Principes et Algorithmes de Cryptographie

TP 4 : RSA

Objectifs : Travailler autour de RSA.

Matériel requis :
- le logiciel MAPLE (ou tout autre logiciel de calcul en précision illimitée sur les entiers, comme la commande `bc` de Linux par exemple)

Quelques mots sur MAPLE : Vous pouvez utiliser MAPLE en ligne de commande dans un shell, ou bien avec son interface graphique. Dans le premier cas vous tapez la commande maple

```
[wegrzyno@lxo1:TP-RSA] maple
    |\^/|     Maple 11 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2007
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
>
```

Pour quitter, il suffit de taper la commande quit;.

Si vous préférez les interfaces graphiques, tapez la commande xmaple ou mieux si vous utilisez la version 11 de MAPLE sur des machines un peu limitées la commande maple −cw.

Préambule : Dans tout le TP, n désigne la partie nommée modulus d'une clé RSA. C'est le produit de deux nombres premiers distincts p et q (gardés secrets), tandis que n est rendu public. Les exposants de chiffrement (public) et déchiffrement (privé) sont notés e et d.

1 Familiarisation avec RSA avec MAPLE

1.1 Génération d'une paire de clés

1.1.1 Les nombres premiers

La commande `isprime` permet de déterminer si un nombre est premier ou non.

```
isprime(7);
                    true
isprime(8);
                    false
```

La commande `isprime` fonctionne aussi pour de très grands entiers.

```
isprime(578343762594484036292010369409436606060268690301866988470371);
                    true
isprime(245130742330997502479586474263802430118200560196663515061697);
                    false
```

La commande `nextprime` donne le premier nombre premier qui suit l'entier passé en paramètre.

```
nextprime(20);
                    23
nextprime(23);
                    29
```

1.1.2 Nombres premiers d'une taille donnée

Désignons par t la taille en bits de la clé RSA que l'on souhaite. Il nous faut donc trouver deux nombres premiers de taille $t/2$. Une façon d'atteindre cet objectif consiste à choisir deux nombres de $t/2$ bits au hasard et de trouver les nombres premiers qui les suivent.

Voici un exemple pour une taille de 30 bits.

```
# initialisation du generateur d'alea (a faire une seule fois en debut de session)
randomize():

# definition de la taille souhaitee du modulus (c'est un nombre pair)
t := 30:

# generation de deux entiers au hasard de t/2 bits
x := rand(2^(t/2-1)..2^(t/2))();
                    x := 28580
y := rand(2^(t/2-1)..2^(t/2))();
                    y := 21672

# calcul des nombres premiers qui suivent immediatement x et y
p := nextprime(x);
                    p := 28591
q := nextprime(y);
                    q := 21673
```

Il ne reste plus qu'à multiplier les deux nombres premiers p et q pour obtenir le modulus public de la clé RSA.

```
n := p*q;
                    n := 619652743
```

On peut vérifier que ce nombre s'écrit bien sur 30 bits

```
# ecriture binaire de n
convert(n,base,2);
   [1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0,
```

(MAPLE donne l'écriture binaire en commençant par le bit de poids faible.)

On peut aussi calculer la taille de n en base 2

```
floor(log[2](n))+1;
                    30
```

1.1.3 Exposants publics et privés

Les deux exposants public (e) et privé (d) doivent être tels que $ed \equiv 1 \pmod{\varphi(n)}$, où $\varphi(n) = (p-1)(q-1)$.

Dans notre exemple, $\varphi(n) = 619602480$.

```
phi := (p-1)*(q-1);
                    phi := 619602480
```

On peut choisir $e = 65537$ comme exposant public puisque e est premier avec $\varphi(n)$

```
e := 65537:
igcd(e,phi);
                    1
```

et l'inverse de e modulo φ se calcule par

```
d := 1/e mod phi;
                    d := 130581953
```

On dispose ainsi d'une paire de clés

– **publique** $n = 619652743, e = 65537$,
– et **privée** $d = 130581953$.

