CRYPTANALYSE DE RSA

Abderrahmane Nitaj

CRYPTANALYSE DE RSA

Abderrahmane Nitaj

Laboratoire de Mathématiques Nicolas Oresme
Université de Caen, France

http://www.math.unicaen.fr/~nitaj

nitaj@math.unicaen.fr

Table des matières

Préface

Si vous enseignez à un homme, vous n'enseignez qu'à une personne. Si vous enseignez à une femme, vous enseignez à toute une famille.

إِذَا علَّمتَ رَجُلًا فَقَد علَّمتَ فَردًا، أَمَّا إِذَا علَّمتَ إِمرأَةً فَقَد علَّمتَ عَائِلةً.

La cryptographie moderne est basée sur les mathématiques pour sécuriser l'information. On distingue deux types de protocoles cryptographiques : la cryptographie à clé privée et la cryptographie à clé publique. La cryptographie à clé publique à été introduite par Whitfield Diffie et Martin Hellman en 1976, marquant ainsi la naissance de la cryptographie moderne. Le principe de la cryptographie à clé publique repose sur deux types de clés : une clé publique et une clé privée. Pour chiffrer un message, on utilise la clé publique de son destinataire. Alors, seul le destinataire peut déchiffrer le message reçu avec sa propre clé privée. En 1978, Ronald Rivest, Adi Shamir et Leonard Adleman ont proposé le premier cryptosystème à clé publique, appelé **RSA**. Ce cryptosystème est devenu le plus répandu dans le monde car il est facile à réaliser mais très difficile à casser. En effet, sa sécurité repose sur l'un des problèmes les plus difficiles en mathématiques : la factorisation des grand nombres.

Dans ce travail, nous introduisons les principes généraux du cryptosystème RSA ainsi que certaines attaques permettant de le casser, si les paramètres de sécurité sont mal choisis ou s'il vérifient des relations permettant à un attaquant d'en tirer profit. Dans le chapitre 1, nous donnons les principes généraux du cryptosystème RSA et nous présentons quelques attaques élémentaires permettant de le casser. Dans le chapitre 2, nous présentons une introduction à la théorie des fractions continues et leur utilisation pour attaquer le cryptosystème RSA dans certains cas. Dans le chapitre 3, nous présentons quelques aspects de la réduction des réseaux, plus précisément l'algorithme **LLL** et son utilisation pour attaquer le cryptosystème RSA grâce à la méthode de Coppersmith.

Dans ce travail, la plupart des résultats sont illustrés par des algorithmes programmés à l'aide des systèmes de calcul **Maple 12** et **Magama** dont un calculateur

en ligne est à l'adresse `http://magma.maths.usyd.edu.au/calc/`.

Chapitre 1

Introduction au cryptosystème RSA

Celui qui n'aime pas gravir les montagnes, vivra toute sa vie dans les trous.
Abou Al Qasim Achabi

وَ مَن لَا يُحِبُّ صُعُودَ الجِبَالِ يَعشُ أَبَدَ الدَّهرِ بَينَ الحُفَرِ.
ابُو القَاسِم الشَابِّي

1.1 Principe de RSA

Toutes les opération du cryptosystème RSA se passe dans un ensemble de nombre entiers. Soient p et q deux nombres premiers assez grands. On note $N = pq$. Le nombre N est appelé *module RSA*. Supposons que deux personnes A et B veulent communiquer de façon sûre en utilisant le cryptosystème RSA. Pour cela, ils doivent, chacun de son coté préparer un module RSA, deux clés e et d, exécuter une procédure de chiffrement et de signature et une procédure de déchiffrement et de vérification de la signature.

1.1.1 Le module RSA

Avant tout, pour utiliser le cryptosystème RSA, chacun des intervenants A et B doit fabriquer son propre module RSA. L'algorithme 1 peut être alors utilisé.

Algorithme 1 : Fabrication du module

Entrée : Une taille t pour le module du cryptosytème RSA.
Sortie : Un module RSA N de taille t.

1: Prendre un nombre premier aléatoire p dans l'intervalle $\left[2^{\frac{t}{2}}, 2^{\frac{t+1}{2}}\right]$.
2: Prendre un nombre premier aléatoire q dans l'intervalle $\left[2^{\frac{t}{2}}, 2^{\frac{t+1}{2}}\right]$.
3: **Si** $p = q$ **alors**
4: Aller à l'étape 2.
5: **Sinon**
6: $N = pq$.
7: **Fin Si**

Comme p et q sont dans l'intervalle $\left[2^{\frac{t}{2}}, 2^{\frac{t+1}{2}}\right]$, alors on a

$$2^t < pq < 2^{t+1},$$

ce qui montre que $N = pq$ est de taille t.

Maple 1.1.1.

Avec Maple 12, l'algorithme 1 peut être réalisé comme ceci.

Programme	Commentaires
`t:=1024:`	`<----- la taille du module RSA`
`x1:=round(2.0^(t/2)):`	`<----- la borne inférieur 2^(t/2)`
`x2:=round(2.0^((t+1)/2)):`	`<----- la borne supérieur 2^((t+1)/2)`
`m1:=(rand(x1 .. x2))():`	`<----- une valeur aléatoire`
`p:=nextprime(m1);`	`<----- le nombre premier p`
`m2:=rand(x1 .. x2)():`	`<----- une valeur aléatoire`
`q:=nextprime(m2);`	`<----- le nombre premier q`
`N:=p*q`	`<----- le module RSA`

Magma 1.1.2.

Avec Magma, l'algorithme 1 peut être réalisé comme ceci.

Programme	Commentaire
t:=1024;	<----- la taille du module RSA
x1:=Round(2.0^(t/2));	<----- la borne inférieur 2^(t/2)
x2:=Round(2.0^((t+1)/2));	<----- la borne supérieur 2^((t+1)/2)
m1:=Random(x1,x2);	<----- une valeur aléatoire
p:=NextPrime(m1);	<----- le nombre premier p
m2:=Random(x1,x2);	<----- une valeur aléatoire
q:=NextPrime(m2);	<----- le nombre premier q
N:=p*q;	<----- le module RSA
N;	<----- affichage du module RSA

Dans Magma, la fonction **RSAModulus(t,e)** permet de créer un module RSA N à t-bits tel que $\text{pgcd}(e, \phi(N)) = 1$. Voici un exemple et sa vérification.

Programme	Commentaire
t:=100;	<--- Taille du module
e:=3;	<--- L'exposant publique
N:=RSAModulus(t,e);	<--- Création du module
N;	<--- Affichage de N
f:=Factorization(N);	<--- Sa factorization
f;	<--- Affichage de la factorisation
Log(N)/Log(2);	<--- Taille du module
p:=f[1,1];	<--- Premier nombre premier
q:=f[2,1];	<--- Deuxième nombre premier
p;	<--- Affichage de p
q;	<--- Affichage de p

1.1.2 Les clés publiques et privées

Après avoir fabriqué un module RSA, chacun des intervenants doit préparer une clé secrète d et une clé publique e :

Dans certains cas, on peut prendre pour la clé publique des valeurs spécifiques, par exemple $e = 3$ ou $e = 2^{16} + 1$. Dans ce cas, les étapes 2 à 5 de l'algorithme 2 ne sont pas exécutées.

