****

**Chiffrement de Hill**

**S3 Groupe A**

**FRELOT Victor**

**ZHAO Mengzi**

# Introduction et Principe de Chiffrement de Hill

## Introduction :

Le chiffrement de Hill est un modèle simple d'extension du [chiffrement affine](http://fr.wikipedia.org/wiki/Chiffre_affine) à un [bloc](http://fr.wikipedia.org/wiki/Chiffrement_par_bloc). Ce système étudié par [Lester S. Hill](http://fr.wikipedia.org/wiki/Lester_S._Hill)[1](http://fr.wikipedia.org/wiki/Chiffre_de_Hill#cite_note-1), utilise les propriétés de l'[arithmétique modulaire](http://fr.wikipedia.org/wiki/Arithm%C3%A9tique_modulaire) et des [matrices](http://fr.wikipedia.org/wiki/Matrice).

Il s'agit de chiffrer le message en substituant les lettres du message, non plus lettre à lettre, mais par groupe de lettres. Il permet ainsi de rendre plus difficile le cassage du code par observation des fréquences.

Lester S. Hill a aussi conçu une machine capable de réaliser mécaniquement un tel codage[2](http://fr.wikipedia.org/wiki/Chiffre_de_Hill#cite_note-2).

## Principe :

Chaque caractère est d'abord codé par un nombre compris entre 0 et n -1 (son rang dans l'alphabet diminué de 1 ou son code ASCII diminué de 32). Les caractères sont alors regroupés par blocs de p caractères formant un certain nombre de vecteurs X = (x_1, x_2, \cdots, x_p). Les nombres x_i étant compris entre 0 et n - 1, on peut les considérer comme des éléments de \Z/n\Z et X est alors un élément de (\Z/n\Z)^p.

On construit alors une matrice p × p d'éléments de \Z/n\Z : A. Le bloc X est alors chiffré par le bloc Y = AX, le produit s'effectuant modulo n.

Pour déchiffrer le message, il s'agit d'inverser la matrice A. Cela peut se faire si le déterminant de cette matrice est premier avec n. En effet, si Det(A) est premier avec n, il existe un élément k de \Z/n\Z tel que k\times Det(A) \equiv 1 \pmod n Ceci est une conséquence directe du [théorème de Bachet-Bézout](http://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_de_Bachet-B%C3%A9zout)

Le produit de A et de sa [comatrice](http://fr.wikipedia.org/wiki/Comatrice) transposée tCom(A) donne Det(A) I_p. Si l'on appelle B la matrice ktcom(A). On aura AB=BA=I_p. Soit encore

Y=AX \Leftrightarrow X=BY.

# Exercices du cours :

## Principe de chiffrement de Hill

### Codage :

**Exemples du cours:**

1. Montrer que la matrice B est inversible.

**Principe à utiliser :** Si la matrice B est inversible mod 26, il faut que le pgcd entre le déterminant de matrice B et le 26 soit 1.

restart:with(linalg):

***Explication*** *:*

*Le package linalg contient plus d'une centaine d'outils en algèbre linéaire*

matrixB:=matrix(2,2,[2,3,7,5]);  
***Explication*** *:*

*matrix : entrer une matrice de taille 2\*2*

detBmod26:=map(x->x mod 26,det(matrixB));

***Explication*** *:*

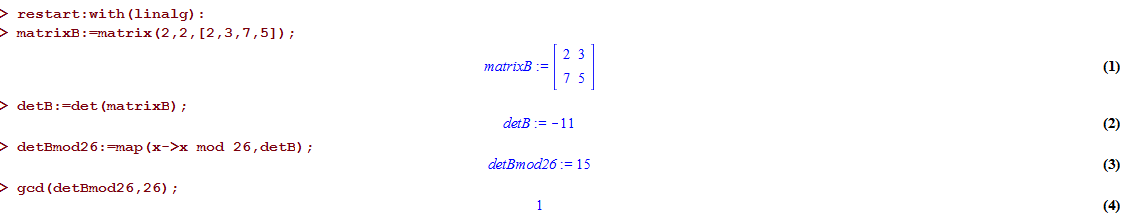
1. *map : appliquer une procédure à chaqueopérande d'une expression*
2. *det : calculer le déterminant de matrice*
3. *mod : modulo*

pgcd := gcd(detBmod26,26);

***Explication*** *:*

*gcd : calculer le pgcd entre le déterminant de matrice B et le 26 soit 1*

Nous avons obtenu : pgcd := 1, donc la matrice B est inversible.



1. La matrice C et D, montrer laquelles de ces deux matrices a un inverse dans Z/26Z

**Principe à utiliser :** Si 1 matrice est inversible mod 26, il faut que le pgcd entre le déterminant de cette matrice et le 26 soit 1.