1.2 Chiffrement et déchiffrement

Les messages que l'on peut chiffrer sont les entiers compris entre 0 (inclus) et n (exclu).
Le chiffrement d'un message m s'obtient par le calcul de

$$c = m^e \pmod{n}.$$

Pour le message $m = 1234$, on obtient

```
m := 1234:
c := m&^e mod n;
                  c := 227889921
```

Le déchiffremenr d'un message chiffré c s'obtient par le calcul de

$$c^d \pmod{n}.$$

```
c&^d mod n;
                  1234
```

NB dans le calcul du chiffrement et du déchifffrment, il faut utiliser l'opérateur inerte &^ d'exponentiation, et non l'opétateur ^. En effet, l'usage de l'opérateur ^ déclenche le calcul de la puissance dans les $\mathbb{N}$, ce qui donne des nombres d'une taille inenvisageable en mémoire.

1.3 Sécurité

La sécurité de RSA repose sur la difficulté de factoriser le modulus de la clé publique.
La commande `ifactor` de MAPLE factorise les entiers qu'on lui passe en paramètre. Par exemple, la commande

```
ifactor(252);
                   2      2
                (2)    (3)    (7)
```

donne la factorisation $252 = 2^2 \times 3^2 \times 7$.
Question 1. Générez au hasard des modulus de clé RSA de taille de plus en plus grande, et constatez l'augmentation du temps de calcul de la factorisation de ce modulus.
Sur la machine que vous utilisez, à partir de quelle taille du modulus, la factorisation demande-t-elle un temps dépassant l'ordre de la minute ?

2 Utilisation de RSA

On utilise RSA essentiellement pour communiquer de manière confidentielle une clé d'un système à clé secrète.
Le fichier `cryptogram13` a été chiffré en utilisant l'AES en mode CBC avec un vecteur d'initialisation nul et une clé de 128 bits. Cette clé K a été elle-même chiffrée avec la clé publique RSA

$$\begin{aligned} n &= 484001516976824291824081505569967425918027658822251613166283 7 \\ e &= 65537, \end{aligned}$$

et on obtient

$$K^e \pmod{n} = 233627333367588510154859814969759518085615053960877783737 0662.$$

Question 2. Le modulus n est un produit de deux nombres premiers assez proches l'un de l'autre. Il est possible de le factoriser par la méthode de Fermat. Faîtes-le.

Question 3. Déduisez-en la clé privée d, puis retrouvez la clé secrète K.

Question 4. Une fois la clé K connue, utilisez `openssl` (cf `../TP2/`) pour déchiffrer le cryptogramme. Que représente l'image obtenue ?

3 Factoriser le modulus

Il existe un algorithme probabiliste qui permet (avec une probabilité non négligeable) de factoriser le modulus n en temps polynomial, lorsqu'on connaît les exposants de chiffrement e et de déchiffrement d.

Cette méthode est décrite ci–dessous en MAPLE (les variables `n`, `e` et `d` désignent ... euh ... n, e et d, et la fonction `igcd` le pgcd).

```
#########################################
# Factorisation de n connaissant e et d
#########################################
# la cle publique
n := 7507749913: e := 3217382455: d := 7:

# initialisation du generateur de nombres pseudo-aleatoires de MAPLE
randomize():

# recherche de l'entier impair u tq ed-1 = u*2^l
u := e*d-1:
while irem(u,2,'quot') = 0 do u := quot od:

# recherche d'une racine carree modulo n de 1
# de la forme a^(u*2^m) mod n
# a entier aleatoire dans Z_n
a := rand(2..n-1)();
                    a := 257126156156
igcd(a,n);

                    1
b := a&^u mod n:
b2 := b*b mod n:
while b2<>1 do
  b := b2:
  b2 := b*b mod n:
od:

p1 := igcd(b-1,n);
                    p1 := 509449
q1 := igcd(b+1,n);
                    q1 := 14737
n-p1*q1;
                    0
# BINGO si 1< p1 <n
# sinon on recommence avec une autre valeur de a
```

Question 5. Testez cet algorithme avec une clé RSA de 300 bits.

4 L'attaque de Wiener

M. Wiener a publié en 1990 (*Cryptanalysis of Short RSA Secret Exponents*, IEEE Transaction on Information Theory, vol 36, n° 3, mai 1990) une attaque qui permet de retrouver la clé privée d en connaissant uniquement la clé publique e et n lorsque cette clé privée d est inférieure à la racine quatrième du modulus n.

