Introduction à la cryptologie.

TP du 22 mars 2013, de 13h30 à 16h45, durée 3h.
Cours de Alexei Pantchichkine

Un compte-rendu par binôme est à adresser par courriel à Alexei Pantchichkine, panchish@ujf-grenoble.fr avant le 2 avril 2013. Il devra comporter :

– un rapport (au format PDF de préférence) avec vos réponses aux questions ainsi que les commentaires que vous jugerez utile sur votre code ;

– les sources des programmes réalisés ;

– d'autres annexes, le cas échéant.

Un rendu de TP sans rapport sera noté 0/20. Des commentaires éparpillés dans le code ne constituent pas un rapport valide. Groupez vos fichiers et n'oubliez pas d'indiquer vos deux noms.

Les fichiers nécessaires indiqués dans le TP se trouvent sur
`http://www-fourier.ujf-grenoble.fr/ panchish/tp/pariCRY/`

Le but de ce TP est d'étudier les algorithmes sur les polynômes, corps finis et primalité (Polynômes sur les corps finis, division euclidienne, racines, irréductibilité, algorithme d'Euclide étendu, le théorème chinois des restes pour les polynômes, Construction et structure des corps finis, Le groupe multiplicatif des inversibles d'un corps finis, Primalité et fabrication de nombres premiers, Nombres pseudopremiers de Fermat ; Test de Miller-Rabin), Schémas de chiffrement asymétriques et cryptographie à clef publique (Le protocole RSA, Chiffrement de Rabin, Échange de clés selon Diffie-Hellman, Le protocole ElGamal), ainsi que Logarithmes discrets (Algorithme pas de bébé/ pas de géant de Shanks, Algorithme ρ de Pollard, Calcul d'indice).

1. Polynômes, corps finis et primalité. Arithmétique modulaire des polynômes

> Polynômes sur les corps finis, division euclidienne, racines, irréductibilité, algorithme d'Euclide étendu, le théorème chinois des restes pour les polynômes, Construction et structure des corps nis, Le groupe multiplicatif des inversibles d'un corps finis

1.1. **Exemple de division euclidienne.** Traitons d'abord un exemple dans $K = \mathbb{Q}$. On veut effectuer la division euclidienne de $f = X^4 - 7X + 11$ par $g = 2X^2 - 6$. On initialise r à f et q à 0. Comme $r \neq 0$ et $\deg(r) \geq \deg(g)$, on calcule $h = \frac{1}{2}X^2$, on remplace r par $r - gh = 3X^2 - 7X + 11$ et on remplace q par $q + h = \frac{1}{2}X^2$. Ensuite, r est toujours non nul et on a encore $\deg(r) \geq \deg(g)$ donc on calcule $h = \frac{3}{2}$, on remplace r par $r - gh = -7X + 20$ et on remplace q par $q + h = \frac{1}{2}X^2 + \frac{3}{2}$. Enfin, $r \neq 0$ et $\deg(r) < \deg(g)$ donc on arrête le calcul et on a

$$X^4 - 7X + 11 = (\frac{1}{2}X^2 + \frac{3}{2})(2X^2 - 6) + (-7X + 20)$$

```
gp > f=X^4-7*X+11; g=2*X^2-6;[f,g]
%1 = [X^4 - 7*X + 11, 2*X^2 - 6]
gp > f%g
%2 = -7*X + 20
gp > f/g
%3 = (X^4 - 7*X + 11)/(2*X^2 - 6)
gp > (f-f%g)/g
%4 = 1/2*X^2 + 3/2
```

Traitons maintenant un exemple dans un des corps fini $\mathbb{Z}/5\mathbb{Z}$. On rappelle que c'est un corps car 5 est un nombre premier. On travaille avec le système de représentants formés des plus petits restes positifs. Pour un entier $x \in \{0, \ldots, 4\}$ on notera x plutôt que $\widetilde{x}$ pour alléger.

Effectuons la division euclidienne de $f = X^3 + X^2 + X + 2$ par $g = 2X^2 + 2$ sur PARI/GP

```
gp > f=(X^3+X^2+X+2)*Mod(1,5); g=(2*X^2+2)*Mod(1,5);f
% = Mod(1, 5)*X^3 + Mod(1, 5)*X^2 + Mod(1, 5)*X + Mod(2, 5)
gp > f%g
% = Mod(1, 5)
gp > (f-f%g)/g
% = Mod(3, 5)*X + Mod(3, 5)
gp > lift(%)
% = 3*X + 3
```

1.2. **Effectuer la division euclidienne de f par g.** Vérifier la validité de la solution trouvée sur PARI/GP

```
gp > f = X^3- X^2 + 6;g = X^2 + 4; f%g
% = -4*X + 10
```

- $f = 4X^3 + 3X^2 + 2X + 1$ et $g = 4X^2 + 1$ dans $(\mathbb{Z}/5\mathbb{Z})[X]$
- $f = X^6$ et $g = 3X^3 + 2X^2 + X$ dans $(\mathbb{Z}/7\mathbb{Z})[X]$
- $f = X^4 + 2X^3 + 3X^2 + 4X + 5$ et $g = X^2 + 1$ dans $(\mathbb{Z}/11\mathbb{Z})[X]$

1.3. **Trouver toutes les racines de f et factoriser f le plus possible.**

- $f = X^3 + X^2 + 4X + 1$ dans $(\mathbb{Z}/7\mathbb{Z})[X]$
- $f = X^4 - 1$ dans $(\mathbb{Z}/13\mathbb{Z})[X]$
- $f = X^3 + 2X + 1$ dans $\mathbb{Z}/3\mathbb{Z}[X]$

1.4. **Calculer** $\mathrm{pgcd}(f, g)$ **et un couple de Bezout pour f et g avec l'algorithme d'Euclide étendu.** *Indication :*

```
gp > f = X^2 - 4*X - 7; g = X + 2;f
% = X^2 - 4*X - 7
gp > bezout(f,g)
% = [1/5, -1/5*X + 6/5, 1]
```

- $f = X^5 + 3X^2 + 4X + 6$ et $g = 2X^4 + X^3 + 5$ dans $(\mathbb{Z}/7\mathbb{Z})[X]$
- $f = X^3 + X^2 + X + 10$ et $g = X^2 + 2X + 8$ dans $(\mathbb{Z}/11\mathbb{Z})[X]$
- $f = 5X^4 + X^3 + 8X^2 + 3$ et $g = 2X^3 + X^2 + 12X + 1$ dans $(\mathbb{Z}/13\mathbb{Z})[X]$

1.5. **Montrer que les polynômes suivants sont irréductibles sur $\mathbb{F}_p$.** Donner une construction de $\mathbb{F}_{p^{\deg(f)}}$ et écrire la table de multiplication.

- $f = X^2 + X + 1$, $p = 2$
- $f = X^2 + 1$, $p = 3$
- $f = X^3 + X + 1$, $p = 2$
- $f = X^4 + X^3 + 1$, $p = 2$

1.6. **Trouver tous les polynômes unitaires sur $\mathbb{F}_3$ irréductibles de degré 3.** Trouver tous les polynômes unitaires sur $\mathbb{F}_2$ irréductibles de degré 4. Proposer plusieurs constructions possibles de $\mathbb{F}_{27}$ et $\mathbb{F}_{16}$.