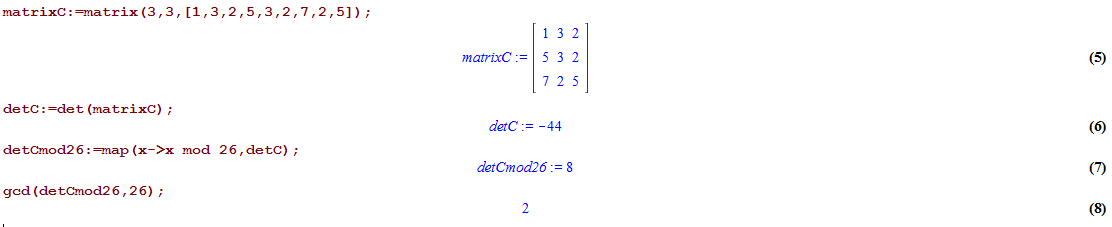
matrixC:=matrix(3,3,[1,3,2,5,3,2,7,2,5]);

detC:=det(matrixC);

detCmod26:=map(x->x mod 26,detC);

gcd(detCmod26,26);

**Nous avons obtenu : pgcd := 2, donc la matrice C n’est pas inversible.**



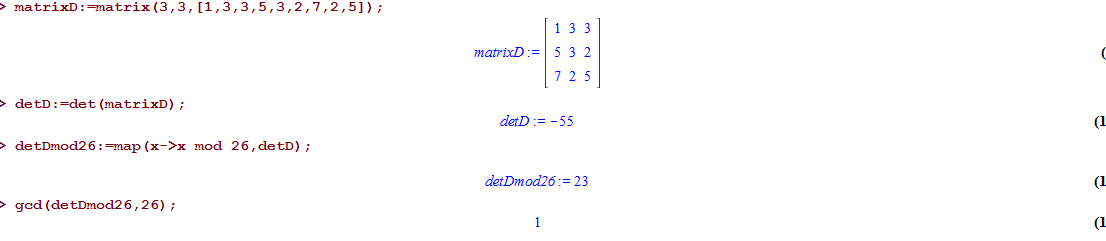
matrixD:=matrix(3,3,[1,3,3,5,3,2,7,2,5]);

detD:=det(matrixD);

detDmod26:=map(x->x mod 26,detD);

gcd(detDmod26,26);

**Nous avons obtenu : pgcd := 1, donc la matrice D est inversible.**



1. Coder le message “HILL” par la matrice B.

letters := array ( 0..25,[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]);

***Explication*** *:*

*créer une liste contenante tous les alphabets, l’index de cette liste commence par 0, donc la dernière indice est 25.*

letters [5];

***Explication*** *:*

*tester la liste contenante les alphabets est bien crée, donc lettre[5] := F*

itable := table();

***Explication*** *:*

*créer un table*

for i from 0to 25 do

itable[ letters[i] ] := i:

od:

***Explication*** *:*

*mettre tous les lettres que nous venons de créer, c’est-à-dire : transformer tous les lettres comme A, B, C en chiffre (leurs indices).*

itable[V];

***Explication*** *:*

*tester le tableau que nous avons cré est bien fonctionné, donc nous avons obtennu 21*

with(linalg):

***Explication*** *:*

*Le package linalg contient plus d'une centaine d'outils en algèbre linéaire*

B:= matrix(2,2,[2,3,7,5]);

***Explication*** *:*

*matrix : entrer une matrice de taille 2\*2*

d := det(B) mod 26;

***Explication*** *:*

*det : calculer le déterminant de matrice*

gcd(d,26);

***Explication*** *:*

*gcd : calculer le pgcd entre le déterminant de matrice B et le 26 soit 1*

t := vector([H, I, L, L]);

***Explication*** *:*

*vector : construire un vecteur*

vectdim(t);

***Explication*** *:*

*vectdim : déterminer la dimension d'un vecteur*

for i from 1 to vectdim(t) do

t[i] := itable[ t[i]]:

od:

evalm(t);

***Explication*** *:*

*transformer ce message “H I L L” en dimension dans le vecteur (leurs indices)*

blocksize := rowdim(B);

***Explication*** *:*

*rowdim : déterminer la dimension de ligne d'une matrice, c’est à dire le nombre de colonne de cette matrice*

numblocks := vectdim(t)/blocksize;

***Explication*** *:*

*calculer le nombre de ligne de matrice créant par les lettre de message*

pmatrix := matrix(numblocks, blocksize,t);

***Explication*** *:*

*créer la matrice de message*

*matrix : construire une matrice*

cmatrix := map(m-> m mod 26, evalm(pmatrix &\*B));

***Explication*** *:*

*map : appliquer une procédure à chaque opérande d'une expression, ici, nous faisons modulo 26 pour le produit de pmatrix et matriceB.*

ct := convert (cmatrix, vector);

***Explication*** *:*

*covertir cmatrix en type vecteur*

*convert : convertir une expression à une forme différente*

for i from 1 to vectdim(ct)do

ct[i] := letters[ct[i]];

