appliquée comme suit :
trouver la première lettre du message dans la première colonne. De cette lettre tracer horizontalement jusqu'à trouver la première lettre de la clé et trouver la lettre de codage en haut. Il faut inverser ces étapes pour le déchiffrement.

II) Chiffres polyalphabétiques

1°) Chiffre de Hill

On découpe d'abord le message en blocs de n lettres. L'espace des clés consiste en matrices n x n inversibles. Si k est une matrice n x n inversible alors :

$E_k(x) = k.x, \quad D_k(y) = k^{-1}.y$

On doit aussi s'assurer que l'arithmétique soit faite dans un corps modulo un nombre premier. On choisit en général deux codes :

- A-Z représentés par 0-25, l'espace par 26, la virgule 27 et le stop 28 ce qui fait 29 symboles.
- Alphabet alphanumérique Σ de taille 37 : 0-9 pour les nombres décimaux, le blanc par 10, et les 26 lettres de l'alphabet par 11-36.

Par exemple si l'on travaille avec le corps de 37 éléments avec une taille de bloc de n=2 et si la matrice clé est

$$k = \begin{pmatrix} 3 & 13 \\ 22 & 15 \end{pmatrix}$$

le message GOOD est d'abord écrit comme 17, 25, 25, 14 puis

$$\begin{pmatrix} 3 & 13 \\ 22 & 15 \end{pmatrix}\begin{pmatrix} 17 & 25 \\ 25 & 14 \end{pmatrix} = \begin{pmatrix} 6 & 35 \\ 9 & 20 \end{pmatrix}$$

et donc le cryptogramme est C=6, 9, 35, 20

2°) Chiffrement par transposition

Transposition d'ordre d
Pour tout entier d on divise le message M en blocs de longueur d. Puis on prend une permutation p de 1, 2, ..., d et appliquons π à chaque bloc. Par exemple si π = (4 1 3 2 5) un message tel que :

M = JOHN |IS A |GOOD |SKIER

est transformé en

C=ONHJ |SA I |ODOG |KEISER

Le décodage se fait avec la permutation π^{-1}.

Pour une permutation π soit $k^{\pi}_{i,j}=1$ si $\pi(i)=j$ et $k^{\pi}_{i,j}=0$ autrement. C'est un chiffre de Hill avec la matrice $k^{\pi} = (k^{\pi}_{i,j})$.

3°) Chiffre de Vigenere

Soit $k = (k_1, k_2, ..., k_m)$ un vecteur de m éléments. Alors :

$E_k(x) = x + k$, et $D_k(y) = y - k$.

Cryptanalyse du chiffrement de Vigenere.
On considère l'attaque texte chiffré seul.
On considère d'abord deux techniques pour calculer la taille du bloc m

L'observation de Kasiski : deux motifs identiques de texte clair seront chiffrées de la même manière si leur position est x tel que x = 0 mod m.
On enregistre les distances des mêmes motifs et si ces distances sont notées d1, d2, ... alors on peut supposer que m divise le pgcd de $d_1, d_2, ...$
Le test de Friedman utilise les indices de coïncidence : soit $I_C(x)$ la probabilité que deux éléments aléatoires de n lettres x soient identiques.
Soient $f_0, f_1, ..., f_{25}$ les différentes probabilités de A, B, ..., Z, respectivement dans le texte chiffré.

$$I_C(x) = \frac{\sum_{i=0}^{i=25} \binom{f_i}{2}}{\binom{n}{2}} = \frac{\sum_{i=0}^{i=25} f_i(f_i - 1)}{n(n-1)}$$

En se servant de la table des fréquences des lettres on peut avoir la valeur de $I_C(x)$. p_i^2 est la probabilité que dans la chaîne x deux éléments choisis au hasard sont égaux à la ième lettre.
Ici p_i est la probabilité attendue de la ième lettre dans le texte anglais.
Maintenant le texte anglais et aléatoire diffèrent :

Texte chiffré	*Texte anglais*	*Texte aléatoire*
$I_C(x)$ dans la formule (1)	$\sum_{i=0}^{i=25} p_i^2 = 0.065$	$\sum_{i=0}^{i=25} p_i^2 = 26(1/26)^2 = 0.038$