La fonction ϕ joue un rôle central dans le cryptosystème RSA et s'appelle la fonction d'Euler.

Définition 1.1.3. *Soit n un nombre entier. La fonction indicatrice d'Euler est*

$$\phi(n) = \#\left\{a \mid 0 \leq a \leq n-1,\ pgcd(a,n) = 1\right\}.$$

Algorithme 2 : Fabrication des clés

Entrée : Deux nombres premiers p et q.
Sortie : Une clé privée d et une clé publique e.

1: Calculer $\phi(N) = (p-1)(q-1)$.
2: Prendre un nombre aléatoire e dans l'intervalle $[1, \phi(N)]$.
3: **Si** $\text{pgcd}(e, \phi(N)) \neq 1$ **alors**
4: Aller à l'étape 2.
5: **Sinon**
6: Calculer $d \equiv e^{-1} \pmod{\phi(N)}$.
7: **Fin Si**

Cette fonction est définie pour tout entier $n \geq 2$. Si la décomposition en facteurs premiers est

$$n = \prod_{i=1}^{s} p_i^{x_i},$$

alors on a :

$$\phi(n) = \prod_{i=1}^{s} p_i^{x_i - 1}(p_i - 1).$$

Maple 1.1.4.
Dans Maple 12, la fonction ϕ est tout simplement **phi**. Attention, pour calculer $\phi(N)$, Maple est obligé de factoriser N, ce qui peut être très difficile si N est du type module de RSA.

Programme	Commentaires
`with(numtheory):`	`<--- Package numtheory "Théorie des Nombres"`
`N:=5*8*61:`	`<--- un exemple simple`
`phi(N);`	`<--- Calcul de phi(n)`

La réponse est immédiate : 960.

Magma 1.1.5.
Dans Magma, la fonction ϕ est tout simplement **EulerPhi**. Attention, pour calculer $\phi(N)$, Magma aussi est obligé de factoriser N, ce qui peut être très difficile si N est du type module de RSA.

Programme	Commentaires
`N:=5*8*61:`	`<--- un exemple simple`
`EulerPhi(N);`	`<--- Calcul de phi(N)`

Maple 1.1.6.

Avec Maple 12, l'algorithme 2 peut être écrit comme ceci où on suppose que la procédure 1.1.1 a été exécutée.

Programme	Commentaires
phi:=(p-1)*(q-1):	<--- L'indicateur d'Euler
e:=rand(3..phi/2)():	<--- On choisit un exposant e aléatoire
while(gcd(e,phi) <> 1) do	<--- il faut que pgcd(e,phi)=1
e:=rand(3..phi)();	<--- On reprend un autre exposant e
od;	
d := modp(1/e, phi);	<--- d est l'inverse de d modulo phi

Magma 1.1.7.

Avec Magma, l'algorithme 2 peut être écrit comme ceci.

Programme	Programme
t:=102;	<-- Taille du module
x1:=Round(2.0^(t/2));	<-- Valeur aléatoire
x2:=Round(2.0^((t+1)/2));	<-- Valeur aléatoire
m1:=Random(x1,x2);	<-- Valeur aléatoire
p:=NextPrime(m1);	<-- Premier nombre premier
m2:=Random(x1,x2);	<-- Valeur aléatoire
q:=NextPrime(m2);	<-- deuxième nombre premier
N:=p*q;	<-- Module RSA
N;	<-- Valeur de N
phi:=(p-1)*(q-1);	<-- phi
e:=Random(3,Round(phi/2));	<-- Un exposant public aléatoire
while(GCD(e,phi) gt 1) do	<-- procédure de recherche
e:=Random(3,Round(phi/2));	<-- GCD(e,phi)=1
end while;	<--
d:= InverseMod(e,phi);	<-- Inverse de e modulo phi
d;	<-- Valeur de l'exposant privé

1.1.3 Envoi d'un message

Supposons maintenant que les intervenants A et B possèdent chacun son module RSA et ses clés, N_A, e_A, d_A pour A et N_B, e_B, d_B pour B.

Si A veut envoyer un message à B, il peut procéder comme dans l'algorithme 3.

Maple 1.1.8.

Algorithme 3 : Chiffrement d'un message

Entrée : Un message clair et la clé publique (N_B, e_B).
Sortie : Un message chiffré C.
1: Transformer le message en un nombre entier M de l'intervalle $[2, N_B]$.
2: Calculer $C \equiv M^{e_B} \pmod{N_B}$.
3: Envoyer le message C.

Avec Maple 12, l'algorithme 3 peut être programmé de la façon suivante.

Programme	Commentaires
`M:=12345:`	`<--- un exemple simple de message`
`C:=modp(M&^e,N):`	`<--- &^ permet de calculer  M^e modulo N`

Magma 1.1.9.
Avec Magma, l'algorithme 3 peut être programmé de la façon suivante.

Programme	Commentaires
`M:=12345;`	`<--- un exemple simple de message`
`C:=Modexp(M,e,N);`	`<--- Fonction pour calculer  M^e modulo N`

1.1.4 Déchiffrement d'un message

Supposons enfin que B a reçu un message chiffré C de la part de A. Alors B peut le déchiffrer en utilisant sa clé secrète d_B comme dans l'algorithme 4.

Algorithme 4 : Déchiffrement d'un message

Entrée : Un message chiffré C et la clé privée (N_B, d_B).
Sortie : Un message clair M.
1: Calculer $M \equiv C^{d_B} \pmod{N_B}$.
2: Transformer le nombre M en un message clair.

Maple 1.1.10.
Avec Maple 12, l'algorithme 4 qui permet de déchiffrer un message C peut être le suivant.

Programme	Commentaires
`M2:=modp(C&^d,N):`	`<--- Calcul de C^d modulo N`
`M-M2`	`<--- Permet de vérifier que M2=M`

Magma 1.1.11.
Avec Magma, l'algorithme 4 qui permet de déchiffrer un message C peut être le suivant.

Programme	Commentaires
`M2:=Modexp(C,d,N);`	`<--- Calcul de C^d modulo N`
`M-M2;`	`<--- Permet de vérifier que M2=M`

1.1.5 Signature d'un message

Supposons que A veut envoyer un message M à B. Comment B peut-il être sûr que le message reçu a bien été envoyé par A. Pour résoudre ce problème, RSA peut être utilisé aussi bien pour transmettre un message que pour le signer.

Pour cela, A et B doivent avoir chacun ses clés publiques et privées, N_A, e_A, d_A pour A et N_B, e_B, d_B pour B.

Tout d'abord, A utilise la clé publique (N_B, e_B) de B pour transmettre le message C et produire une signature S du message à l'aide de sa propre clé privée (N_A, e_A). En effet, A doit produire et transmettre C et S comme ceci (voir algorithme 5).

$$C \equiv M^{e_B} \pmod{N_B}, \quad S = C^{d_A} \pmod{N_A}.$$

En possession du message chiffré C et de la signature S, B peut alors vérifier si c'est bien A qui a transmit ce message en utilisant la clé publique (N_A, e_A) de A, du fait de la relation (voir algorithme 6).

$$S^{e_A} \equiv \left(C^{d_A}\right)^{e_A} \equiv C^{d_A e_A} \equiv C \pmod{N_A}.$$

En effet, seul A peut produire la signature S et donc B peut aisément vérifier si la signature S donne une deuxième fois le message chiffré C.