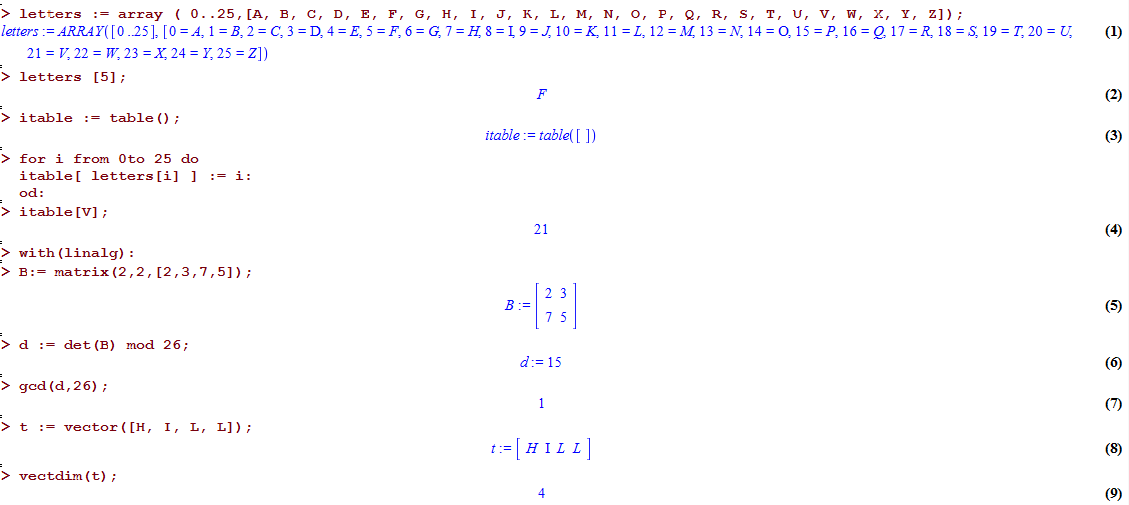
od;

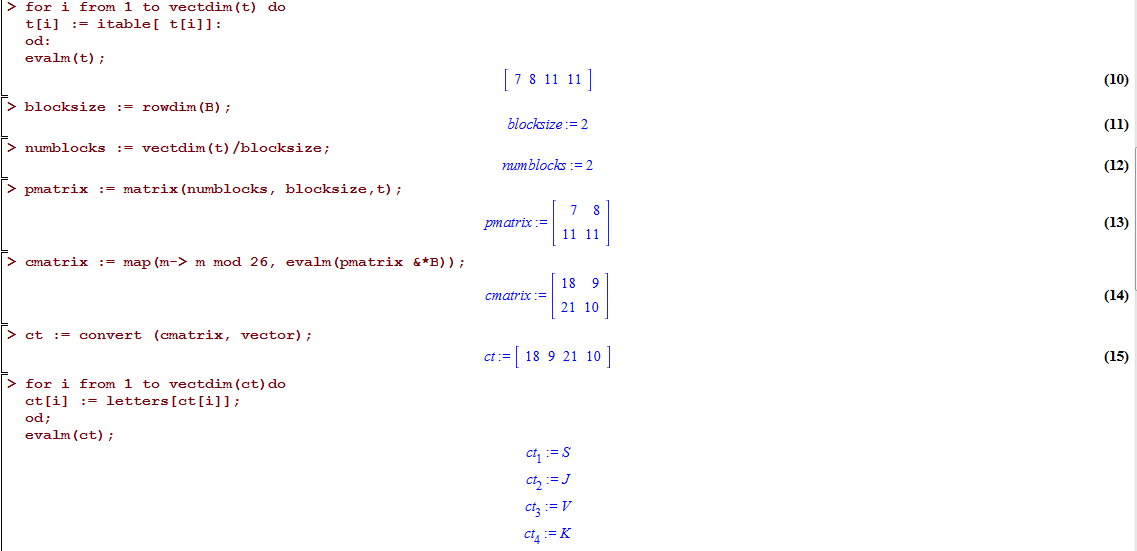
evalm(ct);

***Explication*** *:*

*parcourir le vecteur*

*transformer les chiffres en texte*





1. Combien y a-t-il de clés possibles pour le code Hill 2\*2.

Pour coder une message, il faut le pgcd entre le déterminant qui est fait modulo 26 et le 26 soit 1, donc le déterminant de clés ne doivent pas être divisé par 2 ou 13 ou 26.

### Décodage :

1. Une information codée par une clé A (inversible) est décodé tout simplement en écrivant : X = A^(-1) Y (mod 26).

Décoder le cryptogramme suivant “UWGMWZRREIUB” sachant que la clé de chiffrement est A:= matrix(2,2,[9,4,5,7]);

letters := array ( 0..25,[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]);

***Explication :***

*créer une liste contenante tous les alphabets, l’index de cette liste commence par 0, donc la dernière indice est 25.*

letters [24];

***Explication*** *:*

*tester la liste contenante les alphabets est bien crée, donc lettre[24] := Y*

itable := table();

***Explication*** *:*

*créer un table*

for i from 0 to 25 do

itable[ letters[i] ] := i:

od:

***Explication*** *:*

*mettre tous les lettres que nous venons de créer, c’est-à-dire : transformer tous les lettres comme A, B, C en chiffre (leurs indices).*

with(linalg):

***Explication*** *:*

*Le package linalg contient plus d'une centaine d'outils en algèbre linéaire*

itable[E];

***Explication*** *:*

*tester le tableau que nous avons cré est bien fonctionné, donc nous avons obtennu 4*

A:= matrix(2,2,[9,4,5,7]);

***Explication*** *:*

*matrix : entrer une matrice de taille 2\*2*

d := det(A) mod 26;

***Explication*** *:*

*det : calculer le déterminant de matrice*

gcd(d,26);

