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(57834376259448403629201036940943660606026869030186698847037 1);
                    true
isprime(24513074233099750247958647426380243011820056019666351506 1697);
                    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
l := convert(n,base,2);
   l := [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

nops(l) ;
                    30
```

(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
```

Question 1. Pourquoi l'exposant public e (ainsi que l'exposant privé d) ne peut-il pas être pair ?

On peut choisir $e = 65537$ comme exposant public puisque e est premier avec $\varphi(n)$ (avec d'autres valeurs de p et q il pourrait arriver que e ne soit pas premier avec $\varphi(n)$; dans ce cas, soit on change la valeur de$e$, soit on change les nombres premiers p et q)

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

et l'inverse de e modulo φ se calcule par

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

Il est possible de regrouper le calcul du pgcd et de l'inverse avec la commande igcdex qui applique l'algorithme d'Euclide étendu et calcule les coefficients de Bezout. Ces coefficients pouvant être négaitifs, il peut s'avérer nécessaire d'effectuer une réduction modulo $\varphi(n)$.

```
igcdex(e,phi,'d');
                          1
d := d mod phi;
                        d := 130581953
```

Notez les apostrophes autour du troisième paramètre de la commande igcdex.

On dispose ainsi d'une paire de clés
- **publique** $n = 619652743, e = 65537$,
- et **privée** $d = 130581953$.

Question 2. Dans la pratique, il est très fréquent de fixer la valeur de l'exposant public e à 65537. C'est le choix par défaut avec OpenSSL par exemple. Pourquoi ce choix ? Quel avantage présente-t-il ? Décrivez la procédure de génération d'une paire de clés RSA, si on impose $e = 65537$.

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 3. 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

Question 4. Récupérez le fichier `les_cryptogrammes4.zip` et décompressez-le.

On utilise RSA essentiellement pour communiquer de manière confidentielle une clé d'un système à clé secrète.

Le fichier `cryptogram16` 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

$$K^e \pmod n = 233627333367588510154859814969759518085615053960877783737066 2.$$

Question 5. 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 6. Déduisez-en la clé privée d, puis retrouvez la clé secrète K.

Question 7. Une fois la clé K connue, utilisez `openssl` (cf le 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 8. 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 9. Pourquoi pourrait-il être intéressant (pour le titulaire de la clé) d'avoir un petit exposant de déchiffrement ?

Question 10. Les cryptogrammes suivants ont été produits avec des clés RSA dont l'exposant de déchiffrement satisfait l'hypothèse de l'attaque de Wiener.

Chiffré	Modulus	Exposant de chiffrement
783829455698841046080191161 8	726076907963822195338556015 7	677159723405737836471611843 3
474336965523762977662313406 5	422697983935892589478301226 4	029817839390670196399192710 8
592536592528368972073992144 6	746192242312064761807469686 5	027628141059072148144735004 4
106628478365403845720240383 0	560327989977672186524038499 9	194135093898387521607321728 9
044504376648763024360414206 2	278415098285210206719578492 4	746481138039808478820690147 8
997708360896189100881098719 3	553481000061931627100261501 0	376077107638769067644093024 7
983820035454812581609657041 3	904204768191588911074788655 2	896667015586547005213922476 6
755061613485229318236280200 9	100484259456562186576156821 9	182261668833464366482138348 4
048890679686093695681420980 6	736813805495030606420410646 7	032774980431792606135434693 6
242046039613716276965821449 3	489805123379408585387506517 9	254039078032252161732203490 7
917660515828417381339578346	889076860401832811516500557 3	064050933691430165924843906 7

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

Question 12. La clé privée produite par ce générateur vérifie-t-elle les hypothèses de l'attaque de Wiener ?

Question 13. Indiquez comment l'auteur de ce programme peut retrouver la clé privée en connaissant la clé publique.

Question 14. 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	Exposant de chiffrement	Trappe	Message chiffré
152340917674053822933312	125603267856784279105792	186166570161033043410344	150566825798183151272193
481998971581584529236883	380706052139443130474321	415770536751930267094255	845925935647007323690405
325414640426105789290371	632965374679126523178492	468204654566224615606416	746131468305068567581232
333758134502765704181365	596119550217241540705526	501000464581582889085097	376685647560466212707710
460990236623310066275421	673608669228200186826162	059611704946390206585951	305344288524573658357964
508250006903145242509195	225482364635730386515403	127789811172455336262347	989530981388134753958177
070233606095080194282835	712625179764253741641405	614684258490475340792297	712687753429812102651545
412907028509891138620468	000054063831756916855459	861252695166802312474877	033012720613202387992863
878270276627672213131003	932644016615405233011448	110961633292161499439705	922621689569805194988948
270466128386612489928808	673498452095136507943274	000793854819238848003438	315688264114727380545939
837618541574658762281569	033162396876321069107400	801966378788611186180962	419216700437015595721697
053798763614541265975492	464473902591465701212738	381648173603420993167483	123087652799909165018013
749269724529905586066157	663287378935769704372947	173638773944972565402	968670985640212679646282
637862937225736787461034	900773307748415925238570		380125725245915672331529
483829174300674191279786	882144183613521397925073		652422569477076246185400
432438902182552555861193	332003303445429421113163		305993428630511999931698
556282634429744890479291	983486249912977343613940		959439424268386348292065
730190275667359117733408	729362568900254280288240		078272126520604434464823
299473164114354736692242	440773536719244796377086		348274024527373651474523
116917742111680682874704	794966563425342578682925		212852626121164388786911
836901243413246501134302	390875910222210811098223		635634690913295707760041
622581197048403230981457	339750838434773242738397		867693259712122132586387
697234654182126299402343	093826617993518096367769		743748904515420699443118
689511821192973424171138	729120207583507851040869		994175851667456276355321
339664886818605329938329	705847717198462630948617		365137907947989288635778
76155650441484201	56536881923889259		26864208570225975

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"
```