Algorithme 5 : Signature d'un message

Entrée : Un message clair M, deux clés publiques (N_B, e_B) et (N_A, e_A) et une clé privée (N_A, d_A).

Sortie : Un message chiffré C et sa signature S.

1: A transforme le message en un nombre entier M de l'intervalle $[2, N_B]$.
2: A calcule $C \equiv M^{e_B} \pmod{N_B}$.
3: A calcule $S \equiv C^{d_A} \pmod{N_A}$.
4: A transmet le message C et la signature S.

Maple 1.1.12.
Avec Maple 12, l'algorithme 5 qui permet de chiffrer et signer un message C peut être le suivant.

Programme	Commentaires
NA:=1577801413:	<--- Module RSA de A
eA:=147944791:	<--- Clé publique de A
dA:=295049071:	<--- Clé privée de A
NB:=1303570001:	<--- Module RSA de B
eB:=902841103:	<--- Clé publique de B
dB:=1208086831:	<--- Clé privée de B
M:=12345:	<--- Un exemple simple de message clair
C:=modp(M&^eB,NB):	<--- Le message chiffré
S:=modp(C&^dA,NA):	<--- La signature du message

Magma 1.1.13.
Avec Magma, l'algorithme 5 qui permet de chiffrer et signer un message C peut être le suivant.

Programme	Commentaires
NA:=1577801413;	<--- Module RSA de A
eA:=147944791;	<--- Clé publique de A
dA:=295049071;	<--- Clé privée de A
NB:=1303570001;	<--- Module RSA de B
eB:=902841103;	<--- Clé publique de B
dB:=1208086831;	<--- Clé privée de B
M:=12345;	<--- Un exemple simple de message clair
C:=Modexp(M,eB,NB);	<--- Le message chiffré
S:=Modexp(C,dA,NA);	<--- La signature du message

L'algorithme de la vérification de la signature est alors l'algorithme 6.

Algorithme 6 : Vérification d'une signature

Entrée : Un message chiffré C, une signature S, la clé publique (N_A, e_A).
Sortie : Vérification d'une signature.
1: B calcule $C' \equiv S^{e_A} \pmod{N_A}$.
2: Si $C' = C$ la signature est authentifiée.

Maple 1.1.14.

Avec maple 12, on peut alors vérifier la signature avec un programme simple.

Programme	Commentaires
`C2:=modp(S&^eA,NA):`	`<--- calcul de S^eA modulo NA`
`dif:=C2-C:`	`<--- Difference des messages`
`if dif=0 then`	`<--- vérification de la signature`
`  print('Signature valide')`	
`else`	
`  print('Signature non valide')`	
`end if`	

Magma 1.1.15.
Avec Magma, on peut alors vérifier la signature avec un programme simple.

Programme	Commentaires
`C2:=Modexp(S,eA,NA);`	`<--- Calcul de S^eA modulo NA`
`dif:=C2-C;`	`<--- Difference des messages`
`if dif eq 0 then`	`<--- Vérification de la signature`
`  print("Signature valide");`	
`else`	
`  print("Signature non valide");`	
`end if`	

1.1.6 Preuve de RSA

La justesse du cryptosystème RSA qu'un message chiffré C à partir d'un message clair M redonne bien le message original M est basée sur le théorème suivant, dû à Euler.

Théorème 1.1.16 (Euler)**.** *Soit N un nombre entier et $\phi(N)$ l'indicateur d'Euler. Si a est un entier premier avec N, alors*

$$a^{\phi(N)} \equiv 1 \pmod{N}.$$

Démonstration. Par définition, on a $\phi(N) = \#\{a \mid 0 \leq a \leq N-1,\ \text{pgcd}(a,N) = 1\}$, ce qui montre qu'on peut écrire

$$\{a \mid 0 \leq a < N,\ \text{pgcd}(a,N) = 1\} = \left\{a_1, a_2, \cdots, a_{\phi(N)},\ a_1 < a_2 < \cdots < a_{\phi(N)} < N\right\}.$$

Soit a un entier tel que $\text{pgcd}(a,N) = 1$. On considère maintenant l'ensemble

$$\left\{\overline{aa_1}, \overline{aa_2}, \cdots, \overline{aa_{\phi(N)}}\right\},$$

où $\bar{x}$ désigne la réduction de x modulo N. Si $\overline{aa_i} = \overline{aa_j}$, alors $\overline{a}(\overline{a_i} - \overline{a_j}) \equiv 0 \pmod{N}$, et donc $a_i = a_j$ puisque $\text{pgcd}(a, N) = 1$ et $|a_i - a_j| < N$. Ainsi, on a

$$\left\{a_1, a_2, \cdots, a_{\phi(N)}\right\} = \left\{\overline{aa_1}, \overline{aa_2}, \cdots, \overline{aa_{\phi(N)}}\right\}.$$

En formant les produits des éléments de ces deux ensembles, on obtient :

$$\prod_{i=1}^{\phi(N)} a_i = \prod_{i=1}^{\phi(N)} \overline{aa_i} \equiv a^{\phi(N)} \prod_{i=1}^{\phi(N)} a_i \pmod{N}.$$

De plus, pour chaque i, on a $\text{pgcd}(a_i, N) = 1$. Donc le produit $\prod_{i=1}^{\phi(N)} a_i$ est inversible modulo N, ce qui donne

$$a^{\phi(N)} \equiv 1 \pmod{N}.$$

□

1.2 Un exemple d'utilisation de RSA

Dans cette section, on donne le détail de l'utilisation du cryptosystème RSA sur un exemple concret. En effet, on montre comment chiffrer, transmettre et déchiffrer le message suivant :

Cette semaine, je suis à Oujda. Je pars le 22 mai à 13h

1.2.1 Transformation d'un texte en nombres

Il y a plusieurs façon de réaliser une correspondance entre les lettres de l'alphabet et des valeurs numériques. On peut par exemple faire correspondre la valeur 1 à la lettre **a**, la valeur 2 à la lettre **b**,..., et la valeur 26 à la lettre **z** et travailler ainsi en mode 27. Cette correspondance peut ne pas être pratique sur des textes très longs.

Une autre façon, plus efficace est d'utiliser la correspondance ASCII (American Standard Code for Information Interchange). ASCII est la norme d'encodage informatique des caractères alphanumériques de l'alphabet latin. Dans la table ci-dessous, on a représenté quelques correspondances.

Caractère	a	A	b	z	0	1	2	"espace"	à	+
Code	097	065	098	122	048	049	050	32	224	43

Maple 1.2.1.

Dans Maple 12, la fonction qui transforme un caractère en valeur numérique est **convert("texte en caractères", bytes)**. Inversement, la fonction qui transforme une liste de valeurs numériques en texte est **convert([v1,v2,...,vn],bytes)**.