Méthode de Friedman : conjecturer m. partitionner le texte chiffré en m colonnes $y_1, y_2, ... y_m$ chacune de longueur n/m. Si notre supposition est correcte alors $I_C(y_i) \approx 0.065$ tandis que si m n'est pas correct $I_C(y_i) \approx 0.038$

Exemple

Considérons le texte chiffré obtenu par chiffrement Vigenere.

```
CHREEVOAHMAERATBIAXXWTNXBEEOPHBSBQMQEQERBW
RVXUOAKXAOSXXWEAHBWGJMMQMNKGRFVGXWTRZXWIAK
LXFPSKAUTEMNDCMGTSXMXBTUIADNGMGPSRELXNJELX
VRVPRTULHDNQWTWDTYGBPHXTFALJHASVBFXNGLLCHR
ZB
```

La méthode de Kasiski donne : le motif « CHR » apparaît à quatre places aux positions 1,166,236,286. Les distances de la première apparition sont donc 165, 235, 285 Comme le PGCD (165,235,285) = 5 on conjecture une clé de taille 5.
La méthode de Frieman donne : on calcule l'indexe de coïncidence du texte chiffré donné.
Pour chaque supposition de m on a une taille de colonne de n/m. Ainsi nous avons la taille de 5 comme meilleure supposition :

M	*IC*				
1	0.045				
2	0.046	0.041			
3	0.043	0.050	0.047		
4	0.042	0.039	0.046	0.040	
5	0.063	0.068	0.069	0.061	0.072

Supposons donc que nous connaissions la taille de la clé. On définit l'indice de coïncidence mutuelle comme $MI_C(x,x')$ comme la probabilité qu'un élément aléatoire de x soit identique à un élément aléatoire de x'. Soient $f_0, f_1, ..., f_{25}, f'_0, f'_1, ..., f'_{25}$ les fréquences d'apparition de A, B, ..., Z respectivement dans les chaînes x et x'. On a évidemment :

$$MI_C(x, x') = \frac{\sum_{i=0}^{i=25} f_i f'_i}{n.n'}$$

Comme nous connaissons m on peut diviser le texte chiffré en m blocs $y_1, y_2, ..., y_m$. En supposant $K = (k_1, k_2, ..., k_m)$ est la clé étant utilisée on peut estimer $MI_C(y_i,y_j)$.
Prenons un caractère aléatoire dans y_i et un caractère aléatoire dans y_j. La probabilité que les deux caractères soient A est $p_{-ki}.p_{-kj}$. La probabilité que les deux caractères soient B est $p_{1-ki}.p_{1-kj}$, etc.
Ainsi

$$MI_C \approx \sum_{h=0}^{h=25} p_{h-ki}\, p_{h-kj} = \sum_{h=0}^{h=25} p_h\, p_{h+k_i-k_j.}$$

Noter que la coïncidence mutuelle dépend de la différence relative k_j-k_i. De plus la coïncidence mutuelle du décalage relatif k_i-k_j est la même que celle de k_j-k_i.

III) Le cas ENIGMA

La machine ENIGMA mérite une attention toute particulière pour plusieurs raisons. La première est qu'elle va nous permettre de présenter la notion de protocole utilisée dans la deuxième partie.
Pour présenter les protocoles, nous allons utiliser certaines notations que nous présentons ici.
Définitions :

- Une spécification Alice et Bernard est une séquence d'assertions de la forme A ->B:M où A et B sont processus (ou des machines) et M est un message.
- Une spécification Alice et Bernard annotée est une séquence d'assertions de la forme A->B:$T_1,...T_k$ ||M || $O_1,...O_n$
- où A et B sont des processus, M un message, T_1 ... T_k et $O_1....O_n$ sont des séquences d'opérations réalisées par A et B respectivement.

Mais commençons d'abord par la description de la machine.

1°) La machine Enigma

La machine Enigma est composée de trois rotors (appelés brouilleurs) un réflecteur et un tableau de connexion à fiches entre le clavier et le premier brouilleur.

