Corrigé du DM 15

Méthode $p-1$ de factorisation de Pollard

Partie A - Arithmétique

1. Soit $a \in \mathbb{Z}$.

a) Comme $a \wedge p = 1$, le petit théorème de Fermat donne

$$a^{p-1} \equiv 1 \ [p]$$

et donc en élevant à la puissance $q-1$, il vient $a^{(p-1)(q-1)} \equiv 1 \ [p]$.
Comme on a aussi $a \wedge q = 1$, procédant de même, on obtient $a^{(p-1)(q-1)} \equiv 1 \ [q]$.
Enfin, d'après le corollaire du théorème de Gauss, $a^{(p-1)(q-1)} \equiv 1 \ [pq]$.
Ainsi, $\boxed{\text{si } a \text{ est premier avec } p \text{ et } q, \text{ alors } a^{(p-1)(q-1)} \equiv 1 \ [pq].}$

b) Soit $k \in \mathbb{Z}$. Procédons par disjonction de cas :

- Si $a \wedge n = 1$ alors en élevant à la puissance k le résultat précédent on obtient :

$$a^{k(p-1)(q-1)} \equiv 1 \ [n] \text{ et donc } a^{k(p-1)(q-1)+1} \equiv a \ [n]$$

- Si $q \mid a$ et $a \wedge p = 1$, alors il existe $u \in \mathbb{Z}$ tel que $a = uq$ et $a^{(p-1)(q-1)} \equiv 1 \ [p]$

$$\begin{aligned} a^{(p-1)(q-1)} \equiv 1 \ [p] \quad &\Rightarrow \quad qa^{(p-1)(q-1)} \equiv q \ [pq] \\ &\Rightarrow \quad uqa^{(p-1)(q-1)} \equiv uq \ [pq] \\ &\Rightarrow \quad a^{k(p-1)(q-1)+1} \equiv a \ [n] \end{aligned}$$

- Si $p \mid a$ et $a \wedge q = 1$, alors on trouve de même que $a^{k(p-1)(q-1)+1} \equiv a \ [n]$.
- Si $n \mid a$, alors $a^{k(p-1)(q-1)+1} \equiv 0 \equiv a \ [n]$.

Ainsi, $\boxed{\text{pour tout } a \in \mathbb{Z} \text{ et } k \in \mathbb{N}, \ a^{k(p-1)(q-1)+1} \equiv a \ [n].}$

c) Il existe $k \in \mathbb{Z}$ tel que $de = 1 + k(p-1)(q-1)$.
Ainsi, d'après 1b), $\boxed{\text{pour tout } x \in [\![0, n-1]\!], \text{ alors } x^{de} \equiv x \ [n].}$

2. L'ordre de grandeur de m^k est $\left(2^{1024}\right)^{2^{1024}} = 2^{2^{10}2^{1024}}$ ce qui fait 2^{1034} chiffres dans son écriture binaire. Or $10^{80} < 2^{1034}$, donc les atomes de l'univers connus ne sont pas suffisant pour représenter m^d et donc aussi x^d dès lors que n est de l'ordre 2^{1024}.

3. a) Suivi des variables lors de l'exécution de `algo(-2,13)` :

`r`	`m`	`p`	`p%2`
1	-2	13	1
-2	$(-2)^2 = 4$	6	0
-2	$2^4 = 16$	3	1
$(-2)^5 = -32$	$2^8 = 256$	1	1
$\boxed{(-2)^{13} = -8192}$	2^{16}	0	0

b) L'expression retournée par `algo` est x^d. Cet algorithme s'appelle l'exponentiation rapide.

c) Une version récursive est :

```
def algo_R(x,d):
 if n==0:
  return 1
 else:
  if n%2==0:
   return algo_R(x**2,d//2)
  else:
   return x*algo_R(x**2,d//2)
```

Appels			
x	d	d%2	Sortie
2	13	1	$\boxed{2^{13}}$
2^2	6	0	2^{12}
2^4	3	1	2^{12}
2^8	1	1	2^8
2^{16}	0		1

4. Fonction d'**exponentiation rapide** intégrant une réduction modulaire pour le calcul de

$$y \equiv x^d \ [n]$$

afin de manipuler seulement des entiers inférieurs à n^2 :

```
def alog_R(x,d,n):
    r=1
    while d!=0:
        if d%2==1:
            r=(r*x)%n
        x,d=(x*x)%n,d//2
    return(r)
```

5. Soient $(p_i)_{i\in I} \in \mathbb{P}^I$ et $(\alpha_i)_{i\in I} \in \mathbb{N}^I$ telles que la factorisation première de k soit

$$k = \prod_{i\in I} p_i^{\alpha_i}$$

Soit $i \in I$, alors $\alpha_i = v_{p_i}(k) \leq \alpha$ puisque $\alpha = \max\{\alpha_j;\ j \in I\}$.
est le maximum des valuations.
De plus, $p_i \leq b$ donc $\alpha_i \leq \dfrac{\alpha \times b}{p_i}$: $(\alpha \times b)!$ possède au moins α_i facteurs qui sont multiples de p_i.
Ainsi, $p_i^{\alpha_i} \mid (\alpha \times b)!$.
Les $\left(p_i^{\alpha_i}\right)_{i\in I}$ sont premiers entre eux deux à deux et divisent $(\alpha \times b)!$; d'après le corollaire du théorème de Gauss, leur produit divise aussi $(\alpha \times b)!$.
Ainsi, $\boxed{k \text{ divise divise } (\alpha \times b)!.}$