Programme	Commentaires
`liste:=convert("123 abc...z", bytes)`	`<--- liste = [49, 50, 51, 32, 97, 98, 99, 46, 46, 46, 122]`
`convert(liste, bytes)`	`<--- "123 abc...z"`

Magma 1.2.2.
Dans Magma, la fonction qui transforme un caractère en valeur numérique est **BinaryString("texte en caractères")** ou **BString("texte en caractères")**. Inversement, la fonction qui transforme un code ASCII n en caractère est **CodeToString(n)**. Voici un exemple

Programme	Programme
`s:="Je suis à côté" ;`	`<--- Le texte`
`s;`	`<--- Son affichage`
`t:=BString(s);`	`<--- Transformation en code ASCII`
`t;`	`<--- On obtient [74, 101, ..., 169]`
`l:=#t;`	`<--- Longueur de la liste`
`u:=CodeToString(t[1]);`	`<--- Transformation en texte`
`for i:=2 to l do`	`<--- Formation du texte`
`u:=u*CodeToString(t[i]);`	
`end for;`	
`u;`	`<--- Affichage du texte Je suis à côté`

1.2.2 L'exemple

Dans cette partie, on exécute en détail un exemple de l'utilisation du cryptosystème RSA. On va chiffrer, puis déchiffrer le message suivant

Cette semaine, je suis à Oujda. Je pars le 22 mai à 13h

On commence écrire le programme qui sera exécuté par Maple. La fonction **seq** permet de créer une liste de valeurs et la fonction **nops(liste)** donne le nombre de termes dans son argument (liste).

Maple 1.2.3.

```
      Programme

retart:
t := 100:
x1 := round(2.0^((1/2)*t)):
x2 := round(2.0^((t+1)*(1/2))):
m1 := (rand(x1 .. x2))():
p := nextprime(m1):
m2 := (rand(x1 .. x2))():
q := nextprime(m2):
N := p*q:
phi := (p-1)*(q-1):
e := (rand(3 .. phi))():
while gcd(e, phi) <> 1 do e := (rand(3 .. phi))() end do:
d := modp(1/e, phi):
message:="Cette semaine, je suis à Oujda. Je pars le 22 mai à 13h":
listeclaire:= convert(message, bytes):
listecrypte:=[seq(modp(listeclaire[k]&^e,N),k=1..nops(listeclaire))]:
listedecrypte:=[seq(modp(listecrypte[k]&^d,N),k=1..nops(listecrypte))]:
messageoriginale:=convert(listedecrypte, bytes):
listeclaire-listedecrypte;
```

La dernière ligne permet de vérifier que le message de départ et le message décrypté sont identiques. La différence listeclaire-listedecrypte doit donner une liste de zéros.

Magma 1.2.4.

```
      Programme

t:=100;
x1:=Round(2.0^(t/2));
x2:=Round(2.0^((t+1)/2));
m1:=Random(x1,x2);
p:=NextPrime(m1);
m2:=Random(x1,x2);
q:=NextPrime(m2);
N:=p*q;
phi:=(p-1)*(q-1);
e:=Random(3,Round(phi/2));
while(GCD(e,phi) gt 1) do
e:=Random(3,Round(phi/2));
end while;
```

```
d:= InverseMod(e,phi);
message:="Cette semaine, je suis à Oujda. Je pars le 22 mai à 13h";
listeclaire:= BString(message);
l:=#listeclaire;
listecrypte:= AssociativeArray();
for k:=1 to l do
listecrypte[k]:=Modexp(listeclaire[k],e,N);
end for;
listedecrypte:= AssociativeArray();
for k:=1 to l do
listedecrypte[k]:=Modexp(listecrypte[k],d,N);
end for;
u:=CodeToString(listedecrypte[1]);
for i:=2 to l do
u:=u*CodeToString(listedecrypte[i]);
end for;
u;
```

1.3 Cryptanalyses élémentaires de RSA

Dans cette partie, on présente quelques attaques élémentaires sur le cryptosystème RSA, qui consistent à factoriser le module.

1.3.1 Cryptanalyse de RSA connaissant $\varphi(N)$

Proposition 1.3.1. *Soit N un module RSA. Si on connait $\varphi(N)$, alors on peut factoriser N.*

Démonstration. Supposons que $\varphi(N)$ est connu. Ainsi, on dispose d'un système de deux équations en p et q :

$$\begin{cases} pq & = N, \\ p+q & = N+1-\varphi(N), \end{cases}$$

qui donnent l'équation en p :

$$p^2 - (N+1-\varphi(N))p + N = 0.$$

On obtient ainsi

$$\begin{aligned} p &= \frac{N+1-\varphi(N)+\sqrt{(N+1-\varphi(N))^2-4N}}{2}, \\ q &= \frac{N+1-\varphi(N)-\sqrt{(N+1-\varphi(N))^2-4N}}{2}. \end{aligned}$$

□

Maple 1.3.2.
Avec Maple 12, cette attaque peut être programmée comme dans l'exemple suivant où on utilise la fonction **solve** qui permet de résoudre une équation.

Programme	Commentaires
p := 67676767676789:	<--- Le premier nombre premier
q := 33838383838463:	<--- Le deuxième nombre premier
N := p*q:	<--- Le module RSA
ph := (p-1)*(q-1):	<--- L'indicateur d'Euler
eq:=x^2-(N+1-ph)*x+N:	<--- L'équation
solve(eq);	<--- Résolution de l'équation
67676767676789, 33838383838463	<--- Les deux solutions

Magma 1.3.3.
Avec Magma, cette attaque peut être programmée comme dans l'exemple suivant où on utilise la fonction **Roots** qui permet de résoudre une équation.

Programme	Commentaires
P<x> := PolynomialRing(IntegerRing());	
p := 67676767676789;	<--- Le premier nombre premier
q := 33838383838463;	<--- Le deuxième nombre premier
N := p*q;	<--- Le module RSA
ph := (p-1)*(q-1);	<--- L'indicateur d'Euler
A:=x^2-(N+1-ph)*x+N;	<--- L'équation
Roots(A);	<--- Résolution de l'équation
[<33838383838463, 1>,	<--- Les deux solutions
<67676767676789, 1>]	

1.3.2 Utilisation du même module et deux exposants différents

Proposition 1.3.4. *Soit N un module RSA. Soient e_1 et e_2 deux exposants premiers entre eux. Si un message clair M est chiffré avec e_1 et e_2, alors on peut calculer M.*

Démonstration. Soient C_1 et C_2 deux messages chiffrés par

$$\begin{aligned} C_1 &\equiv M^{e_1} \pmod N, \\ C_2 &\equiv M^{e_2} \pmod N, \end{aligned}$$

et supposons que C_1 et C_2 sont rendus publiques. Si $\text{pgcd}(e_1, e_2) = 1$, alors il existe deux entiers x_1 et x_2 tels que $x_i < \frac{1}{2}e_i$ et vérifiant $e_1x_1 - e_2x_2 = \pm 1$. Ces deux entiers peuvent être déterminer par l'algorithme d'Euclide. D'autre part, ils vérifient

$$\left|\frac{e_1}{e_2} - \frac{x_2}{x_1}\right| = \frac{|e_1x_1 - e_2x_2|}{x_1e_1} = \frac{1}{x_1e_1} < \frac{1}{2x_1^2}.$$

Ainsi x_1 et x_2 peuvent être déterminés comme dénominateur et numérateur de l'une des convergentes de $\frac{e_1}{e_2}$. Si $e_1x_1 - e_2x_2 = 1$, on obtient alors

$$C_1^{x_1}C_2^{-x_2} \equiv M^{e_1x_1}M^{-e_2x_2} \equiv M^{e_1x_1-e_2x_2} \equiv M \pmod N,$$

ce qui donne le message M. Si $e_1x_1 - e_2x_2 = -1$, on calcule $C_1^{-x_1}C_2^{x_2}$ et on obtient le même résultat. □

Maple 1.3.5.