Nous donnons un programme en C trouvé sur Internet permettant une simulation complète de la machine Enigma. Ce programme a été modifié pour plusieurs raisons :

- La première est que tel qu'il a été téléchargé il ne pouvait pas fonctionner à cause de la mauvaise utilisation de variables locales. Les déclarer en statique résout ce problème.
- La deuxième est que je tenais à faire une machine Enigma compatible à celle qui est décrite dans le livre de S. Singh qui est le seul que j'ai lu sur le sujet : les rotors tournent après l'encodage du caractère. Était-

ce comme cela en réalité, un spécialiste de la question pourrait me le dire.
Ce programme permet aussi de réaliser une « bombe » machine à décoder Enigma. Nous le modifierons quand même dans la partie cryptanalyse.

```
/* enigma simulation and bombe, harald schmidl, april 1998
  the encoding scheme uses code from Fauzan Mirza's
  3 rotor German Enigma simulation
  Written by  */
/* modifie par l'auteur le 05/02/00
  les auteurs ne connaissaient pas la notion de variable
  statique ce qui posait problème dans le prgm original */
#include <stdio.h>
#define MSGLEN 80
#define TO 'E'
/* Rotor wirings */
char rotor[5][26]={
       /* Input "ABCDEFGHIJKLMNOPQRSTUVWXYZ" */
       /* 1: */ "EKMFLGDQVZNTOWYHXUSPAIBRCJ",
       /* 2: */ "AJDKSIRUXBLHWTMCQGZNPYFVOE",
       /* 3: */ "BDFHJLCPRTXVZNYEIWGAKMUSQO",
       /* 4: */ "ESOVPZJAYQUIRHXLNFTGKDCMWB",
       /* 5: */ "VZBRGITYUPSDNHLXAWMJQOFECK" };
char ref[26]="YRUHQSLDPXNGOKMIEBFZCWVJAT";
char notch[5]="QEVJZ";
/* Encryption parameters follow */
typedef struct P {
  char order[3];/*={ 1, 2, 3 };*/
  char rings[3];/*={ 'A','A','A' };*/
  char pos[3];/*={ 'A','A','A' };*/
  char plug[10];/*="AMTE"; */
} Params;
/*take a char and return its encoded version according to the
  encryption params, update params, i.e. advance wheels
  this part uses Fauzan Mirza's code*/
char scramble(char c, Params *p) {
  int i, j;
  static flag = 0; /*doit être déclaré en statique */
            c=toupper(c);
            if (!isalpha(c))
                  return -1;
            /* Step up first rotor */
            p->pos[0]++;
            if (p->pos[0]>'Z')
                  p->pos[0] -= 26;
            /* Check if second rotor reached notch last time */
            if (flag){ /* pourquoi on change seulement au passage suivant */
                  /* Step up both second and third rotors */
                  p->pos[1]++;
                  if (p->pos[1]>'Z')
                        p->pos[1] -= 26;
                  p->pos[2]++;
                  if (p->pos[2]>'Z')
                        p->pos[2] -= 26;
                  flag=0;
            }
            /*  Step up second rotor if first rotor reached notch */
            if (p->pos[0]==notch[p->order[0]-1]){
                  p->pos[1]++;
                  if (p->pos[1]>'Z')
                        p->pos[1] -= 26;
                  /* Set flag if second rotor reached notch */
                  if (p->pos[1]==notch[p->order[1]-1])
                        flag=1;
            }
            /*  Swap pairs of letters on the plugboard */
            for (i=0; p->plug[i]; i+=2){
                  if (c==p->plug[i])
                        c=p->plug[i+1];
                  else if (c==p->plug[i+1])
                        c=p->plug[i];
            }
            /*  Rotors (forward) */
            for (i=0; i<3; i++) {
                  c += p->pos[i]-'A';
                  if (c>'Z')
                        c -= 26;
                  c -= p->rings[i]-'A';
                  if (c<'A')
                        c += 26;
                  c=rotor[p->order[i]-1][c-'A'];
                  c += p->rings[i]-'A';
```

```
                if (c>'Z')
                        c -= 26;
                c -= p->pos[i]-'A';
                if (c<'A')
                        c += 26;
        }
        /*  Reflecting rotor */
        c=ref[c-'A'];
        /*  Rotors (reverse) */
        for (i=3; i; i--)   {
                c += p->pos[i-1]-'A';
                if (c>'Z')
                        c -= 26;
                c -= p->rings[i-1]-'A';
                if (c<'A')
                        c += 26;
                for (j=0; j<26; j++)
                        if (rotor[p->order[i-1]-1][j]==c)
                                break;
                c=j+'A';
                c += p->rings[i-1]-'A';
                if (c>'Z')
                        c -= 26;
                c -= p->pos[i-1]-'A';
                if (c<'A')
                        c += 26;
        }
        /*  Plugboard */
        for (i=0; p->plug[i]; i+=2){
                if (c==p->plug[i])
                        c=p->plug[i+1];
                else if (c==p->plug[i+1])
                        c=p->plug[i];
        }
  return c;
}

/*take a string, return encoded string*/
char *enigma(char *in, Params *p){
  int j;
  static char out[MSGLEN]; /* ajouté static autrement ???? */
  for(j = 0; j < strlen(in); j++)