1.7. **Listes d'éléments de $\mathbb{F}(p^d)$, l'addition et multiplication.**
(voir le fichier fflists_with_pari12-1.gp). Utiliser le programme suivant

```
Chiffres(d,l,n)=
{
 \\ integer n, base d
 \\ vector length l
 local(i,m,v);
 v=vector(l);
 m=n;
for(i = 0,l-1,
v[l-i]=m%d;
m=floor(m/d));
return(v);
} /* end of Chiffres */
\\\\\\\\\\\\\\\\\\\\\\\
{
   p=3;d=2;
   G=ffinit(p,d);
\\G=Mod(1, 2)*x^4 + Mod(1, 2)*x^3 + Mod(1, 2);
   \\G=Mod(1, 2)*x^3 + Mod(1, 2)*x^2 + Mod(1, 2);
}
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
add(k,l)=
{
local(ck,pk,cl,pl,a);
ck=Chiffres(p,d,k);
pk=Pol(ck);
pkm=Mod(pk,G);
cl=Chiffres(p,d,l);
pl=Pol(cl);
plm=Mod(pl,G);;
a=subst(lift(lift((pkm+plm))*Mod(1,p)), x, p);
;
return(a);
}/*end of add*/
\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\
mul(k,l)=
{
local(ck,pk,cl,pl,a);
ck=Chiffres(p,d,k);
pk=Pol(ck);
pkm=Mod(pk,G);
cl=Chiffres(p,d,l);
pl=Pol(cl);
plm=Mod(pl,G);;
m=subst(lift(lift((pkm*plm))*Mod(1,p)), x, p);
;
return(m);
}/*end of mul*/
\\\\\\\\\\\\\\\\\\\\\\\\\
{
for(k=0,p^d-1,
```

```
ck=Chiffres(p,d,k);
Pol(ck);
pkm=Mod(pk,G);
print(k,"\t", lift(Pol(ck)),"\t", "G="lift(G))
);
}
0       0       G=x^2 + x + 2
1       1       G=x^2 + x + 2
2       2       G=x^2 + x + 2
3       x       G=x^2 + x + 2
4       x + 1   G=x^2 + x + 2
5       x + 2   G=x^2 + x + 2
6       2*x     G=x^2 + x + 2
7       2*x + 1 G=x^2 + x + 2
8       2*x + 2 G=x^2 + x + 2
gp > add(7,8)
% = 3
```

1.8. **Calculer l'inverse de $x \in \mathbb{F}_{p^{\deg(f)}} \cong \mathbb{F}_p[X]/(f)$, où α est l'image de X dans $\mathbb{F}_p[X]/(f)$.**

```
gp > f = Mod(1,2)*(X^4 + X + 1);f
% = Mod(1, 2)*X^4 + Mod(1, 2)*X + Mod(1, 2)
gp > a=ffgen(f)
% = X
gp > a^2
% = X^2
gp > a^4
% = X + 1
gp > a^8
% = X^2 + 1
gp > a^16
% = X
```

Soit α la classe résiduelle de X mod f. Nous déterminons l'inverse de $\alpha+1$. Pour cela, nous utilisons l'algorithme d'Euclide étendu. Nous avons

$$f(X) = (X+1)q(X) + 1 \text{ avec } q(X) = X^7 + X^6 + X^5 + X^4 + X^2 + X$$

et nous obtenons le tableau suivant :

k	0	1	2	3
r_k	f	$X+1$	1	0
q_k		$q(X)$	$X+1$	
x_k	1	0	1	
y_k	0	1	$q(X)$	

et nous trouvons ainsi $f(X) - q(X)(X+1) = 1$. Par conséquent, la classe résiduelle $\alpha^7 + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + \alpha$ de $q(X)$ est inverse de $\alpha + 1$.

```
  gp >  a=ffgen(f)
%  = X
gp > 1/(a+1)
%  = X^7 + X^6 + X^5 + X^4 + X^2 + X
```

- $f = X^2 + 1$, $p = 19$, $x = 13\alpha + 7$
- $f = X^5 + 4X + 3$, $p = 5$, $x = 3\alpha^4 + \alpha^3 + \alpha + 2$
- $f = X^7 + 6X + 4$, $p = 7$, $x = 4\alpha^6 + 2\alpha^4 + \alpha^3 + \alpha + 5$

Algorithme d'Euclide étendu pour les polynômes. Justifier l'algorithme suivant et donner un exemple de calcul :

L'algorithme d'Euclide étendu

Entrée : deux polynômes a et b *non nuls* **Sortie :** deux polynômes x et y tels que $au + bv = pgcd(a, b)$

$(u_0, u_1, v_0, v_1) \leftarrow (1, 0, 0, 1)$

$s \leftarrow 1$

tant que $b \neq 0$

$r \leftarrow a \bmod b$

$q \leftarrow \dfrac{a - r}{b}$

$a \leftarrow b$

$b \leftarrow r$

$u \leftarrow u_1$

$v \leftarrow v_1$

$u_1 \leftarrow qu_1 + u_0$

$v_1 \leftarrow qv_1 + v_0$

$u_0 \leftarrow u$

$v_0 \leftarrow v$

$s \leftarrow -s$

$u \leftarrow su_0$

$v \leftarrow -sv_0$

retourner (u, v)

(s est le signe $(-1)^k$).

Prouver la validité de cet algorithme Étudier la fonction "bezout" en PARI/GP pour les polynômes, et donner des exemples.

1.9. **Le groupe multiplicatif des inversibles d'un corps fini.** Soit $p = 2$ et soit $f(x) = x^8 + x^4 + x^3 + x + 1$. Montrer que ce polynôme est irréductible dans $\mathbb{F}_2[x]$.

Indication : Utiliser l'application de Frobenius $x \mapsto x^p$ pour $p = 2$ et son action sur l'ensemble des racines α dans un corps fini contenant $\mathbb{F}_p$.

```
gp > f=(x^8 +x^4 +x^3 + x + 1)*Mod(1,2);f
% = Mod(1, 2)*x^8 + Mod(1, 2)*x^4 + Mod(1, 2)*x^3 + Mod(1, 2)*x + Mod(1, 2)
gp > for(k=1,8,print(k,"\t",lift(x^(2^k)%f)))
1       x^2
2       x^4
3       x^4 + x^3 + x + 1
4       x^6 + x^4 + x^3 + x^2 + x
5       x^7 + x^6 + x^5 + x^2
6       x^6 + x^3 + x^2 + 1
7       x^7 + x^6 + x^5 + x^4 + x^3 + x
8       x
```

Calculer $f'(x)$ et trouver $\text{pgcd}(f, f')$.

1.10. **Structure du groupe des éléments inversibles d'un corps fini.** Le résultat principal est que ce groupe est toujours engendré par un élément. Trouver l'ordre du polynôme f et un élément primitif dans $\mathbb{F}_p[x]/(f)$. Construir un polynôme primitif de degré 8.

```
gp > fforder(a)
% = 51
```

```
gp > b=ffprimroot(a)
% = x^7 + x^6 + x^5 + x
gp > lift(minpoly(b,X))
% = X^8 + X^6 + X^4 + X^3 + X^2 + X + 1
```

Ceci est particulièrement intéressant en cryptographie car on va chercher à travailler avec des éléments de très grand ordre.

On remarque que si K est un corps fini ayant q éléments, alors le groupe K^* possède forcément $q-1$ éléments car seul 0 n'est pas inversible.