6. a) Comme $a \equiv 3^{B!}\ [n]$ alors $n \mid a - 3^{B!}$. Or $p \mid n$, donc par transitivité de la relation de divisibilité, $p \mid a - 3^{B!}$. Ainsi, $\boxed{a \equiv 3^{B!}\ [p].}$

b) Comme p est un grand nombre premier, 3 et p sont premier entre eux.
Le petit théorème de Fermat donne que $\boxed{3^{p-1} \equiv 1\ [p].}$
Or $p-1$ divise $B!$; notons v le quotient de $B!$ par $p-1$.
En élevant à la puissance v la relation précédente, il vient $3^{B!} \equiv 1\ [p]$ c'est-à-dire $\boxed{a \equiv 1\ [p].}$

c) Par définition de la congruence, il vient que $\boxed{p \text{ est un diviseur de } a-1.}$

d) Comme p est un diviseur commun à $a-1$ et n ; ainsi, $\boxed{p \text{ divise } (a-1) \wedge n.}$

Partie B - Chiffrement RSA

7. La fonction de chiffrement :

```
1  def chiffrer_RSA(f1,f3,d,n):
2      f2=f1[:-4]+'N'+f1[-4:]
3      numerise(f1,f2,n)
4      f=open(f2,mode='rt',encoding='utf8')
5      L=f.readlines()
6      f.close()
7      f=open(f3,mode='wt',encoding='utf8')
8      for e in L:
9          f.write(str(ERM(int(e.strip()),d,n))+'\n')
10     f.close()
```

L'introduction du fichier `f2`, à la ligne 2, se fait en rajoutant un `N` à la fin du nom du fichier `f1` tout en conservant l'extension d'origine qui est supposé être `.txt` (donc 4 caractères). On pouvait tout simplement définir `f2='temporaire.txt'`.
A la ligne 9, nous utilisant la méthode `.strip()` pour enlever le caractère de retour (`\n`) à la ligne qui suit l'entier de chaque ligne. On aurait pu mettre `e[:-1]` plutôt que `e.strip()`.

8. La fonction de déchiffrement :

```
def dechiffrer_RSA(f1,f3,e,n):
    f2=f1[:-4]+'D'+f1[-4:]
    f=open(f1,mode='rt',encoding='utf8')
    L=f.readlines()
    f.close()
    f=open(f2,mode='wt',encoding='utf8')
    for g in L:
        f.write(str(ERM(int(g.strip()),e,n))+'\n')
    f.close()
    denumerise(f2,f3)
```

9. a) Algorithme simple pour factoriser :

```
def facto(n):
    for i in range(2,int(n**0.5)):
        if n%i==0:
            return i,n//i
    return "n est Premier"
```

Cela donne $p = 135979$ et $q = 115979$.

b) La relation de Bezout obtenue est : $5 \times 3154091297 - (p-1)(q-1) = 1$.
L'inverse de d modulo $(p-1)(q-1)$ est $e = 3154091297$.

c) L'instruction suivant donne le message originel : le poême donnant les première décimale de π :

```
>>> dechiffrer_RSA('secret1.txt','clair1.txt',e,n)
```

Partie C - Cryptanalyse par la méthode $p-1$ de Pollard

10. La fonction `Pollard_p_1(n,B=10000)` qui recherche un facteur premier p de n dans le cas où $p-1$ est friable :

```
def pollard_p_1(n,B=10000):
    k,a,d=1,3,1 #g=3
    while d==1 and k<=B:
        k,a=k+1,ERM(a,k,n)
        d=pgcd(a-1,n)
    return(d)
```

11. L'identité de Bezout pour le couple $\big(d,(p-1)(q-1)\big)$ donne l'existence de $u,v\in\mathbb{Z}$ tel que

$$du+(p-1)(q-1)v=1$$

Ainsi $e=u$ convient ou encore $\boxed{e\equiv u\ [(p-1)(q-1)].}$

12. Recherche de la clé de déchiffrement associée à $(d=5,n=2021)$:

```
>>> pollard_p_1(2021)
43
>>> p,q=43,2021//43
>>> # 2021 = 43 x 47 produit de deux nombres premiers !
>>> Bezout(5,(p-1)*(q-1))
(773, -2, 1)
```

Ainsi, $\boxed{\text{la clé de déchiffrement est } (e=773,n=2021).}$

Vérification sur un nombre quelconque de $[\![0,2020]\!]$: $12^{de}\overset{?}{\equiv}12\ [2021]$

```
>>> ERM(12,5,2021)
249
>>> ERM(249,773,2021)
12
```

13. Décryptage du fichier `secret.txt` dont la clé de chiffrement est donné :

```
>>> p=pollard_p_1(n)    # augmenter B au besoin ...
>>> q= n//p
>>> e=Bezout(5,(p-1)*(q-1))[0] % ((p-1)*(q-1))
>>> dechiffrer_RSA('secret2.txt','clair2.txt',e,n)
```