Avec Maple 12, cette attaque peut programmée comme dans l'exemple suivant.

Programme	Commentaires
`N:=22900724415936720487705353O7:`	`<--- Le module RSA`
`e1:=4543322112211:`	`<--- Le premier exposant publique`
`e2:=787654321187:`	`<--- Le deuxième exposant publique`
`igcdex(e1,e2,'x1','x2'):`	`<--- L'algorithme d'Euclide`
`x1;x2;`	`<--- Affichage de x1 et x2`
`M:=98776655441:`	`<--- Le message clair`
`C1:=modp(M&^e1,N):`	`<--- Le premier message chiffré`
`C2:=modp(M&^e2,N):`	`<--- Le deuxième message chiffré`
`M2:=modp(C1&^x1,N)`	`<--- Produit des deux messages`
`*modp(C2&^x2,N) mod N:`	
`M2-M:`	`<--- Vérification des messages`

Magma 1.3.6.

Avec Maple 12, cette attaque peut programmée comme dans l'exemple suivant. La fonction **XGCD(a,b)** retourne trois nombres entiers, g, x et y vérifiant $g = \text{pgcd}(a, b) = ax + by$.

Programme	Commentaires
N:=22900724415936720487705353O7;	<--- Le module RSA
e1:=4543322112211;	<--- Le premier exposant publique
e2:=787654321187;	<--- Le deuxième exposant publique
g,x1,x2:=XGCD(e1,e2);	<--- L'algorithme d'Euclide
g;x1;x2;	<--- Affichage de g, x1 et x2
M:=98776655441;	<--- Le message clair
C1:=Modexp(M,e1,N);	<--- Le premier message chiffré
C2:=Modexp(M,e2,N);	<--- Le deuxième message chiffré
M2:=Modexp(C1,x1,N)	<--- Produit des deux messages
*Modexp(C2,x2,N) mod N;	
M2-M;	<--- Vérification des messages

1.3.3 Utilisation de modules différents pour le même message.

Dans cette attaque, on considère qu'un message M est envoyé un certain nombre de fois en utilisant plusieurs clés publiques (N_i, e_i). Ce scénario peut se produire si le même message doit être diffusé à plusieurs destinataires. Cette attaque, due à Hastad, est basée sur le théorème des restes chinois.

Théorème 1.3.7. *Si les entiers $N_1, N_1, \cdots, N_k$ sont deux à deux premiers entre eux, alors le système*

$$\begin{cases} x = a_1 \pmod{N_1}, \\ x = a_1 \pmod{N_1}, \\ \vdots = \quad \vdots \\ x = a_k \pmod{N_k}, \end{cases}$$

admet une solution unique modulo $N = \prod_{i=1}^{k} N_i$. Cette solution est

$$x \equiv \sum_{i=1}^{k} a_i p_i M_i \pmod{N},$$

avec $p_i = \frac{N}{N_i}$ et $M_i \equiv p_i^{-1} \pmod{N_i}$.

Démonstration. Soit $N = \prod_{i=1}^{k} N_i$, $p_i = \frac{N}{N_i}$ et $M_i \equiv p_i^{-1} \pmod{N_i}$. Alors, pour chaque i, $1 \leq i \leq k$, on a $p_i M_i \equiv 1 \pmod{N_i}$ et $N_i | p_j$ si $i \neq j$. Ainsi, avec $x \equiv \sum_{i=1}^{k} a_i p_i M_i \pmod{N}$, on a pour $1 \leq i \leq k$, en considérant x modulo N_i :

$$x \equiv a_i p_i M_i + \sum_{j \neq i}^{k} a_j p_j M_j \equiv a_i + \sum_{j \neq i}^{k} a_j M_j N_i \times \frac{p_j}{N_i} \equiv a_i \pmod{N_i}.$$

Ceci montre que x est solution du système. Si x' est une autre solution du système avec $0 \leq x' < N$, alors pour chaque i, $1 \leq i \leq k$, on a

$$x' - x \equiv 0 \pmod{N_i}.$$

Puisque les entiers N_i, $1 \leq i \leq k$, sont premiers entre deux à deux, alors

$$x' - x \equiv 0 \pmod{N}$$

Ainsi, puisque $|x' - x| < N$, on a $x' = x$, ce qui montre l'unicité de la solution. □

Maple 1.3.8.
Dans Maple, le théorème des restes chinois est la fonction

$$\textbf{chrem}([a_1, a_2, \cdots, a_k], [N_1, N_2, \cdots, N_k]).$$

Voici un exemple pour résoudre le système

$$\begin{cases} x = 1 \pmod{101}, \\ x = 2 \pmod{103}, \\ x = 3 \pmod{201}. \end{cases}$$

Programme	Commentaires
`x:=chrem([1,2,3],[101,103,201]):`	`<--- Le théorème des restes chinois`
`x;`	`<--- On obtient x=1482378.`

Magma 1.3.9.
Dans Magma, le théorème des restes chinois est la fonction

$$\textbf{CRT}([a_1, a_2, \cdots, a_k], [N_1, N_2, \cdots, N_k]).$$

Voici un exemple pour résoudre le système

$$\begin{cases} x = 1 \pmod{101}, \\ x = 2 \pmod{103}, \\ x = 3 \pmod{201}. \end{cases}$$

Programme	Commentaires
`x:=CRT([1,2,3],[101,103,201]);`	`<--- Le théorème des restes chinois`
`x;`	`<--- On obtient x=1482378`

Proposition 1.3.10. *Soient $k \geq 2$ et k modules RSA N_i avec $1 \leq i \leq k$. Soient C_i, $1 \leq i \leq k$, des messages chiffrés du même message clair M à l'aide du même exposant e. Si $m^e < \prod_{i=1}^{k} N_i$ alors on peut déterminer le message clair M sans factoriser les modules.*

Démonstration. Supposons que le même message clair M est chiffré k fois par

$$C_i \equiv M^e \pmod{N_i},$$

pour $i = 1, \cdots, k$. Soit $N = \prod_{i=1}^{k} N_i$. On peut appliquer le Théorème des Restes Chinois pour résoudre le système formé des k équations

$$x \equiv C_i \pmod{N_i},$$

avec $x < N$. Si on suppose que $M^e < N$, alors $x = M^e$ en tant que nombres entiers. Ainsi

$$M = x^{\frac{1}{e}},$$

ce qui donne le message clair M. □

Maple 1.3.11.

Pour illustrer cette attaque, on teste un exemple avec $e = 3$ et $k = 4$ dans Maple.

Programme	Commentaires
`for i from 1  to 4 do`	
`N[i]:=nextprime(rand())*nextprime(rand()):`	`<--- 4 module RSA aléatoires`
`end do:`	
`e:=3:`	`<--- e=3`
`M:=5454321115654321:`	`<--- un message claire`
`for i from 1  to 4 do`	`<--- 4 messages chiffré`
`C[i]:=modp(M&^e,N[i]):`	
`end do:`	
`x:=chrem([C[1],C[2],C[3],C[4]],`	`<--- Le théorème`
`[N[1],N[2],N[3],N[4]]):`	`des restes chinois`
`M2:=round((x^(1/e))):`	`<--- racine e-ème de x`
`M2-M;`	`<--- Comparaison de M2 et M`

Magma 1.3.12.