1.11. **Étudier les fonctions sur les corps finis sur PARI/GP.**

```
?ffinit
ffinit(p,n,{v=x}): monic irreducible polynomial of degree n over Fp[v]
gp > ffinit(2,8)
%
5 = Mod(1, 2)*x^8 +  Mod(1, 2)*x^6 + Mod(1, 2)*x^5 +  Mod(1, 2)*x^4 + Mod(1, 2)*
^3 + Mod(1, 2)*x + Mod(1, 2)
gp > lift(ffinit(2,8))

% 6 = x^8 + x^6 + x^5 + x^4 + x^3 + x + 1
```

1.12. **Exemples : Tables de l'addition et de la multiplication dans $\mathbb{F}_{p^d}$.** Voir le fichier fflists_with_pari12-2.gp.

•Le cas de $\mathbb{F}_8$

```
gp > Chiffres(d,l,n)=
{
 \\ integer n, base d
 \\ vector length l
 local(i,m,v);
 v=vector(l);
 m=n;
for(i = 0,l-1,
v[l-i]=m%d;
m=floor(m/d));
return(v);
} /* end of Chiffres */
gp > {
   p=2;
   d=3;
   \\ G=ffinit(p,d);
   G=Mod(1, 2)*x^3 + Mod(1, 2)*x^2 + Mod(1, 2);
   s=ffgen(G,s);
   u=ffprimroot(s)
}
% = s
gp > Mac=matrix(p^d,p^d,k,l, 0);
gp > {
for(k=0,p^d-1,
for(l=0,p^d-1,
ck=Chiffres(p,d,k);
pk=Pol(ck);
pkm=Mod(pk,G);
```

```
cl=Chiffres(p,d,l);
pl=Pol(cl);
plm=Mod(pl,G); ;
Mac[k+1,l+1]= subst(lift(lift((pkm+plm)*Mod(1,p))), x, p);
));Mac
}
% =
[0 1 2 3 4 5 6 7]

[1 0 3 2 5 4 7 6]

[2 3 0 1 6 7 4 5]

[3 2 1 0 7 6 5 4]

[4 5 6 7 0 1 2 3]

[5 4 7 6 1 0 3 2]

[6 7 4 5 2 3 0 1]

[7 6 5 4 3 2 1 0]

gp > Mmc=matrix(p^d,p^d,k,l, 1);
gp > {
for(k=1,p^d-1,
for(l=1,p^d-1,
ck=Chiffres(p,d,k);
pk=Pol(ck);
pkm=Mod(pk,G);
cl=Chiffres(p,d,l);
pl=Pol(cl);
plm=Mod(pl,G); ;
Mmc[k,l]= subst(lift(lift((pkm*plm)*Mod(1,p))), x, p);
));Mmc
}
% =
[1 2 3 4 5 6 7]

[2 4 6 5 7 1 3]

[3 6 5 1 2 7 4]

[4 5 1 7 3 2 6]

[5 7 2 3 6 4 1]

[6 1 7 2 4 3 5]

[7 3 4 6 1 5 2]
```

1.13. **Exemple de factorisation dans $\mathbb{F}_{p^d}[x]$ avec PARI/GP.** Voir le fichier factorff_with_pari.gp. En utilisant le modèle suivant, donner votre exemple de factorisation dans $\mathbb{F}_{p^d}[x]$ avec $p = 3, d = 5$:

```
{
p=3;
d=2;
G=ffinit(p,d,a);
a=ffgen(G,a);
\\
P=x^4-1;
\\
F=lift(lift(factorff(P,p,G)));
print("factorisation de  ",P,"  :  ");
for(i=1,#(F~),print("(",F[i,1],")^",F[i,2]));
}
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
gp > {
p=3;
d=2;
G=ffinit(p,d,a);
a=ffgen(G,a);
\\
P=x^4-1;
\\
F=lift(lift(factorff(P,p,G)));
print("factorisation de  ",P,"  :  ");
for(i=1,#(F~),print("(",F[i,1],")^",F[i,2]));
}
factorisation de  x^4 - 1  :
(x + 1)^1
(x + 2)^1
(x + (a + 2))^1
(x + (2*a + 1))^1
gp > fforder(a)
% = 8
gp > G
% = Mod(1, 3)*a^2 + Mod(1, 3)*a + Mod(2, 3)
gp > lift(G)
% = a^2 + a + 2
```

1.14. **Théorème chinois des restes pour les polynômes.** Soient $m_1, \ldots, m_n \in K[X]$ des polynômes de degrés positifs sur un corps K, deux-à-deux premiers entre eux, et $a_1, \ldots, a_n$ des polynômes. On considère les congruences simultanées dans $K[X]$:

$$u \equiv a_1 \bmod m_1, u \equiv a_2 \bmod m_2, \ldots u \equiv a_n \bmod m_n.$$

Posons

$$m = \prod_{i=1}^{n} m_i,\ M_i = m/m_i,\quad 1 \leq i \leq n$$

On utilise l'algorithme d'Euclide étendu pour calculer des polynômes $v_i \in K[X]$, $i \leq i \leq n$ tels que

$$v_i M_i \equiv 1 \bmod m_i \text{ (puisque } \operatorname{pgcd}(M_i, m_i) = 1)$$

puis on pose

$$u \equiv \sum_{i=1}^{n} a_i v_i M_i \bmod m.$$

Prouver la validité de cet algorithme. Écrire un programme qui l'implémente.

2. Primalité et fabrication de nombres premiers

Nombres pseudopremiers de Fermat ; Test de Miller-Rabin

2.1. **Rappels sur l'exponentiation rapide et nombres pseudopremiers de Fermat.** Prouver la validité de l'algorithme suivant :

Entrée :
– Entier n de congruence.
– Puissance e.
– Élément a de $\mathbb{Z}/m\mathbb{Z}$
Sortie :
– Valeur de a^e dans $\mathbb{Z}/m\mathbb{Z}$.
Algorithme :
1. $x := a$, $y = e$;
2. $z := 1$;
3. Tant que y n'est pas nul
   3.1 si y est impaire, $z := z * x \bmod n$
   3.2 $x := x * x \bmod n$
   3.3 $y := \lfloor y/2 \rfloor$
4. renvoyer z

Justifier par calcul l'exemple suivant : Soit $p = 323$. Est-ce que p est premier ? On calcule 2^{322} modulo 323. On organise les calculs dans une table :

i	m	m_i	$2^{2^i} \bmod 323$
0	322	0	2
1	161	1	4
2	80	0	16
3	40	0	256
4	20	0	290
5	10	0	120
6	5	1	188
7	2	0	137
8	1	1	35

Alors

$$2^{322} \equiv 4 \cdot 188 \cdot 35 \equiv 157 \pmod{323},$$

donc 323 n'est pas premier. En effet, $323 = 17 \cdot 19$.

2.2. **Test de Miller-Rabin.** Ce test de Miller-Rabin peut prouver la composition de n'importe quel entier positif. Pour faire simple, l'analogue des nombres de Carmichael n'existe pas pour le test de Miller-Rabin. Le test de Miller-Rabin est basé sur une modification du petit théorème de Fermat La situation est la suivante. Soit n un entier positif impair ; notons

$$s = \{\max r \in \mathbb{N} : 2^r \text{ divise } n-1\}$$

autrement dit, 2^s est la plus grande puissance de 2 qui divise $n-1$. Posons

$$d = (n-1)/2^s$$

Par définition d est impair.