***Explication*** *:*

*gcd : calculer le pgcd entre le déterminant de matrice B et le 26 soit 1*

t := vector([U, M, G, M, W, Z, R, R, E, I, U, B]);

***Explication*** *:*

*vector : construire un vecteur*

vectdim(t);

***Explication*** *:*

*vectdim : déterminer la dimension d'un vecteur*

for i from 1 to vectdim(t) do

t[i] := itable[ t[i]]:

od:

evalm(t);

***Explication*** *:*

*transformer ce message “U, M, G, M, W, Z, R, R, E, I, U, B” en dimension dans le vecteur (leurs indices)*

blocksize := rowdim(A);

***Explication*** *:*

*rowdim : déterminer la dimension de ligne d'une matrice, c’est à dire le nombre de colonne de cette matrice*

numblocks := vectdim(t)/blocksize;

***Explication*** *:*

*calculer le nombre de ligne de matrice créant par les lettre de message*

pmatrix := matrix(numblocks, blocksize,t);

***Explication*** *:*

*créer la matrice de message*

*matrix : construire une matrice*

cmatrix := map(m-> m mod 26, evalm(pmatrix &\*A));

***Explication*** *:*

*map : appliquer une procédure à chaque opérande d'une expression, ici, nous faisons modulo 26 pour le produit de pmatrix et matriceA.*

ct := convert (cmatrix, vector);

***Explication*** *:*

*covertir cmatrix en type vecteur*

*convert : convertir une expression à une forme différente*

for i from 1 to vectdim(ct)do

ct[i] := letters[ct[i]];

od;

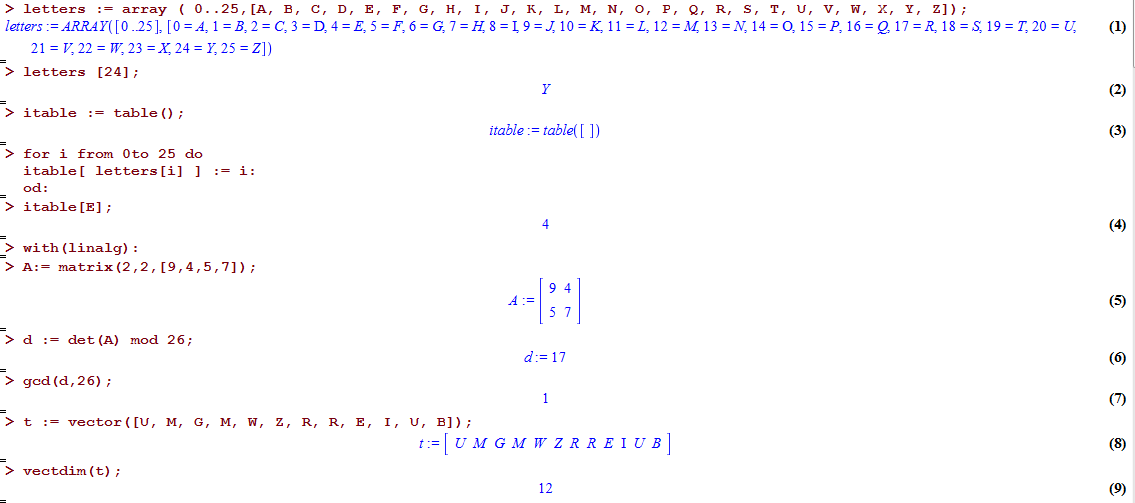
evalm(ct);