Pour illustrer la même attaque avec Magma, on teste le même exemple avec $e = 3$ et $k = 4$. La fonction **Random(b)** retourne un nombre entier aléatoire de l'intervalle $[0, b]$.

```
        Programme

b:=2^30;
N := AssociativeArray();
for i:=1  to 4 do
N[i]:=NextPrime(Random(b))
   *NextPrime(Random(b));
end for;
e:=3;
M:=5454321115654321;
C := AssociativeArray();
for i:=1  to 4 do
    C[i]:=Modexp(M,e,N[i]);
end for;
x:=CRT([C[1],C[2],C[3],C[4]],
       [N[1],N[2],N[3],N[4]]);
M2:=Round((x^(1/e)));
M2-M;
```

```
        Programme

<--- Borne pour les nombres premiers
<--- Création du tableau N
<--- Création de 4 module RSA




<--- e=3
<--- Le message claire
<--- Création d'un tableau C
<---  Création des 4 messages chiffrés


<--- Théorème Chinois

<--- Calcule de x^(1/e)
<--- Vérification des messages
```

1.3.4 Cryptanalyse de RSA si $|p-q| < cN^{1/4}$: Méthode de Fermat

Dans cette partie, on suppose que les nombres premiers p et q qui forment le module RSA $N = pq$ sont très proches, plus précisément $|p-q| < cN^{1/4}$ où c est une constante fixe, assez petite.

Théorème 1.3.13. *Soit $N = pq$ un modules RSA où les nombres premiers p et q vérifient $|p-q| < cN^{1/4}$ où c est une constante assez petite. Alors on peut factoriser N en temps polynômial dépendant de c.*

Démonstration. La méthode de Fermat consiste en la recherche de deux nombres entiers x et y tels que

$$4N = x^2 - y^2 = (x+y)(x-y).$$

Si en plus $x - y \neq 2$, alors on obtient la factorisation de N en posant

$$p = \frac{x+y}{2}, \quad q = \frac{x-y}{2}.$$

Pour déterminer x et y, on prend pour x les valeurs $x_0 = \left[2\sqrt{N}\right]$, $x_1 = \left[2\sqrt{N}\right] + 1$, $x_2 = \left[2\sqrt{N}\right] + 2$, $\cdots$ et on teste si $4N - x^2$ est un carré parfait. On désigne alors

par k le nombre entier pour lequel $x_k = \left[2\sqrt{N}\right] + k$ donne la factorisation de N. Alors $x_k = p + q$ et on a, en supposant que $|p - q| < cN^{1/4}$:

$$\begin{aligned} k &= x_k - \left[2\sqrt{N}\right] \\ &= p + q - \left[2\sqrt{N}\right] \\ &< p + q - 2\sqrt{N} + 1 \\ &= \frac{(p+q)^2 - 4N}{p + q + 2\sqrt{N}} + 1 \\ &= \frac{(p-q)^2}{p + q + 2\sqrt{N}} + 1 \\ &< \frac{c^2\sqrt{N}}{2\sqrt{N}} + 1 \\ &< \frac{c^2}{2} + 1. \end{aligned}$$

Il en résulte que le nombre de tests est assez petits si c est une constante qui ne dépend pas de N. □

Maple 1.3.14.

On teste maintenant la méthode de Fermat avec Maple 12 sur des modules RSA $N = pq$ de taille $t = 100$, avec $|p - q| < cN^{1/4}$. Pour cela, on utilise le programme suivant qui fabrique deux nombres premiers p et q avec

$$|p - q| < 10N^{1/4}.$$

Ensuite, on exécute la méthode de Fermat pour retrouver un des nombres premiers de N.

Programme	Commentaires
`Digits := 100:`	`<--- Précision des calculs`
`t := 100:`	`<--- Taille du module`
`x1 := round(2.0^((1/2)*t)):`	`<--- Borne inférieur pour p`
`x2 := round(2.0^((t+1)*(1/2))):`	`<--- Borne supérieur pour p`

```
m1 := (rand(x1 .. x2))():          <--- Valeur aléatoire
p := nextprime(m1):                <--- Nombre premier suivant
m2 := round(p+10*2^(t/4)):         <--- Borne inférieur pour q
q := nextprime(m2):
N := p*q:                          <--- Module RSA
c := trunc(abs(p-q)/N^0.25)+1:     <--- Rapport
n := round(2*sqrt(N)):             <--- Entier le plus proche
k := 0:                            <--- Compteur
while k < (1/2)*c^2+1 do           <--- Boucle de Fermat
    x := n+k:                      <--- Une valeur pour x
    y := round(sqrt(x^2-4*N)):     <--- La valeur de y
    s := round((x+y)/2):           <--- Un candidat pour q
    if gcd(s, N) > 1 then          <--- pgcd
    print('Un facteur est', s):
    break:
    end if:
    k := k+1:
end do:
```

En exécutant ce programme, on a obtenu les résultats suivants.

```
p=1184367162931181                          <--- Un facteur premier
q=1184367498475709                          <--- L'autre facteur premier
N=1402725974037575305865967182329           <--- Le module RSA
c=10                                        <--- c vérifie abs(p-q)<cN^(1/4)
k=24                                        <--- Le nombre d'itérations
Un facteur est 1184367498475709             <--- Sortie de la procédure
```

Magma 1.3.15.

On teste ausi la méthode de Fermat avec Magma sur des modules RSA $N = pq$ de taille $t = 100$, avec $|p - q| < cN^{1/4}$. Pour cela, on utilise le programme suivant qui fabrique deux nombres premiers p et q avec

$$|p - q| < 10N^{1/4}.$$

Ensuite, on exécute la méthode de Fermat pour retrouver un des nombres premiers de N.

```
Programme

t := 100;
x1 := Round(2^((1/2)*t));
x2 := Round(2^((t+1)*(1/2)));
m1 := Random(x1,x2);
p := NextPrime(m1);
m2 := Round(p+10*2^(t/4));
q := NextPrime(m2);
N := p*q;
c := Truncate(Abs(p-q)/N^0.25)+1;
n := Round(2*Sqrt(N))+1;
k := 0;
while k lt (1/2)*c^2+1 do
    x := n+k;
    y := Round(Sqrt(x^2-4*N));
    s := Round((x+y)/2);
    if GCD(s, N) gt 1 then
    printf("Un facteur est ");s;
    break;
    end if;
    k := k+1;
end while;
```

```
Programme

<--- Taille du module
<--- Valeur minimale du nombre premier
<--- Valeur maximale du nombre premier
<--- Valeur aléatoire
<--- Premier nombre premier
<--- Valeur aléatoire
<--- Deuxième nombre premier
<--- Module RSA
<--- Le rapport Abs(p-q)/N^0.25)+1;
<--- Valeur de départ
<--- Initiation
<--- Boucle de Fermat
<--- Valeur de x
<--- Valeur de y
<--- Valeur candidate pour p


<--- Arrêt de la boucle


<--- Fin de la boucle
```