Théorème 1 (rappel de cours)**.** *Si n est premier et si a est un entier premier à n, avec la nolation précédente, nous avons soit*

$$a^d \equiv 1 \bmod n \tag{1}$$

ou bien il existe r *dans l'ensemble* $\{0, 1, \cdots, s-1\}$ *avec*

$$a^{2^r d} \equiv -1 \bmod n \tag{2}$$

Démonstration. Soit a un entier premier à n. Parce que n est un nombre premier l'ordre du groupe multiplicatif des résidus mod$n$ est $n-1 = 2^s d$. Par consequence, l'ordre k des classes residuelles $a^d + n\mathbb{Z}$ est un puissance de 2. Si cet ordre est $k = 1 = 2^0$, alors

$$a^d \equiv 1 \bmod n.$$

Si $k > 1$, alors $k = 2^l$ avec $1 \leq l \leq s$. Par conséquent, la classe residuelle $a^{2^{l-1}d} + n\mathbb{Z}$ est d'ordre 2. D'après l'exercice le seul élément d'ordre 2 dans $(\mathbb{Z}/n\mathbb{Z})^*$ est $-1 + n\mathbb{Z}$, ce qui implique

$$a^{2^r d} \equiv -1 \bmod n$$

pour $r = l - 1$.

Si n est un nombre premier, alors au moins une des conditions du théorème est vérifiée : Par conséquent, si l'on trouve un entier a premier à n qui ne satisfait ni (1) ni (2) pour un certain $r \in \{0, \ldots, s-1\}$, on a la preuve que n est composé. On dit qu'un tel entier est *un témoin* de la composition de n.

2.3. **Justifier par calcul l'exemple suivant.**

Exemple 2 (rappel de cours). *Soit* $n = 561$. *Puisque* n *est un nombre de Carmichael, le test de Fermat ne peut pas prouver qu'il est compose mais* $a = 2$ *est un témoin qui prouve que* n *est composé. En effet, nous avons* $s = 4$, $d = 35$ *et* $2^{35} \equiv 263 \bmod 561$, $2^{2*35} \equiv 166 \bmod 561$, $2^{4*35} \equiv 67 \bmod 561$, $2^{8*35} \equiv 1 \bmod 561$. *Par conséquent, le théorème prouve que 561 est composé.*

Pour garantir l'efficacité du test de Miller-Rabin, il faut qu'il y ait suffisamment de temoins permettant de prouver qu'un nombre est composé. Dans le prochain théorème, on verra que c'est toujours le cas.

Théorème 3 (rappel de cours)**.** *Si* $n \geq 3$ *est un nombre compose impair ; l'ensemble* $\{1, \ldots, n-1\}$ *contient au plus* $(n-1)/4$ *nombres premiers à* n *qui ne ne sont pas des témoins de la composition de* n. □

Pour appliquer le test de Miller-Rabin à un entier impair positif, n, nous choisissons au hasard un nombre $a \in \{2, 3, \ldots, n-1\}$. Si $\mathrm{PGCD}(a, n) > 1$, alors n est composé. Sinon, nous calculons $a^d, a^{2d}, \ldots, a^{2^{s-1}d}$. Si nous trouvons un témoin de la composition de n, nous avons preuve que n est composé. D'après le cours la probabilité que n soit composé et qu'on ne trouve pas de témoin est au plus $1/4$. Si nous répétons le test de Miller-Rabin t fois et si n est composé, la probabilité de n pas trouver de témoin est au plus $(1/4)^t$. Par exemple, pour $t = 10$, cette probabilité est au plus $1/2^{20} \sim 1/10^6$, ce qui est très improbable.

2.4. **Trouver le plus petit témon de la composition du nombre de Carmichael 1729 pour le test de Miller-Rabin.**

2.5. **Écrire un prograrmne qui implemente le test de Miller-Rabin.** Utiliser ce prograrmne pour determiner le plus petit nombre premier de 512 bits.

Réponse : 13407807929942597099574024998205846127479365820592393377723561443721764030
073546976801874298166903427690031858186486050853753882811946569946433649006084171

Dans de nombreux cryptosystemes à clés publique, on a besoin de nombres premier aléatoires ayant un nombre fixé de bits. Nous decrivons la construction de tels nombres premiers aleatoires.

On veut générer, au hasard, un nombre premier de k bits. Alors, on génère de façon aléatoire un nombre impair de k bits. Pour cela, on donne au premier et au demier bit de n la valeur 1, pendant que les $k-2$ bits restants sont choisis au hasard avec la distribution uniforme. Ensuite, on teste si ce nombre est premier. D'abord, on teste s'il est divisible par un nombre premier inférieur à une borne B prédéfinie ; typiquement $B = 10^6$. Si aucun diviseur premier de n n'a été trouvé, nous lui appliquons le test de Miller-Rabin t fois. Le choix $t = 3$ suffit pour avoir une probabilié d'erreur

inférieure à $(1/2)^{80}$ quand $k \geq 1000$. Si ce test ne trouve pas de témoine la composition de n, le nombre n est considéré comme premier.

3. Chiffrement à clés publiquesSchémas de chiffrement asymétriques et cryptographie à clef publique

Rappels sur le protocole RSA, Chiffrement de Rabin, Échange de clés selon Diffie-Hellman, Le protocole ElGamal

3.1. **Exemple de cryptage avec RSA (rappel).** On considère avec les notations usuelles, une clef publique RSA :

$$(n, e) = (1732657411, 100001),$$

et soit $\mathcal{E} : x \mapsto x^e \bmod n$ l'application de cryptage de l'ensemble $\mathbb{Z}/n\mathbb{Z}$ dans $\mathbb{Z}/n\mathbb{Z}$.

3.2. En utilisant PARI/GP trouver la valeur de la fonction d'Euler $\varphi(n)$.

3.3. Calculer la clef secrète d correspondente, telle que l'application $\mathcal{D} : y \mapsto y^d \bmod n$ soit réciproque de $\mathcal{E}$.

3.4. Etudier en PARI le programme disponible à l'adresse :
http://www-fourier.ujf-grenoble.fr/ panchish/tp/pariCRY/

```
RSAcrypt(PubKey, Message)=
{
 \\ PubKey should be a pair of positive integers [N,e]
 \\ We assume that N>Message.

 local(e,N);
 N=PubKey[1];
 e=PubKey[2];
 C=gcd(Message,N);
 if(C!=1,print("RSA module is broken. A factor is ",C)
 , /* ELSE */
  C=lift(Mod(Message,N)^e);
 );
 return(C);
} /* end of RSAcrypt */

RSAdecrypt(PrivateKey, CipheredMessage)=
{
 \\ PrivateKey is either a pair [N,d] or a quadruple [p,q,dp,dq]
 \\ of positive integers. In the last case, dp is the remainder of
 \\ d modulo p-1 and dq is the remainder of d modulo q-1.

 local(d,dp,dq,N,p,q);
 if(length(PrivateKey)==2,
  N=PrivateKey[1];
  d=PrivateKey[2];
  M=lift(Mod(CipheredMessage,N)^d);
 ,/* ELSE */
  p=PrivateKey[1];
  q=PrivateKey[2];
  dp=PrivateKey[3];
  dq=PrivateKey[4];
  M=lift(chinese(Mod(CipheredMessage,p)^dp,Mod(CipheredMessage,q)^dq));
 );
```

```
 return(M);
} /* end of RSAdecrypt */
```