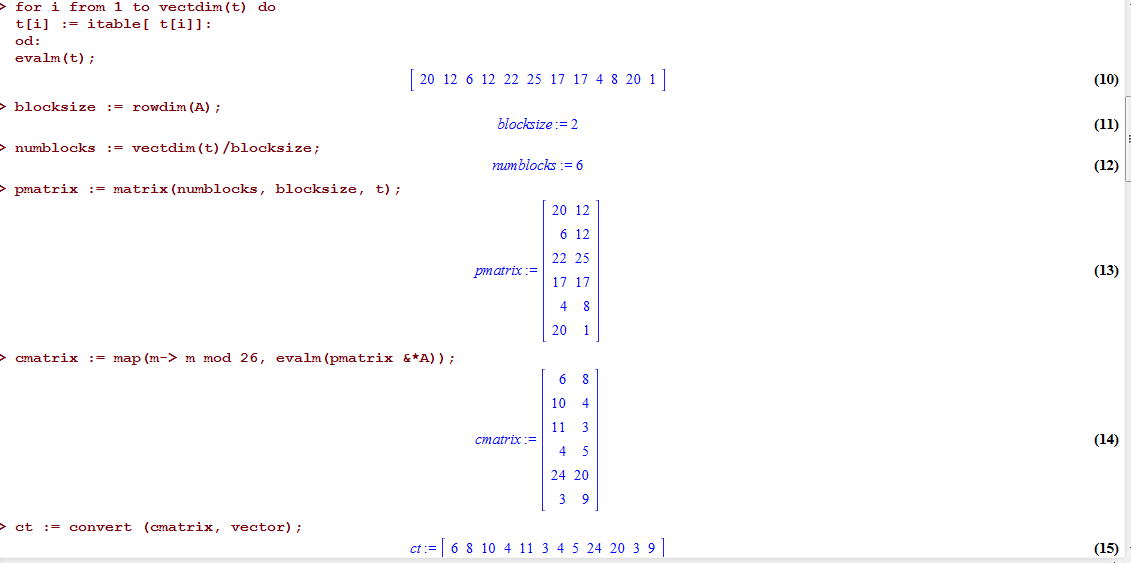
***Explication*** *:*

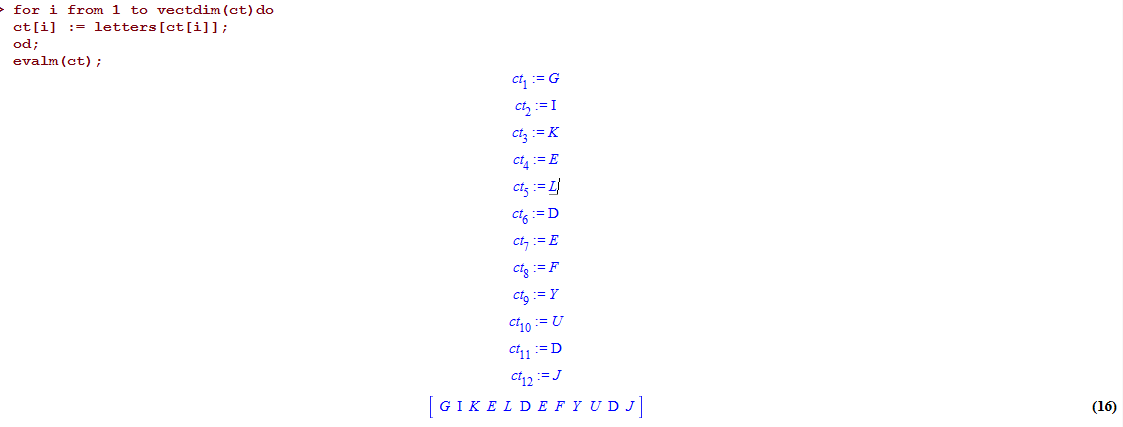
*parcourir le vecteur*

*transformer les chiffres en texte*

**Donc, nous avons obtenu le résultat: G,I,K,E,L,D,E,F,Y,U,D,J**







# Donc, pour le décoder, nous utilisons le même principe :

for i from 1 to vectdim(ct) do

ct[i] :=itable[ ct[i] ];

od;

evalm(ct);

cmatrix:= matrix(numblocks, blocksize, ct);

pmatrix := map(m -> m mod 26, evalm(cmatrix &\* A^(-1)));

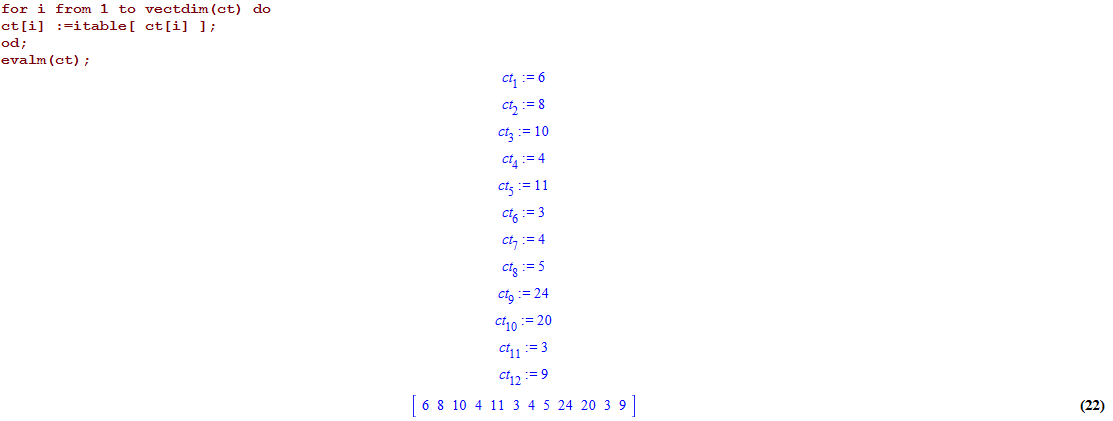
ptext := convert(pmatrix, vector);

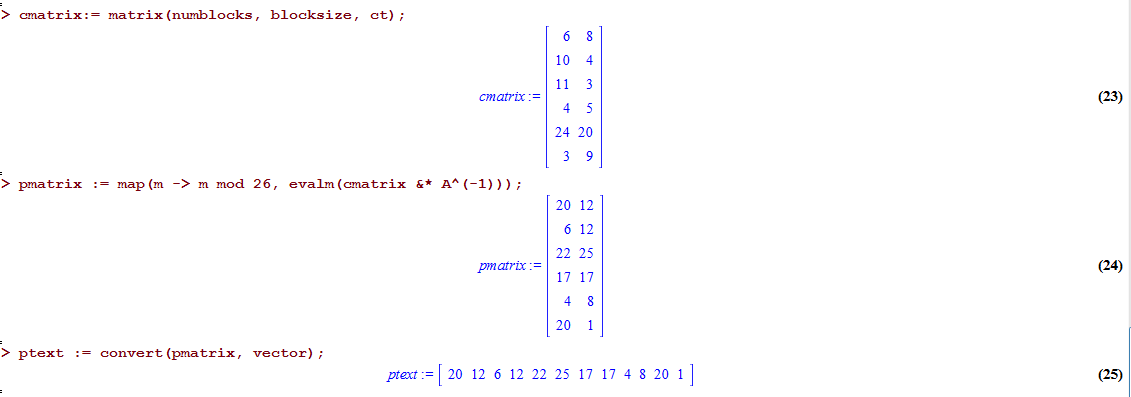
for i from 1 to vectdim(ptext) do

ptext[i] := letters[ ptext[i] ];

od;

evalm(ptext);