Sa technique s'appuie sur le développement en fraction continue de e/n. Parmi les réduites obtenues, l'une a pour dénominateur l'exposant privé d.

Voici un exemple en MAPLE

```
#######################################
# Attaque de Wiener
```

```
#############################################
# la cle publique
n := 7507749913: e := 3217382455:
# calcul des reduites de la fraction continue associee a e/n
convert(e/n,confrac,'reduites'):
# Voici les reduites
reduites;
                      2045   6138    45011    51149    3011653   3062802   6074455
9137257    24348969   57835195
[0, 1/2, 2/5, 3/7, ----, -----, ------, ------, -------, -------, --------, --------, --------, ---------,
                      4772   14323   105033   119356   7027681   7147037   14174718
21321755   56818228   134958211

     82184164    222203523   748794733    3217382455
     ---------,  ---------,  ----------,  ----------]
     191776439   518511089   1747309706   7507749913

# Recherche de la cle privee parmi les denominateurs des reduites
M := 12345:
C := M&^e mod n:
i := 1:
while C&^denom(reduites[i]) mod n <> M do i := i+1: od:
i,denom(reduites[i]);
                       4, 7
# BINGO !! on a trouve la cle privee
```

Question 6. Pourquoi pourrait-il être intéressant (pour le titulaire de la clé) d'avoir un petit exposant de déchiffrement ?

Question 7. Les cryptogrammes suivants ont été produits avec des clés RSA dont l'exposant de déchiffrement satisfait l'hypothèse de l'attaque de Wiener.

Chiffrés	**Modulus**	**Exposants de chiffrement**
78382945569884104608019116184[illegible] 297766231340655925365925283689[illegible] 1066284783654038457202403830[illegible] 3024360414206299770836089618910[illegible] 98382003545481258160965704137[illegible] 9318236280200904889067968609[illegible] 24204603961371627696582144939[illegible] 7381339578346	[illegible]82219533855601574[illegible] [illegible]64746192242312064[illegible] [illegible]7218652403849992[illegible] [illegible]24553481000061931[illegible] [illegible]58891107478865521[illegible] [illegible]19736813805495036[illegible] [illegible]408585387506517988[illegible] 28115165005573	[illegible]737836471611843302981783939067 [illegible]080276281410590721481447350044 [illegible]387521607321728974648113803980 [illegible]78376077107638769067644093 0247 [illegible]547005213922476618226166883346 [illegible]840327749804317926061354346936 [illegible]252161732203490706405093369143 01659248439067
833968778853464960057186136770[illegible] 2434543716108765428806198348[illegible] 10216730988884387335480467013[illegible] 319552731167359479461685468623[illegible] 60367609543709872716506886284[illegible] 05668052109927648879040221776[illegible] 21202545221857824185249404393[illegible] 6198338773038898875844973794[illegible]	[illegible]401917286856778611[illegible] [illegible]855248435005934362[illegible] [illegible]035502820382765790[illegible] [illegible]97066108010611312[illegible] [illegible]33048521430113026[illegible] [illegible]75268963701061026[illegible] [illegible]0714711174160665[illegible] [illegible]766187899046644633929848695860[illegible]	[illegible]145671126231090336978691321 34 [illegible]345672027884829247809202221680 [illegible]612441506804926947592674547090 [illegible]539558423259767929049816585021 [illegible]718680145111100209503963121 79 [illegible]590525750668152181064899642491 [illegible]029476025997459567359191570907 5 [illegible]112014315113204177788897205463

Retrouvez le texte original en utilisant la procédure nombreEnTexte décrite dans l'annexe (cf 6).