3.5. **Chiffrer le message "FERMAT" avec un nombre $M \bmod n$ à trouver.** Utiliser l'alphabeth, $N = 26, k = 6, l = 7$

```
A=0,B=1,C=2,D=3,E=4,F=5,G=6,H=7,I=8
J=9,K=10,L=11,M=12;N=13,O=14,P=15,Q=16,R=17,S=18,T=19,U=20,V=21,W=22,X=23,Y=24,Z=25
et la procedure en PARI

Chiffres(d,l,n)=
{
 \\ integer n, base d
 \\ vector length l
 local(i,m,v);
 v=vector(l);
 m=n;
for(i = 0,l-1,
v[l-i]=m%d;
m=floor(m/d));
return(v);
} /* end of Chiffres */
```

et la clef publique RSA :

$$(n, e) = (1732657411, 100001).$$

3.6. **Représenter le message chiffré avec les lettres.**

3.7. **Décrypter le message C avec la clef secrète d.**

3.8. **Donner votre exemple de chiffrement avec RSA.** en utilisant des nombres premiers aléatoires

```
(10:04) gp > p=nextprime(random(10^30))
%34 = 358945505295465214927115886019
(10:04) gp > q=nextprime(random(10^30))
%35 = 616527181647127617059107667887
```

3.9. **Établissement d'une clé secrète selon Diffie-Hellman.** Justifier l'exemple suivant :

1. $p = 11111117$ et $g = 111112$
2. A calcule $a = 111112^{1234} \bmod 11111117 = 7218868$
3. et B $b = 111112^{876} \bmod 11111117 = 8671412$
4. A calcule $k = 8671412^{1234} \bmod 11111117 = 6146319$
5. et B $k' = 7218868^{876} \bmod 11111117 = 6146319$

3.10. **Donner votre exemple d'établissement d'une clé secrète selon Diffie-Hellman.** Utiliser le modèle suivant :

```
gp > p=nextprime(20^35);g=znprimroot(p);ka=random(p-1);
gp > lift(ka)
% = 133934925351222510469226809889190757285217 4757
gp > lift(kb)
% = 7931950802952951093129452809 0493
gp > lift((g^ka)^kb)
% = 2860897805152686834654042219551166811575791368
gp > lift((g^kb)^ka)
% = 2860897805152686834654042219551166811575791368
```

3.11. **Chiffrement selon ElGamal.** Justifier l'exemple suivant

1. $p = 11111117, g = 111112$ et $a = 1234$
2. A calcule $k_a = 111112^{1234} \bmod 11111117 = 7218868$
3. A publie $p = 11111117, g = 111112, k_a = 7218868$

El-Gamal exemple (suite)

1. B choisit $b = 876$ et a pour message 99999
2. B calcule $c_1 = 111112^{876} \bmod 11111117 = 8671412$
3. et $c_2 = 99999 * 7218868^{876} \bmod 11111117 = 3205709$
4. B envoie (8671412, 3205709)

1. A calcule $d_1 = 8671412^{-1234} \bmod 11111117 = 5300581$
2. et $d_2 = 5300581 * 3205709 \bmod 11111117 = 99999$

3.12. **Donner votre exemple de chiffrement selon ElGamal.**

3.13. **Chiffrement de Rabin.** La sécurité du système de Rabin repose aussi, sur la difficulté de factoriser les entiers. Mais, contrairement à RSA, on peut montrer que celui qui peut casser le systeme de Rabin efficacement peut tout efficacement factoriser les entiers.

Fabrication des clés

Alice choisit au hasard deux grands nombres premiers p et q avec $p \equiv q \equiv 3 \bmod 4$ La fabrication des nombres premiers se fait avec les nombres premiers aléatoires. Ces propriétés servent à rendre le chiffrement plus efficace. Mais, comme nous allons voir plus loin, le système de Rabin fonctionnerait aussi sans elles. Alice calcule $n = pq$. Sa clé publique est n. Sa clé privée est la paire (p, q).

Chiffrement

Comme dans le systeme RSA, l'espace des messages en clair est I'ensemble $\{0, \ldots, n-1\}$. Pour chiffrer Ie message en clair $m \in \{0, \ldots, n-1\}$, Bob utilise la clé publique n d'Alice et calcule le cryptogramme

$$c = m^2 \bmod n$$

Comme RSA, le système de Rabin peut etre utilise pour implementer un chiffrernen par bloc.

Déchiffrement

Alice reconstitue le message en clair m en extrayant la racine carrée modulo n du cryptogramme c, ce qu'elle fait de la facon suivante. Elle calcule

$$m_p = c^{(p+1)/4} \bmod p, m_q = c^{(q+1)/4} \bmod q$$

Alors $\pm m_p + p\mathbb{Z}$ sont les deux racines carrées de $c + p\mathbb{Z}$ dans $\mathbb{Z}/p\mathbb{Z}$, et $\pm m_q + q\mathbb{Z}$ sont les deux racines carrees de $c + q\mathbb{Z}$ dans $\mathbb{Z}/q\mathbb{Z}$. Cette méthode pour calculer les racines carrées de $c \bmod p$ et $\bmod q$ fonctionne seulement parce que p et q sont tous les deux congrus à 3 mod 4.

3.14. **Donner votre exemple de chiffrement et déchiffrement de Rabin.** Chiffrer par exemple le même message "FERMAT" avec un nombre $M \bmod n$ à trouver, en utilisant l'alphabeth. Utiliser

```
 p= 36971830199
 q= 24406008779
gp > N
% = 902334812412491317021
gp > C=Rcrypt(N,m);C
% = 3787381700473849
```

Voir le fichier ChRabin.gp. Utiliser le programme suivant :

```
B=10^10;Rabin(B)={
local(p,q,k);
k=1;p=3;
Rp=random(B);
p=Rp*4+3+4*k;
while(isprime(p)<1,
k=k+1;
p=Rp*4+3+4*k;
```

```
);
p=Rp*4+3+4*(k);

       return(p)
}

p=Rabin(B);q=Rabin(B);N=p*q;
s1=Mod(-1,p); s3=Mod(-1,q);

chinese(Mod(1,p),Mod(-1,q));

PrivateKey=[p,q];
Rcrypt(PubKey, Message)=
{
 \\ PubKey should be a positive integer N
 \\ We assume that N>Message.
 local(N);
 N=PubKey;
 C=gcd(Message,N);

 if(C!=1,print("R module is broken. A factor is ",C)
 , /* ELSE */
  C=lift(Mod(Message,N)^2);
 );

 return(C);
} /* end of Rcrypt */

Rdecrypt(PrivateKey, CipheredMessage)=
{
 \\PrivateKey is a pair [p,q]
 local(N,p,q);
  p=PrivateKey[1];
  q=PrivateKey[2];
  M=vector(4);
  M[1]=lift(chinese(Mod(CipheredMessage,p)^((p+1)/4),
Mod(CipheredMessage,q)^((q+1)/4)));

  M[2]=lift(chinese(s1*Mod(CipheredMessage,p)^((p+1)/4),
  Mod(CipheredMessage,q)^((q+1)/4)));

  M[3]=lift(chinese(Mod(CipheredMessage,p)^((p+1)/4),
  s3*Mod(CipheredMessage,q)^((q+1)/4)));
  M[4]=lift(chinese(s1*Mod(CipheredMessage,p)^((p+1)/4),
  s3*Mod(CipheredMessage,q)^((q+1)/4)));

return(M);
} /* end of Rdecrypt */
gp > p
% = 36971830199
gp > N
%9 = 902334812412491317021
gp > m=11111111111111111122;m
```

```
% = 11111111111111111122
gp > C=Rcrypt(N,m);C
% = 69893062469555128183
gp > M=Rdecrypt([p,q],C);M
% = [472294857582160028771, 891223701301380205899, 11111111111111111122,
430039954830331288250]
```