# Decoder3.png

# Cryptanalyse du code de Hill :

## Procedure CryptHill :

restart; with(linalg):

***Explication*** *:*

*Le package linalg contient plus d'une centaine d'outils en algèbre linéaire*

CrypteHill := proc(ma::matrix,la::vector)

***Explication*** *:*

*créer une procedure, ma et la sont des paramètres en entrée, ma est la matrice que nous devons entrer pour décoder, l est le vecteur de texte que nous devons entrer.*

local letters, itable, matrice, pgcd, texte, blocksize, numblocks, pmatrix, cmatrix, ct,i,ptext;

***Explication*** *:*

*créer des variables*

letters := array ( 0..25,[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]);

***Explication :***

*créer une liste contenante tous les alphabets, l’index de cette liste commence par 0, donc la dernière indice est 25.*

itable := table();

***Explication*** *:*

*créer un table*

for i from 0 to 25 do

itable[ letters[i] ] := i:

od:

***Explication :***

*créer une liste contenante tous les alphabets, l’index de cette liste commence par 0, donc la dernière indice est 25.*

matrice:=ma;

***Explication :***

*ma est une variable en entrée donc nous ne pouvons pas changer dans la procédure, donc on donne la valeur ma (matrice : la clé) à une variable locale et nous pouvons convertir son type ou faire les autres choses*

texte := la;

***Explication :***

*la est une variable en entrée donc nous ne pouvons pas changer dans la procédure, donc on donne la valeur la (texte à décoder) à une variable locale texte et nous pouvons convertir son type ou faire les autres choses*

pgcd := gcd(det(matrice) mod 26, 26);

***Explication :***

*calculer le pgcd entre le déterminant de cette matrice (qui est déja modulo 26) et le 26*

if pgcd = 1 then

***Explication :***

*vérifier le pgcd est égal à 1 ou pas, si le pgcd = 1, le procédure peut continuer, sinon, le procedure arrête.*

for i from 1 to vectdim(texte) do

texte[i] := itable[ texte[i]]:

od:

evalm(texte);

***Explication*** *:*

*transformer le message que nous avons entrer en dimension dans le vecteur (leurs indices)*

blocksize := rowdim(ma);

***Explication*** *:*

*rowdim : déterminer la dimension de ligne d'une matrice, c’est à dire le nombre de colonne de cette matrice*

numblocks := vectdim(texte)/blocksize;

***Explication*** *:*

*calculer le nombre de ligne de matrice créant par les lettre de message*

pmatrix := matrix(numblocks, blocksize, texte);

***Explication*** *:*

*créer la matrice de message*

*matrix : construire une matrice*

cmatrix := map(m-> m mod 26, evalm(pmatrix &\*ma));

***Explication*** *:*

*map : appliquer une procédure à chaque opérande d'une expression, ici, nous faisons modulo 26 pour le produit de pmatrix et matrice.*

ct := convert (cmatrix, vector);

***Explication*** *:*

*covertir cmatrix en type vecteur*

*convert : convertir une expression à une forme différente*

for i from 1 to vectdim(ct)do

ct[i] := letters[ct[i]];

od;

evalm(ct);

***Explication*** *:*

*parcourir le vecteur*

*transformer les chiffres en texte*

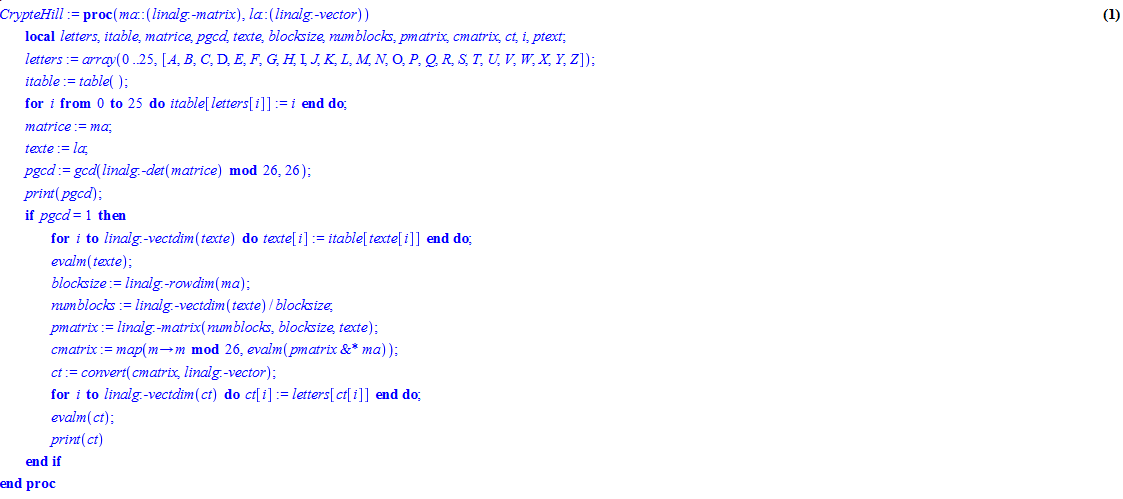
print(ct);