5 Produire des clés avec trappe

Il est possible de s'appuyer sur l'attaque de Wiener pour produire des clés RSA avec trappe. Voici un générateur de clés RSA, proposé par C. Crépeau et A. Slakmon, qui permet à son auteur de retrouver la clé privée correspondant à une clé publique produite par ce générateur. Voici décrit ce générateur (tel qu'on le trouve sur leur site) :

```
Let M := a STRENGTH bit even constant fixed in the program.
REPEAT pick a random number P of STRENGTH/2 bits UNTIL it is a prime
REPEAT pick a random number Q of STRENGTH/2 bits UNTIL it is a prime
Let N:=P × Q, Φ:=(P − 1) × (Q − 1)
REPEAT
 REPEAT
  pick a random number D such that |D| < |N|/4
 UNTIL gcd(D,Φ)=1
 find E such that D × E = 1 (mod Φ).
 let E' := E + M
UNTIL gcd(E',Φ) = 1
find D' such that D' × E' = 1 (mod Φ).
Output Private Key := (D',P,Q), Public Key := (E',N).
```

où STRENGTH désigne le nombre de bits de la clé à produire, et la notation $|D|$ désigne le nombre de bits de l'entier D.

Question 8. Quel algorithme faut-il utiliser dans ce programme pour calculer E et D' ?

Question 9. La clé privée produite par ce générateur vérifie-t-elle les hypothèses de l'attaque de Wiener ?

Question 10. Indiquez comment l'auteur de ce programme peut retrouver la clé privée en connaissant la clé publique.

Question 11. Retrouvez la partie privée de la clé RSA produite par un tel générateur en connaissant la partie publique, ... et la trappe ! Puis déchiffrez le message.

Modulus	**Exposants de chiffrement**	**Trappes**	**Messages**
6100514307489148784996713[illegible]	[illegible]17688821	[illegible]252864258	[illegible]8710228565670
0077544818371775359938496[illegible]	[illegible]27315165	[illegible]27759615	[illegible]38576389131
0730583596493664202678439[illegible]	[illegible]25614705	[illegible]45043208	[illegible]35728808427
5521222854452994528174644[illegible]	[illegible]98046541	[illegible]94389505	[illegible]530185257461144891204645
6496411351126797287532700[illegible]	[illegible]01705279	146783033945851530	60971041768211343844490013562
5907190504194000270219989[illegible]	[illegible]26256016	557897302035461701	82817045692582459931330765560
2929412717540848904058698[illegible]	[illegible]22704686	914127878844307401	65901599665247475596473930372
77173949143553	33582284471411		33670817257764

6 Annexe

Les cryptogrammes des parties 4 et 5 sont obtenus par le codage de petits textes en français transformés en nombres entiers. L'alphabet utilisé pour les textes est constitué des 26 lettres (majuscules et non accentuées) de l'alphabet latin, ainsi que de l'espace. L'alphabet comporte ainsi 27 caractères.

Le codage utilisé pour transformer un texte en un nombre consiste à transformer chaque lettre en un nombre compris entre 10 et 36 et à concaténer les nombres entre eux. Avec ce codage, le texte `PAC` est transformé en le nombre 261113

Voici le code en MAPLE de procédures de codage et de décodage de textes sous forme de nombres entiers .

```
# l'alphabet utilisable
Alphabet:=" ABCDEFGHIJKLMNOPQRSTUVWXYZ":
# taille de l'alphabet
nb_lettres:=length(Alphabet):
# construction de la correspondance lettre<->code
# Dans ce codage chaque caractere est code par un nombre
# entier compris entre 10 (pour l'espace) et 36 (pour le z)
for i from 1 to nb_lettres do code[Alphabet[i]]:=10+i-1 od:
code["P"], code["A"], code["C"];
                       26, 11, 13
```

```
# procedure de transformation d'un texte en un nombre entier
texteEnNombre := proc(t)
local i;
  return convert([seq(code[t[i]]*10^(2*(length(t)-i)),
                 i=1..length(t))],`+`);
end proc:
# exemple avec le texte "PAC"
texteCode := texteEnNombre("PAC");
                     texteCode := 261113

# Pour decoder un code n, il suffit de considerer le
# caractere de position n-9 dans l'alphabet
Alphabet[26-9];
                     "P"
# procedure de transformation d'un nombre en texte
nombreEnTexte := proc(n)
local i,L;
  L := convert(n,base,100);  # conversion de l'entier n en base 100
  L := [seq(L[nops(L)-i+1],i=1..nops(L))]; # renversement de la liste
  return cat(seq(Alphabet[i-9],i=L));
end proc:
nombreEnTexte(texteCode);
                     "PAC"
```