4. Méthodes de factorisation

4.1. **Méthode p - 1 de Pollard.** Voir le fichier fPollard.gp. Il y a des algorithmes de factorisation qui marchent particulièrement bien pour les entiers composés ayant certaines propriétés. On doit éviter d'utiliser ces entiers comme module RSA ou Rabin. Comme exemple d'un tel algorithme de factorisation, nous décrivons la méthode $p-1$ de John Pollard. La méthode $p-1$ fonctionne au mieux pour un entier compose n ayant un facteur premier p pour lequel $p-1$ n' a que des petits diviseurs premiers. II est alors possible de determiner un multiple k de $p-1$ sans connaitre $p-1$ (voir les détails ci-dessous). Avec ce nombre k, le petit théorème de Fermat donne

$$a^k \equiv 1 \bmod n.$$

L'algorithme calcule $g = PGCD(a^k - 1, n)$ pour une base a convenable. Si aucun diviseur de n n' a été trouvé, on essaie une nouvelle borne B.

Si les puissances des nombres premiers qui divisent $p-1$ sont toutes inferieures à B, l'entier k est un multiple de $p-1$.

4.2. **Vérifier par calcul l'exemple suivant.**

Exemple 4. *Le nombre composé $n = 1241143$ restait à factoriser. Nous utilisons $B = 13$. Alors $k = 8*9*5*7*11*13$ et*

$$PGCD(2^k - 1, n) = 547$$

Donc, $p = 547$ est un diviseur de n. Le cofacteur est $q = 2269$. On peut verifier que les nombres 547 et 2269 sont premiers.

```
  gp > factor(1241143)
%28 =
[547 1]
[2269 1]

> fPoll(n,B,a)={
local(k,l,m,ep, g, d);
p=2;
k=1;
ep=floor(log(B)/log(p));
k=k*(p^(ep));
m=a^k% n;
l=(m-1)% n;
g=gcd(l,n);d=g;
     while((g<2)&(nextprime(p+1)<B),
p=nextprime(p+1);
ep=floor(log(B)/log(p));
k=k*(p^(ep));
m=(m^(p^ep))% n;
el=(m-1)% n;
g=gcd(el,n);
d=g;
```

```
return([n,k,d, n/d]);

gp > fPoll(2^2+1,65,5)
%  = [4294967297, 64, 641, 6700417]
```

4.3. **Donner votre exemple de factorisation par la méthode p-1.** Utiliser les programmes ci-dessus.

4.4. **Factorisation de Fermat.** Si $n = x^2 - y^2$, alors $x - y$ est dans beaucoup de cas un facteur non-trivial de n.

Cette observation simple amène à "l'algorithme de factorisation de Fermat" qui nécessite en général $O(n^{1/2})$ opérations mais il est plus efficace si n est une produit de nombres t, s ayant petit différance. Alors $n = x^2 - y^2$ où $x = (t + s)/2$, $y = (t - s)/2$. L'algorithme consiste de calcule de $x^2 - n$ pour x à partir de $[\sqrt{n}] + 1$ jusqu'un carré parfait a été trouvé.

4.5. **Justifier par calcul l'exemple suivant.**

EXEMPLE 5. *Soient* $n = 7429, x = 173, y = 150$. *Alors* $x^2 - y^2 = n$, $x - y = 23$, $x + y = 323$, $[\sqrt{n}] = 86$.

```
 > factor(7429)
%26 =
[17 1]
[19 1]
[23 1]
```

4.6. **Justifier l'algorithme de factorisation de Fermat suivant.**

```
fFermat(n)={
local(x,y,x1,x2,s);

s=floor(sqrt(n));
y=1;
x=n;
x1=floor(sqrt(n+1));
x2=n+1;
while(x^2-y^2-n,y=y+1;
x1=floor(sqrt(n+y^2));
x2=n+y^2;
x=x1;
);
return([x, y, s, n, x-y,x+y]);
}

gp > fFermat(1241143)
%  = [1408, 861, 1114, 1241143, 547, 2269]
gp > #
   timer = 1 (on)
gp > fFermat(2^32+1)
time = 7,534 ms.

% 3 = [3350529, 3349888, 65536, 4294967297, 641, 6700417]
```

5. Logarithmes discrets

5.1. **Algorithme pas de bébé/ pas de géant de Shanks.** Une amélioration considerable de l'algorithme d'énumération est l'algorithme pas de bébé/ pas de géant de D.Shanks. Cet algorithme demande moins d'opérations du groupe mais plus de stockage. Nous le décrivons maintenant. Nous posons

$$m = [\sqrt{n}] + 1$$

et nous écrivons le logarithme discret inconnu x sous la forme

$$x = qm + r, \quad 0 \leq r < m$$

Donc, r est le *reste* et q le *quotient* de la division de x par m. Pour trouver x, l'algorithme pas de bébé/pas de géant calcule q et r. II opère de la façon suivante. La relation

$$\gamma^{qm+r} = \gamma^x = \alpha$$

donne

$$(\gamma^m)^q = \gamma^x \gamma^{-r} = \alpha\gamma^{-r}$$

Nous calculons d'abord l'ensemble des pas de bébé

$$B = \{\alpha\gamma^{-r} : 0 \leq r < m\}$$

Si nous trouvons dans cet ensemble un couple de la forme $(1, r)$, nous avons $\alpha\gamma^{-r} = 1$, ou encore $\alpha = \gamma^r$; nous pouvons poser $x = r$ et cet x est le plus petit possible. Si nous ne trouvons pas de couple de cette forme, nous calculons

$$\delta = \gamma^m$$

puis nous testons, pour $q = 1, 2, 3, \ldots$, si l'élément δ^q du groupe est la premiere composante d'un élément de B autrement dit, s'Il existe un couple de la forme (δ^q, r) dans B. Dès que nous en avons un, nous ecrivons

$$\alpha\gamma^{-r} = \delta^q = \gamma^{qm}$$

ce qui donne

$$\alpha = \gamma^{qm+r} = \gamma^x$$

et par conséquent, le logarithme discret est $qm + r$.

Les éléments δ^q, avec $q = 1, 2, 3 \ldots$ sont les *pas de géant*. Nous devons comparer chaque δ^q avec toutes les premières composantes des pas de bébé. Pour rendre efficace cette comparaison, les éléments de B sont stockés dans une *table de hachage*.

5.2. **Algorithme pas de bébé/ pas de géant (exemple).** Justifier par un programme de calcul l'exemple suivant.