***Explication*** *:*

*afficher le texte que nous avons codé*

fi;

end proc;

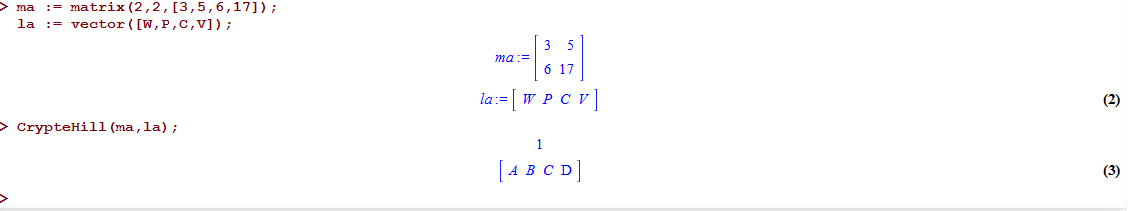


**Donc, pour le tester :**

ma := matrix(2,2,[3,5,6,17]);

la := vector([W,P,C,V]);

CrypteHill(ma,la);



## Procedure DryptHill :

restart; with(linalg):

***Explication*** *:*

*Le package linalg contient plus d'une centaine d'outils en algèbre linéaire*

DCrypteHill := proc(m::matrix,l::vector)

***Explication*** *:*

*créer une procedure, m et l sont des paramètres en entrée, m est la matrice que nous devons entrer, l est le vecteur de texte que nous devons entrer.*

local letters, itable, matrice, pgcd, texte, blocksize, numblocks, pmatrix, cmatrix, ct,i,ptext;

***Explication*** *:*

*créer des variables*

letters := array ( 0..25,[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]);

***Explication :***

*créer une liste contenante tous les alphabets, l’index de cette liste commence par 0, donc la dernière indice est 25.*

itable := table();

***Explication*** *:*

*créer un table*

for i from 0 to 25 do

itable[ letters[i] ] := i:

od:

***Explication :***

*créer une liste contenante tous les alphabets, l’index de cette liste commence par 0, donc la dernière indice est 25.*

matrice:=m;

***Explication :***

*m est une variable en entrée donc nous ne pouvons pas changer dans la procédure, donc on donne la valeur m (matrice : la clé) à une variable locale et nous pouvons convertir son type ou faire les autres choses*

pgcd:=gcd(det(matrice) mod 26, 26);

***Explication :***

*calculer le pgcd entre le déterminant de cette matrice (qui est déja modulo 26) et le 26*

if pgcd = 1 then

***Explication :***

*vérifier le pgcd est égal à 1 ou pas, si le pgcd = 1, le procédure peut continuer, sinon, le procedure arrête.*

texte := l;

***Explication :***

*l est une variable en entrée donc nous ne pouvons pas changer dans la procédure, donc on donne la valeur l (texte à coder) à une variable locale texte et nous pouvons convertir son type ou faire les autres choses*

for i from 1 to vectdim(texte) do

texte[i] := itable[ texte[i]]:

od:

evalm(texte);

***Explication*** *:*

*transformer le message que nous avons entrer en dimension dans le vecteur (leurs indices)*

blocksize := rowdim(matrice);

***Explication*** *:*

*rowdim : déterminer la dimension de ligne d'une matrice, c’est à dire le nombre de colonne de cette matrice*

numblocks := vectdim(texte)/blocksize;

***Explication*** *:*

*calculer le nombre de ligne de matrice créant par les lettre de message*

cmatrix := matrix(numblocks, blocksize, texte);

***Explication*** *:*

*créer la matrice de message*

*matrix : construire une matrice*

pmatrix := map(m -> m mod 26, evalm(cmatrix &\* matrice^(-1)));

***Explication*** *:*

*map : appliquer une procédure à chaque opérande d'une expression, ici, nous faisons modulo 26 pour le produit de pmatrix et matrice.*

ptext := convert(pmatrix, vector);

***Explication*** *:*

*covertir cmatrix en type vecteur*

*convert : convertir une expression à une forme différente*

for i from 1 to vectdim(ptext) do

ptext[i] := letters[ ptext[i] ];

od;

evalm(ptext);

***Explication*** *:*

*parcourir le vecteur*

*transformer les chiffres en texte*

print(ptext);

***Explication*** *:*

*afficher le texte que nous avons codé*

fi;

end proc;