En exécutant ce programme, on a obtenu les résultats suivants.

```
p=1320614251431253                          <--- Un facteur premier
q=1320614586975679                          <--- L'autre facteur premier
N=1744022444208079680278451495787           <--- Le module RSA
c=10                                        <--- c vérifie abs(p-q)<cN^(1/4)
k=20                                        <--- Le nombre d'itérations
Un facteur est 1320614586975679             <--- Sortie de la procédure
```

Chapitre 2

Cryptanalyse de RSA par les fractions continues

Le coléreux ne peut pas atteindre la gloire et le glorieux ne ressent pas la colère.
Antara Ibno Shaddad Al Absy

لَا يَنَالُ العُلَا مَن طَبعُهُ الغَضَبُ وَ لَا يَحمِلُ الحِقدَ مَن تَعلُوا بِهِ الرُّتَبُ

عَنتَرَة إِبنُ شَدَّاد العَبسي

2.1 Les fractions continues

2.1.1 Introduction

On considère le nombre $x = 3+\sqrt{10} \approx 6.162277...$, solution de l'équation x^2-6x-1. On peut alors écrire $x^2 = 6x+1$ et donc $x = 6+\frac{1}{x}$. En recommençant le processus, on obtient

$$x = 6 + \cfrac{1}{6 + \cfrac{1}{6 + \cfrac{1}{6 + \cfrac{1}{\cdots}}}},$$

Une telle fraction, qui peut être finie, s'appelle une fraction continue, ici, c'est la fraction continue de $x = 3+\sqrt{10}$. On note plus simplement $3+\sqrt{10} = [6, 6, 6, \cdots]$. On peut aussi prendre un nombre fini d'éléments 6. On obtient ainsi

$$[6] = 6, \qquad [6, 6] = 6 + \frac{1}{6} = \frac{37}{6} \approx 6, 166666...,$$

$$[6,6,6] = 6 + \cfrac{1}{6 + \cfrac{1}{6}} = \frac{228}{37} \approx 6.162162...,$$

$$[6,6,6,6] = 6 + \cfrac{1}{6 + \cfrac{1}{6 + \cfrac{1}{6}}} = \frac{1405}{228} \approx 6.162280...,$$

Chacune des fractions $6, \frac{37}{6}, \frac{228}{37}, \frac{1405}{228}, \cdots$ s'appelle une *réduite* ou *convergentes*. On observe que ces fractions vérifient

$$|6 - x| > \left|\frac{37}{6} - x\right| > \left|\frac{228}{37} - x\right| > \left|\frac{1405}{228} - x\right|.$$

Ceci est une des nombreuses propriétés des fractions continues.

2.1.2 Définitions et propriétés

Théorème 2.1.1. *Tout nombre réel positif x a une écriture unique comme fraction continue :*

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}} = [a_1, a_2, a_3, \cdots],$$

où les a_i sont des entiers positifs. De plus, la fraction continue est finie si et seulement si le nombre x est rationnel.

Démonstration. Soit x un nombre réel positif. On pose $x_0 = x$. On considère la partie entière $a_0 = [x_0]$. Si x_0 est un entier, $a_0 = x_0$ et la fraction continue de x est $x = [a_0]$.

Supposons que x n'est pas un entier. Dans ce cas, on a $0 < x_0 - a_0 < 1$ et $\frac{1}{x_0 - a_0} > 1$. Puisque $x_0 = a_0 + (x_0 - a_0)$, on définie x_1 par

$$x_1 = \frac{1}{x_0 - a_0} > 1,$$

ce qui donne

$$x_0 = a_0 + \frac{1}{x_1}.$$

On recommence le processus pour x_1. On a $a_1 = [x_1]$. Si x_1 est un nombre entier, alors $a_1 = x_1$, et la fraction continue de x est

$$x = a_0 + \frac{1}{a_1} = \frac{a_0 + a_1}{a_1},$$

qui représente un nombre rationnelle. Si x_1, n'est pas un entier, en recommence le processus en définissant x_2 par

$$x_2 = \frac{1}{x_1 - a_1} > 1,$$

ce qui donne

$$x_0 = a_0 + \frac{1}{a_1 + \dfrac{1}{x_2}}.$$

Ce processus s'arrête au rang n si et seulement si $a_n = [x_n]$, avec $a_n \geq 1$, et dans ce cas, la fraction continue de x est

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}} = [a_1, a_2, a_3, \cdots, a_n],$$

qui représente un nombre rationnel. Si x est un nombre irrationnelle, le processus continue indéfiniment et on obtient pour x une fraction continue infinie :

$$x = [a_1, a_2, a_3, \cdots].$$

□

Définition 2.1.2. *Soit $[a_1, a_2, a_3, \cdots]$ une fraction continue d'un nombre x.*

1. *Les nombres entiers a_i s'appellent les quotients partiels.*
2. *Les nombres rationnels $[a_1, a_2, a_3, \cdots, a_k]$ s'appellent les réduites de x.*

Maple 2.1.3.

Avec le logiciel Maple, la fonction qui calcule la fraction continue d'un nombre x est est **cfrac(x,n,'quotients')** : n est le nombre de quotients partiels et 'quotients' pour afficher la fraction continue sous la forme $[a_0, a_1, a_2, \cdots, a_n]$, ce qui correspond à la notation standard. Voici un exemple pour calculer la fraction continue de $x = 1 + 7^{\frac{1}{3}}$.

Programme	Commentaire
restart:	<--- Remise à zéro
with(numtheory):	<--- Fonctionnalités "Théorie des Nombres"
Digits:=100:	<--- Précision
x:=1+7^(1/3):	<--- Un nombre réel
n:=30:	<--- nombre de quotients partiels
s:=cfrac(x,n,'quotients'):	<--- Calcul de la fraction continue
s;	<--- s=[2,1,10,2,16,....]

Magma 2.1.4.
Avec le logiciel Magma, la fonction qui calcule la fraction continue de $x = 1 + 7^{\frac{1}{3}}$ est **ContinuedFraction(x)**. Magma donne alors la forme $x = [a_1, a_2, \cdots]$, est commence donc avec a_1.

Programme	Commentaire
x:=1+7^(1/3);	<--- Déclaration de x=1+7^(1/3)
print(x);	<--- Valeur décimal de x
s:=ContinuedFraction(x);	<--- Liste des quotients partiels
s;	<--- Affichage de la liste

Pour un nombre x dont la fraction continue est $[a_0, a_1, a_2, \cdots]$, on peut facilement calculer les premières convergentes :

$$[a_0] = a_0 = \frac{p_0}{q_0}.$$

$$[a_0, a_1] = a_0 + \frac{1}{a_1} = \frac{a_0a_1 + 1}{a_1} = \frac{p_1}{q_1}.$$

$$[a_0, a_1, a_2] = a_0 + \frac{1}{a_1 + \frac{1}{a_2}} = \frac{a_0a_1a_2 + a_0 + a_2}{a_1a_2 + 1} = \frac{p_2}{q_2}.$$

On observe facilement que

$$\begin{aligned} p_2 &= a_2(a_0a_1 + 1) + a_0 = a_2p_1 + p_0, \\ q_2 &= a_2a_1 + 1 = a_2p_1 + q_0. \end{aligned}$$

Ceci est une propriété générale des convergentes d'un nombre réel.