Dans $(\mathbb{Z}/2017\mathbb{Z})^*$, déterminons le logarithme discret de 3 dans la base 5. Nous avons $\gamma = 5 + 2017\mathbb{Z}$, $\alpha = 3 + 2017\mathbb{Z}$, $m = [\sqrt{2017}] + 1 = 45$. Si nous représentons les classes résiduelles par leur plus petit représentant non négatif, l'ensemble des pas de bébé est

$$\begin{aligned} B = \{&(3,0),(404,1),(1291,2),(1065,3),(213,4),(446,5),(896,6),\\ &(986,7),(1004,8),(1411,9),(1089,10),(1428,11),(689,12),(1348,13),\\ &(673,14),(538,15),(511,16),(909,17),(1392,18),(1892,19),(1992,20),\\ &(2012,21),(2016,22),(1210,23),(242,24),(1662,25),(1946,26),\\ &(1196,27),(1046,28),(1016,29),(1010,30),(202,31),(1654,32),\\ &(1541,33),(1115,34),(223,35),(448,36),(493,37),(502,38),(1714,39),\\ &(1553,40),(714,41),1353,42),(674,43),(1345,44)\}. \end{aligned}$$

Indication :

```
gp > al=Mod(3,2017);ga=Mod(5,2017);
gp > de=ga^45
% 47 = Mod(45, 2017) B=vector(45);
gp > for(r=0,44,B[r+1]=al*ga^(-r))
gp > for(r=0,44,print(lift(B[r+1]),  r))
```

Alors nous calculons $\delta = \gamma^m = 45 + 2017\mathbb{Z}$, ce qui donne les pas de géant
$45, 8, 360, 64, 863, 512, 853, 62, 773, 496, 133, 1951,$
$1064, 1489, 444, 1827, 1535, 497, 178, 1959, 1424, 1553.$
`gp > for(q=1,45,print(lift(de^(q))))`

Nous remarquons que (1553,40) figure parmi les pas de bébé. Par consequent, $\alpha\gamma^{-40} = 1553 + 2017\mathbb{Z}$. Puisque 1553 a été trouvé comme le 22^e pas de géant, nous obtenons d'abord

$$\gamma^{22*45} = \alpha\gamma^{-40}, \text{ puis } \gamma^{22*45+40} = \alpha, x = 22 * 45 + 40 = 1030$$

5.3. **Algorithme ρ de Pollard.** L'algorithme de Pollard décrit ici a le meme temps d'execution, $O(\sqrt{|G|})$, que l'algorithme pas de bébé/pas de géant. Cependant, son besoin de stockage est constant, contrairement a l' algorithme pas de bébé/pas de géant qui nécessite de stocker environ $\sqrt{|G|}$ éléments du groupe.

Une fois encore, nous voulons resoudre le probleme DL.Nous avons besoin de trois sous-ensembles de G, deux a deux disjoints, G_1, G_2, G_3, tels que $G_1 \cup G_2 \cup G_3 = G$. Définissons $f : G \to G$ par

$$f(\beta) = \begin{cases} \gamma\beta, & \text{si } \beta \in G_1 \\ \beta^2, & \text{si } \beta \in G_2 \\ \alpha\beta, & \text{si } \beta \in G_3 \end{cases}$$

Nous choisissons un nombre aléatoire x_0 dans l'ensemble $\{1, \ldots, n\}$ et nous calculons $\beta_0 = \gamma^{x_0}$ dans le groupe. Après quoi, nous calculons par récurrence une suite (β_i) en posant

$$\beta_{i+1} = f(\beta_i).$$

Les éléments de cette suite peuvent s'ecrire sous la forme

$$\beta_i = \gamma^{x_i}\alpha^{y_i}, \quad i \geq 0$$

avec le nombre aléatoire initial, x_0.

$$x_{i+1} = \begin{cases} x_i + 1 \bmod n, & \text{si } \beta \in G_1 \\ 2x_i \bmod n, & \text{si } \beta \in G_2 \\ x_i, & \text{si } \beta \in G_3 \end{cases}$$

$$y_{i+1} = \begin{cases} y_i, & \text{si } \beta \in G_1 \\ 2y_i \bmod n, & \text{si } \beta \in G_2 \\ y_i + 1 \bmod n, & \text{si } \beta \in G_3 \end{cases}$$

Puisque nous travaillons dans un groupe fini, au bout d'un moment une correspondance est trouvée, deux éléments de la suite (β_i) sont égaux.

5.4. **Programmer la fonction f.**

Autrement dit, il existe $i \geq 0$ et $k \geq 1$ avec $\beta_{i+k} = \beta_i$, ce qui implique

$$\gamma^{x_i}\alpha^{y_i} = \gamma^{x_{i+k}}\alpha^{y_{i+k}}, \quad i \geq 0$$

et on en déduit que

$$\gamma^{x_i - x_{i+k}} = \alpha^{y_{i+k} - y_i} = \gamma^{x(y_{i+k} - y_i)}\alpha^{y_{i+k}}.$$

D'après le corollaire du cours sur l'ordre d'élément, le logaritbme discret x de α dans la base γ satisfait

$$x_i - x_{i+k} \equiv x(y_{i+k} - y_i) \bmod n$$

Résolvons cette congruence. La solution est unique mod$n$ si $y_{i+k} - y_i$ est inversible mod$n$. Si la solution n'est pas unique, le logarithme discret peut etre trouvé en testant les différentes possibilités mod$n$. Quand il existe trop de possibilites, l'algorithme est relancé avec un autre x_0 initial.

Au depart, (β_1, x_1, y_1) est stocké. Supposons plus généralement qu'un certain triplet (β_i, x_i, y_i) a été stocké. Alors on calcule les triplets (β_j, x_j, y_j) pour $j = i + 1, i + 2, \ldots$, et on les compare à $(\beta_i, x_i, y_i))$. On s'arrête quand une correspondance est trouvée, ou quand $j = 2i$. Si l'on n'a pas trouve de correspondance, quand on arrive à $j = 2i$, on efface le triplet (β_i, x_i, y_i) et on le remplace

par $(\beta_{2i}, x_{2i}, y_{2i})$. En faisant ainsi, on ne stocke que les triplets pour lesquels l'indice est puissance de 2. Avant de montrer que cette façon de procéder conduit toujours à une correspondance, nous allons donner un exemple.

5.5. **Algorithme ρ de Pollard (exemple).** Avec l'algorithme ρ de Pollard, résolvons le problème du logarithme discret

$$5^x \equiv 3 \bmod 2017.$$

Toutes les classes résiduelles sont représentées par leur plus petit représentants négatif. Nous posons

$$G_1 = \{1, \ldots, 672\},\ G_2 = \{673, \ldots, 1344\},\ G_3 = \{1345, \ldots, 2016\}$$

Comme valeur de démarrage, nous prenons $x_0 = 1023$.

Dans le tableau ci-dessous, on voit les triplets qui sont stockes momentanément pendant le déroulement de l'algorithme et le triplet final, avec la correspondance

$$\beta_{64} = \beta_{98}.$$

j	β_j	x_j	y_j
0	986	1023	0
1	2	30	0
2	10	31	0
4	250	33	0
8	1366	136	1
16	1490	277	8
32	613	447	155
64	1476	1766	1000
98	1476	966	1128

Nous avons $\beta_{64} = 5^{x_{64}} 3^{y_{64}} = 5^{1766} 3^{1000}$ et

$\beta_{96} = 5^{x_{96}} 3^{y_{96}} = 5^{966} 3^{1128}$.

Nous en déduisons que

$5^{1766} 3^{1000} \equiv 5^{966} 3^{1128} \bmod 2017$

ou encore que $5^{800} \equiv 3^{128} \bmod 2017$.