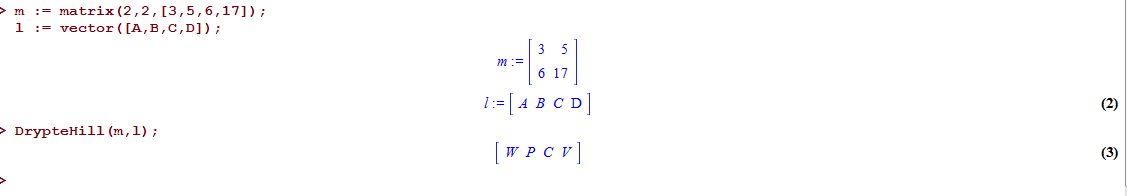
Pour le tester :

m := matrix(2,2,[9,4,5,7]);

l := vector([G,I,K,E,L,D,E,F,Y,U,D,J]);

DCrypteHill(m,l);

**Donc, nous avons obtenu le résultat : UMGMWZRREIUB**



## Procedure AttackHill (n=2) :

restart; with(linalg):

***Explication*** *:*

*Le package linalg contient plus d'une centaine d'outils en algèbre linéaire*

AttackHill := proc(a,b,c,d,e,f,g,h)

***Explication*** *:*

*créer une procedure, m et l sont des paramètres en entrée, m est la matrice que nous devons entrer, l est le vecteur de texte que nous devons entrer.*

local matrice,a1, b1, a2, b2, m1, m2, m3, m4, letters, itable, i,s, aa, bb, cc, dd, ee, ff, gg, hh, pgcd;

***Explication*** *:*

*créer des variables locales*

a1 := vector([a,b]);

b1 := vector([c,d]);

a2 := vector([e,f]);

b2 := vector([g,h]);

***Explication*** *:*

*mettre des lettres que nous avons entré en vecteur*

letters := array ( 0..25,[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]); ***Explication :***

*créer une liste contenante tous les alphabets, l’index de cette liste commence par 0, donc la dernière indice est 25.*  
itable := table();

***Explication :***

*créer un table*

for i from 0 to 25 do

itable[ letters[i] ] := i:

od:

***Explication :***

*créer une liste contenante tous les alphabets, l’index de cette liste commence par 0, donc la dernière indice est 25.*

aa := itable[a]; print(aa);

bb := itable[b]; print(bb);

cc := itable[c]; print(cc);

dd := itable[d]; print(dd);

ee := itable[e]; print(ee);

ff := itable[f]; print(ff);

gg := itable[g]; print(gg);

hh := itable[h]; print(hh);

***Explication :***

*transformer les lettre en chiffre(leurs indices)*

s :=solve({(m1\*aa+m3\*bb) mod 26 = cc,(m2\*aa+m4\*bb) mod 26 = dd, (m1\*ee+m3\*ff) mod 26 = gg,(m2\*ee+m4\*ff) mod 26 = hh},{m1,m2,m3,m4});  
***Explication :***

*trouver la solution des quatre équation*

*solve : trouver la solution des équations*

assign(s);

***Explication :***

*assign : mettre à jour les valeur aux variables*

print(s);

***Explication :***

*print : afficher*

while m1 >= 26 do

m1 := m1-26;

od;

while m1 < 0 do

m1 := m1+26;

od;

while m2 >= 26 do

m2 := m2-26;

od;

while m2 < 0 do

m2 := m2+26;

od;

while m3 >= 26 do

m3 := m3-26;

od;

while m3 < 0 do

m3 := m3+26;

od;

while m4 >= 26 do

m4 := m4-26;

od;

while m4 < 0 do

m4 := m4+26;

od;

***Explication :***

*faire modulo 26 pour m1,m2,m3,m4*

matrice:= matrix(2,2,[m1,m2,m3,m4]);

***Explication :***

*créer la matrice en utilisant m1,m2,m3,m4*

pgcd := gcd(26,det(matrice) mod 26);

***Explication :***

*calculer le pgcd entre le déterminant de matrice (déja modulo 26) et le 26*

print(pgcd);

***Explication :***

*afficher le pgcd*

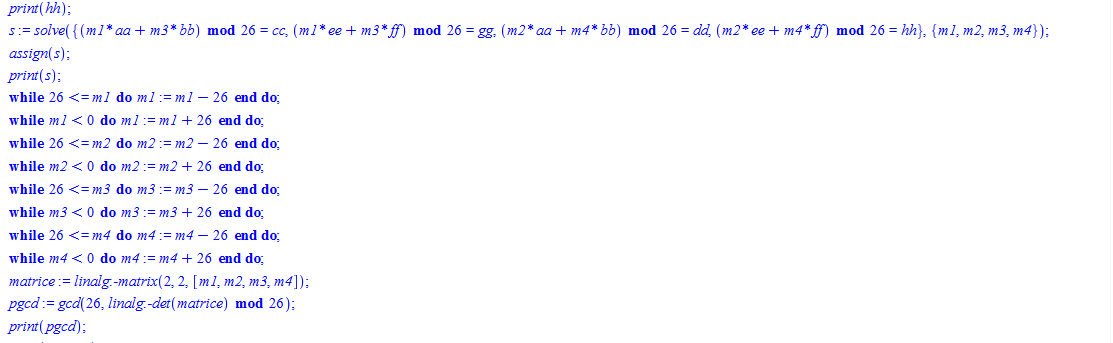
print(matrice);

***Explication :***

*afficher la matrice*

end proc;





Pour le tester :   
AttackHill(A,B,G,R,C,D,Y,J);  
Nous avons obtenu :