  out[j] = scramble(in[j], p);
  out[j] = '\0';
  return out;
}

/*read in a string, and pass it through enigma*/
void cypher(Params p){
  char in[MSGLEN], s[MSGLEN];
  int c, i = 0;
  while((c = getchar()) != '\n'){
    in[i] = toupper(c);
    i++;
  }
  in[i] = '\0';
  strcpy(s, enigma(in, &p));
  printf("\n%s\n%s\n", s, in);
}

/*given a cipher text, and a crib, test all possible settings of wheel order a, b, c*/
void rotate(int a, int b, int c, char *cyph, char *crib, char *plug, int *ct){
  Params p;
  p.order[0] = a;
  p.order[1] = b;
  p.order[2] = c;
  p.rings[0] = p.rings[1] = p.rings[2] = 'A';
  strcpy(p.plug, plug);
  for(p.pos[0] = 'A'; p.pos[0] <= 'Z'; p.pos[0]++) {
    for(p.pos[1] = 'A'; p.pos[1] <= 'Z'; p.pos[1]++){
      for(p.pos[2] = 'A'; p.pos[2] <= 'Z'; p.pos[2]++){
/*        for(p.rings[0] = 'A'; p.rings[0] <= 'Z'; p.rings[0]++){
          for(p.rings[1] = 'A'; p.rings[1] <= 'Z'; p.rings[1]++){
            for(p.rings[2] = 'A'; p.rings[2] <= 'Z'; p.rings[2]++){
*/           Params cp = p;
             int i = 0;
             while(strlen(crib) > i){
              if(cyph[i] != scramble(crib[i], &cp))
              break;
              else
              i++;
             }
```

```
            if(strlen(crib) == i){
             char s[MSGLEN];
             (*ct)++;
              printf("Wheels %d %d %d Start %c %c %c Rings %c %c %c Stecker \"%s\"\n",
                       p.order[0], p.order[1], p.order[2],
                       p.pos[0], p.pos[1], p.pos[2],
                       p.rings[0], p.rings[1], p.rings[2], p.plug);
             cp = p;
             strcpy(s, enigma(cyph, &cp));
                printf("%s decoded -> %s\n", cyph, s);
              }
/*            }
          }
        }
*/      }
    }
  }
}
/*do the whole check including steckering of up to two pairs of letters*/
void test(int a, int b, int c, char *cyph, char *crib, int *ct){
  char A, B, C, D;
  int i = 0, cs;
  char s[5];
  strcpy(s, "");
  printf("Checking wheels %d %d %d\n",  a, b, c);
  for(cs = 0; cs < 3; cs++) {
    if(cs > 0){
      for(A = 'A'; A <= TO; A++){
        for(B = A + 1; B <= TO; B++){
         s[0] = A;
         s[1] = B;
         s[2] = '\0';
         if(cs > 1) {
           for(C = A + 1; C <= TO; C++){
             if(C == B)
               continue;
             for(D = C + 1; D <= TO; D++){
               if(D == A || D == B)
                  continue;
              i++;
              s[2] = C;
              s[3] = D;
              s[4] = '\0';
              rotate(a, b, c, cyph, crib, s, ct);
             }
            }
          }
         else
         rotate(a, b, c, cyph, crib, s, ct);
        }
      }
    }
    else
    rotate(a, b, c, cyph, crib, s, ct);
  }
}
/*run on all permutations of wheels a, b, c*/
void permute(int a, int b, int c, char *cyph, char *crib, int *ct){
  test(a, b, c, cyph, crib, ct);
  test(a, c, b, cyph, crib, ct);
  test(b, a, c, cyph, crib, ct);
  test(b, c, a, cyph, crib, ct);
  test(c, a, b, cyph, crib, ct);
  test(c, b, a, cyph, crib, ct);
}
/*all triples of five possible wheels*/
void permuteAll(char *cyph, char *crib){
  int ct = 0;
  permute(1, 2, 3, cyph, crib, &ct);
  permute(1, 2, 4, cyph, crib, &ct);
  permute(1, 2, 5, cyph, crib, &ct);
  permute(1, 3, 4, cyph, crib, &ct);
  permute(1, 3, 5, cyph, crib, &ct);
  permute(1, 4, 5, cyph, crib, &ct);
  permute(2, 3, 4, cyph, crib, &ct);
  permute(2, 3, 5, cyph, crib, &ct);
  permute(2, 4, 5, cyph, crib, &ct);
  permute(3, 4, 5, cyph, crib, &ct);
  printf("\nFound %d solutions.\n", ct);
}
/*helper to read a character*/
char readCh(){
```

```
  char c, ret;
  while((c = getchar()) != '\n')
  ret = c;
  return ret;
}
/*init the starting position*/
void initParams(Params *p){
  int i;
  char c;

  printf("d)efault or u)ser: ");
  c = readCh();
  if(c != 'u'){
    for(i = 0; i < 3; i++){
      p->order[i] = i + 1;
      p->rings[i] = 'A';
      p->pos[i] = 'A';
    }
    strcpy(p->plug, "");
  }
  else{
      for(i = 0; i < 3; i++){
        printf("Wheel %d: ", i + 1);
        p->order[i] = readCh() - 48;
      }
      for(i = 0; i < 3; i++){
/*        printf("Ring  %d: ", i + 1);*/
        p->rings[i] = 'A';/*readCh();*/
      }
      for(i = 0; i < 3; i++){
        printf("Start %d: ", i + 1);
        p->pos[i] = readCh();
      }
      printf("Stecker: ");
      i = 0;
      while((c = getchar()) != '\n'){
        p->plug[i] = c;
       i++;
      }
      p->plug[i] = '\0';
  }
  printf("Wheels %d %d %d Start %c %c %c Rings %c %c %c Stecker \"%s\"\n",
         p->order[0], p->order[1], p->order[2],
         p->pos[0], p->pos[1], p->pos[2],
         p->rings[0], p->rings[1], p->rings[2], p->plug);
}


/*******************************MAIN*********************************************/
void main(int argc, char *argv[]){
  Params p;
  if(argc > 2)  /*bombe case*/
  {
    permuteAll(argv[1], argv[2]);
  }
  else
  {
    initParams(&p);
    cypher(p);
  }
}
```