Illustration avec PARI/GP

```
> ga^(1766)*al^(1000)

%= Mod(1476, 2017)

> ga^(966)*al^(1128)

%= Mod(1476, 2017)
```

5.6. **Écrire un programme qui calcule β_j, x_j, y_j.**

5.7. **Pour calculer x, résoudre la congruence.**

$$128x \equiv 800 \bmod 2016.$$

Puisque $\mathrm{PGCD}(128, 2016) = 32$ divise 800, cette congruence possède une unique solution modulo $63 = 2106/32$. Nous résolvons la congruence $4z \equiv 25 \bmod 63$ qui a pour solution $z = 22$. Le logarithme discret est donc de la forme $x = 22 + k63$, avec $0 \leq k < 32$. L' essai des valeurs de k donne finalement le logarithme discret $x = 1030$, lorsque $k = 16$.

5.8. **Donner votre exemple de calcul du logarithme discret avec l'algorithme ρ de Pollard.** Vérifier avec znlog en PARI/GP.

Annexe A. Getting Started with PARI

(voir William Stein, page internet : `http://www.wstein.org/ant/ant.pdf`).

A.1. Documentation The documentation for PARI is available at

`http://pari.math.u-bordeaux.fr/`

Some PARI documentation :

(1) Installation Guide : Help for setting up PARI on a UNIX computer.

(2) Tutorial : 42-page tutorial that starts with `2 + 2`.

(3) User's Guide : 226-page reference manual ; describes every function

(4) Reference Card :

A.2. A Short Tour

```
> gp
Reading GPRC: /cygdrive/c/Program Files/PARI/.gprc ...Done.

         GP/PARI CALCULATOR Version 2.4.2 (development CHANGES-1.1971)
           i686 running cygwin (ix86/GMP-4.2.1 kernel) 32-bit version
 compiled: Dec 23 2007, gcc-3.4.4 (cygming special, gdc 0.12, using dmd 0.125)
                 (readline v5.2 enabled, extended help enabled)

                     Copyright (C) 2000-2006 The PARI Group

PARI/GP is free software, covered by the GNU General Public License, and
comes WITHOUT ANY WARRANTY WHATSOEVER.

Type ? for help, \q to quit.
Type ?12 for how to get moral (and possibly technical) support.

parisize = 4000000, primelimit = 500000
? \\ this is a comment
? x = 571438063;
? print(x)
571438063
? x^2+17
%2 = 326541459845191986
```

A.3. Help in PARI

```
? ?
Help topics:
  0: list of user-defined identifiers (variable, alias, function)
  1: Standard monadic or dyadic OPERATORS
  2: CONVERSIONS and similar elementary functions
  3: TRANSCENDENTAL functions
  4: NUMBER THEORETICAL functions
  5: Functions related to ELLIPTIC CURVES
  6: Functions related to general NUMBER FIELDS
  7: POLYNOMIALS and power series
  8: Vectors, matrices, LINEAR ALGEBRA and sets
  9: SUMS, products, integrals and similar functions
 10: GRAPHIC functions
 11: PROGRAMMING under GP
 12: The PARI community

Further help (list of relevant functions): ?n (1<=n<=11).
```

```
Also:
  ? functionname (short on-line help)
  ?\             (keyboard shortcuts)
  ?.             (member functions)
Extended help looks available:
  ??             (opens the full user's manual in a dvi previewer)
  ??  tutorial   (same with the GP tutorial)
  ??  refcard    (same with the GP reference card)

  ??  keyword    (long help text about "keyword" from the user's manual)
  ??? keyword    (a propos: list of related functions).
? ?4

addprimes     bestappr      bezout        bezoutres     bigomega
binomial      chinese       content       contfrac      contfracpnqn
core          coredisc      dirdiv        direuler      dirmul
divisors      eulerphi      factor        factorback    factorcantor
factorff      factorial     factorint     factormod     ffinit
fibonacci     gcd           hilbert       isfundamental isprime
ispseudoprime issquare      issquarefree  kronecker     lcm
moebius       nextprime     numdiv        omega         precprime
prime         primes        qfbclassno    qfbcompraw    qfbhclassno
qfbnucomp     qfbnupow      qfbpowraw     qfbprimeform  qfbred
quadclassunit quaddisc      quadgen       quadhilbert   quadpoly
quadray       quadregulator quadunit      removeprimes  sigma
sqrtint       znlog         znorder       znprimroot    znstar

? ?gcd
gcd(x,y,{flag=0}): greatest common divisor of x and y. flag is optional, and
can be 0: default, 1: use the modular gcd algorithm (x and y must be
polynomials), 2 use the subresultant algorithm (x and y must be polynomials).

? ??gcd
\\ if set up correctly, brings up the typeset subsection from the manual on gcd
```

A.4. **Etudier en PARI les fonctions.**

```
gp > ?ffinit
ffinit(p,n,{v=x}): monic irreducible polynomial of degree n over F_p[v]

gp > ?factorff
factorff(x,p,a): factorization of the polynomial x in the finite field
F_p[X]/a(X)F_p[X]


gp > ?polisirreducible
polisirreducible(pol): true(1) if pol is an irreducible non-constant
polynomial, false(0) if pol is reducible or constant.

gp > ?chinese
chinese(x,{y}): x,y being both intmods (or polmods) computes z in the same
residue classes as x and y.
```

```
gp > ?ffgen
ffgen(P,{v}): return the generator g=X mod P(X) of the finite field defined by
the polynomial P(X). If v is given, the variable name is used to display g,
else the variable of the polynomial P is used.


gp > ?fflog
fflog(x,g,{o}): return the discrete logarithm of the finite field element x in
base g. If present, o is the factorization of the multiplicative order of g.
If no o is given, assume that g is a primitive root.

gp > ?trace
trace(x): trace of x.

gp > ?min poly
minpoly(A,{v=x}): minimal polynomial of the matrix or polmod A.

gp > ?minpoly
minpoly(A,{v=x}): minimal polynomial of the matrix or polmod A.

gp > ?polcoeff
polcoeff(x,n,{v}): coefficient of degree n of x, or the n-th component for
vectors or matrices (for which it is simpler to use x[]). With respect to the
main variable if v is omitted, with respect to the variable v otherwise.

gp > ?fforder
fforder(x,{o}): multiplicative order of the finite field element x. Optional o
is assumed to be a multiple of the order of the element.

gp > ?ffprimroot
ffprimroot(x, {&o}): return a primitive root of the multiplicative group of
the definition field of the finite field element x (not necessarily the same
as the field generated by x). If present, o is set to the factorization of the
order of the primitive root (useful in fflog).


?bezout
bezout(x,y): gives a 3-dimensional row vector [u,v,d] such that d=gcd(x,y) and
u*x+v*y=d.

?chinese
chinese(x,{y}): x,y being both intmods (or polmods) computes z in the same
residue classes as x and y.

?eulerphi
eulerphi(x): Euler's totient function of x.

?isprime
isprime(x,{flag=0}): true(1) if x is a (proven) prime number, false(0) if not.
If flag is 0 or omitted, use a combination of algorithms. If flag is 1, the
primality is certified by the Pocklington-Lehmer Test. If flag is 2, the
primality is certified using the APRCL test.
```

```
?factor
factor(x,{lim}): factorization of x. lim is optional and can be set whenever x
is of (possibly recursive) rational type. If lim is set return partial
factorization, using primes up to lim (up to primelimit if lim=0)

?gcd
gcd(x,{y}): greatest common divisor of x and y.

?lcm
lcm(x,{y}): least common multiple of x and y, i.e. x*y / gcd(x,y)

?znorder
znorder(x,{o}): order of the integermod x in (Z/nZ)*. Optional o is assumed to
be a multiple of the order.

?random
random({N=2^31}): random integer between 0 and N-1.
```