Ce programme lancé tel quel en prenant l'initialisation par défaut codera « BONJOUR » par « FDWIZRH ».

2°) Cryptanalyse d'Enigma

Il est clair qu'actuellement la cryptanalyse d'Enigma par force brute n'est plus un problème avec les ordinateurs.

Le protocole de départ était le suivant. Une autorité S diffusait de manière écrite pour chaque jour des clés K_J (clé du jour). Cette clé servait à coder la clé effective sachant que cette clé était codée deux fois en début de message : on l'appellera clé de session K_S.
On peut résumer cela avec nos notations des protocoles :

$$A \rightarrow B : || E_{K_J}(K_S, K_S, E_{K_S}(M))||$$

Bernard retrouve la clé de session en faisant $D_{KJ}(K_S, K_S, E_{K_S}(M))$ puis le message avec $D_{K_S}(E_{K_S}(M))$.

Le doublement de la clé de session est une faiblesse du protocole que le Polonais Marian Rejewski va commencer à mettre à profit.
L'idée consiste à essayer de démêler l'influence des brouilleurs de celle du réflecteur.

IV) Bibliographie

Simon Singh « Histoire des codes secrets » J.C. Lattès (1999)

V) Exercices

A la fin du livre de simon Singh on trouve un certain nombre de challenges.

Exercice 1

Exercice 2 (Vigenere)

```
KQOWEFVJPUJUUNUKGLMEKJINMWUXFQMKJB
GWRLFNFGHUDWUUMBSVLPSNCMUEKQCTESWR
EEKOYSSIWCTUAXYOTAPXPLWPNTCGOJBGFQ
HTDWXIZAYGFFNSXCSEYNCTSSPNTUJNYTGG
WZGRWUUNEJUUQEAPYMEKQHUIDUXFPGUYTS
MTFFSHNUOCZGMRUWEYTRGKMEEDCTVRECFB
DJQCUSWVBPNLGOYLSKMTEFVJJTWWMFMWPN
MEMTMHRSPXFSSKFFSTNUOCZGMDOEOYEEKC
PJRGPMURSKHFRSEIUEVGOYCWXIZAYGOSAA
NYDOEOYJLWUNHAMEBFELXYVLWNOJNSIOFR
WUCCESWKVIDGMUCGOCRUWGNMAAFFVNSIUD
EKQHCEUCPFCMPVSUDGAVEMNYMAMVLFMAOY
FNTQCUAFVFJNXKLNEIWCWODCCULWRIFTWG
MUSWOVMATNYBUHTCOCWFYTNMGYTQMKBBNL
GFBTWOJFTWGNTEJKNEEDCLDHWTVBUVGFBI
JGYYIDGMVRDGMPLSWGJLAGOEEKJOFEKNYN
OLRIVRWVUHEIWUURWGMUTJCDBNKGMBIDGM
EEYGUOTDGGQEUJYOTVGGBRUJYS
```

Réponse :

La clé était : SCUBA et le texte un célèbre poème de Beaudelaire.

```
SOUVENTPOURSAMUSERLESHOMMESDEQUIPA
GEPRENNENTDESALBATROSVASTESOISEAUX
DESMERSQUISUIVENTINDOLENTSCOMPAGNO
NSDEVOYAGELENAVIREGLISSANTSURLESGO
UFFRESAMERSAPEINELESONTILSDEPOSESS
URLESPLANCHESQUECESROISDELAZURMALA
DROITSETHONTEUXLAISSENTPITEUSEMENT
LEURSGRANDESAILESBLANCHESCOMMEDESA
VIRONSTRAINERACOTEDEUXCEVOYAGEURAI
LECOMMEILESTGAUCHEETVEULELUINAGUER
ESIBEAUQUILESTCOMIQUEETLAIDLUNAGAC
ESONBECAVECUNBRULEGUEULELAUTREMIME
ENBOITANTLINFIRMEQUIVOLAITLEPOETEE
STSEMBLABLEAUPRINCEDESNUEESQUIHANT
ELATEMPETEETSERITDELARCHERBAUDELAI
REEXILESURLESOLAUMILIEUDESHUEESLEM
OTPOURETAGEQUATREESTTRAJANSESMILES
DEGEANTLEMPECHENTDEMARCHER
```

On donne aussi deux sous-programme en C qui permettent de coder et décoder suivant Vigenère :

```
char bufi[80][NCOL],buff[80][NCOL]; /* variables globales i:initial f:final */

void ChiffreVigenere(int line){ /* nb de ligne utilisée dans bufi */
  char clef[20];
  char alphabet[27]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  int lgcle,i,lgline,j,k;
  printf("Quelle est la clé ?\n");
  scanf("%s",clef);
  printf("Ne fonctionne que si le texte source est en majuscule !!!\n");
```

```
  lgcle=strlen(clef);
  i=0;
  for (j=0;j<line;j++) {
    lgline=strlen(bufi[j]);
    for (k=0;k<lgline;k++) {
      if ((bufi[j][k]>='A')&&(bufi[j][k]<='Z')) {
        buff[j][k]=alphabet[(bufi[j][k]-'A'+clef[i]-'A')% 26];
        i++;
        i=i%lgcle;
      } else buff[j][k]=bufi[j][k];
    }
    buff[j][k]=0;// fin de ligne
  }
  for (i=0;i<line;i++)
    printf(buff[i]);
}

void DecodeVigenere(int line){
  char clef[20];
  char alphabet[27]="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  int lgcle,i,lgline,j,k;
  printf("Quelle est la clé ?\n");
  scanf("%s",clef);
  printf("Ne fonctionne que si le texte source est en majuscule !!!\n");
  lgcle=strlen(clef);
  i=0;
  for (j=0;j<line;j++) {
    lgline=strlen(bufi[j]);
    for (k=0;k<lgline;k++) {
      if ((bufi[j][k]>='A')&&(bufi[j][k]<='Z')) {
        buff[j][k]=alphabet[(bufi[j][k]-'A'-(clef[i]-'A')+26)% 26];
        i++;
        i=i%lgcle;
      } else buff[j][k]=bufi[j][k];
    }
    buff[j][k]=0;// fin de ligne
  }
  for (i=0;i<line;i++)
    printf(buff[i]);
}
```