Théorème 2.1.5. *Soit $[a_0, a_1, a_2, \cdots]$ la fraction continue de x. Pour tout $n \geq 0$, on définit les nombres p_n et q_n par,*

$$\begin{aligned} p_n &= a_np_{n-1} + p_{n-2}, \\ q_n &= a_nq_{n-1} + q_{n-2}, \end{aligned}$$

avec la convention

$$p_{-2} = 0, \quad q_{-2} = 1, \quad p_{-1} = 1, \quad q_{-1} = 0.$$

Alors tout $n \geq 0$, *on a* $\frac{p_n}{q_n} = [a_0, a_1, a_2, \cdots, a_n]$.

Démonstration. La preuve se fait par récurrence sur n. On a déjà vu que $[a_0] = \frac{p_0}{q_0}$. Supposons que

$$\begin{aligned} p_n &= a_n p_{n-1} + p_{n-2}, \\ q_n &= a_n q_{n-1} + q_{n-2}, \end{aligned}$$

et que $[a_1, a_2, a_3, \cdots, a_n] = \frac{p_n}{q_n}$. Alors, on a aussi, en utilisant la récurrence.

$$\begin{aligned} [a_1, a_2, a_3, \cdots, a_n, a_{n+1}] &= [a_1, a_2, a_3, \cdots, a_n + \frac{1}{a_{n+1}}] \\ &= \frac{\left(a_n + \frac{1}{a_{n+1}}\right) p_{n-1} + p_{n-2}}{\left(a_n + \frac{1}{a_{n+1}}\right) q_{n-1} + q_{n-2}} \\ &= \frac{(a_n a_{n+1} + 1) p_{n-1} + a_{n+1} p_{n-2}}{(a_n a_{n+1} + 1) q_{n-1} + a_{n+1} q_{n-2}} \\ &= \frac{a_{n+1} (a_n p_{n-1} + p_{n-2}) + p_{n-1}}{a_{n+1} (a_n q_{n-1} + q_{n-2}) + q_{n-1}} \\ &= \frac{a_{n+1} p_n + p_{n-1}}{a_{n+1} q_n + q_{n-1}} \\ &= \frac{p_{n+1}}{q_{n+1}}, \end{aligned}$$

et la récurrence est démontrée. Ceci termine la preuve. □

Maple 2.1.6.

Avec Maple, la fonction qui détermine les convergentes est **nthconver** : la $n+1$-ème convergente d'une liste s de convergente est **nthconver(s,n)**. Le numérateur et le dénominateur sont alors **nthnumer(s,n)** et **nthdenom(s,n)**. Voici un exemple dans lequel on cherche la convergente $\frac{p_7}{q_7}$ de $x = 1 + 7^{\frac{1}{3}}$.

Proramme	Commentaires
restart:	<--- Remise à zéro
with(numtheory):	<--- Fonctionnalités "Théorie des Nombres"
Digits:=100:	<--- Précision
x:=1+7^(1/3):	<--- Un nombre réel
s:=cfrac(x,100,'quotients'):	<--- s=[2,1,10,2,...]
n:=7:	<--- On considère la 8-ème convergentes
nthconver(s,n);	<--- La convergente d'indice 7
nthnumer(s,n);	<--- Son numérateur est 15791
nthdenom(s,n);	<--- Son dénominateur 5421

Magma 2.1.7.

Avec Magma, pour calculer la convergente $\frac{p_n}{q_n}$, il faut utiliser la fonction **Convergents(s)** où $s = [a_1, a_2, \cdots, a_n]$. La réponse est alors une matrice de la forme

$$\begin{bmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{bmatrix}.$$

Programme	Commentaires
x:=1+7^(1/3);	<--- La valeur x=1+7^(1/3)
c:=ContinuedFraction(x);	<--- La liste des quotients partiels
c;	<--- Affichage de la liste
s:=[];	<--- Initialisation
n:=7;	<--- Convergente p7/q7
for i in [1..n] do	<--- Formation de [a1,...,a7]
s:=Append(s,c[i]);end for;	
s;	<--- Affichage de [a1,...,a7]
t:=Convergents(s);	<--- Matrice ([p7,p6],[q7,q6])
t[1,1];	<--- p7
t[2,1];	<--- q7

La réponse est $p_7 = 3379$ et $q_7 = 1160$.

Dans magma, on peut aussi écrire un programme afin de déterminer la liste des numérateurs et dénominateurs des convergentes.

Programme	Commentaires
x:=1+7^(1/3);	<--- La valeur x
x;	<--- Sa valeur décimale
s:=ContinuedFraction(x);	<--- Liste des quotients partiels
s;	<--- Affichage de la liste
p0:=0;q0:=1;p1:=1;q1:=0;	<--- Initialisations
N:=[];D:=[];	<--- Initialisations
l:=15;	<--- Longueur de la liste
for i in [1..l] do	<--- Boucle
p2:=s[i]*p1+p0;	<--- Calcul de p2
q2:=s[i]*q1+q0;	<--- Calcul de p2
N:=Append(N,p2);	<--- Formation de la liste N
D:=Append(D,q2);	<--- Formation de la liste D
p0:=p1;q0:=q1;	<--- Echange
p1:=p2;q1:=q2;	<--- Echange
end for;	<--- Fin de la boucle
N;	<--- Affichage de N
D;	<--- Affichage de D
n:=7;	<--- 7ème convergente
N[n];	<--- Numérateur
D[n];	<--- Dénominateur

La septième convergente est alors $p_7 = 3379$ et $q_7 = 1160$.

Théorème 2.1.8. *Soit $[a_0, a_1, a_2, \cdots]$ la fraction continue d'un nombre x. Alors les convergentes $\frac{p_n}{q_n}$ de x vérifient pour tout $n \geq -2$*

$$\begin{aligned} p_n q_{n+1} - q_n p_{n+1} &= (-1)^{n+1}, \\ p_n q_{n+2} - q_n p_{n+2} &= (-1)^{n+1} a_{n+2}. \end{aligned}$$

Démonstration. Les preuves se font aisément par récurrence. Considérons la première égalité. Pour $n = -2$, on a bien

$$p_{-2}q_{-1} - q_{-2}p_{-1} = -1 = (-1)^{-2+1}.$$

Supposons donc que $p_{n-1}q_n - q_{n-1}p_n = (-1)^n$. On a, en utilisant le théorème 2.1.5,

$$\begin{aligned} p_n q_{n+1} - q_n p_{n+1} &= p_n(a_{n+1}q_n + q_{n-1}) - q_n(a_{n+1}p_n + p_{n-1}) \\ &= p_n q_{n-1} - q_n p_{n-1} \\ &= -(-1)^n \\ &= (-1)^{n+1}, \end{aligned}$$

ce qui démontre la récurrence.

La deuxième égalité s'obtient directement en combinant l'égalité $p_n q_{n+1} - q_n p_{n+1} = (-1)^{n+1}$ et le théorème 2.1.5.

$$\begin{aligned} p_n q_{n+2} - q_n p_{n+2} &= p_n(a_{n+2}q_{n+1} + q_n) - q_n(a_{n+2}p_{n+1} + p_n) \\ &= a_{n+2}(p_n q_{n+1} - q_n p_{n+1}) \\ &= a_{n+2}(-1)^{n+1}. \end{aligned}$$

Ceci prouve donc la deuxième égalité. □

Maple 2.1.9.

Vérifions ce théorème avec les convergentes de $x = 1 + 7^{\frac{1}{3}}$.

Programme	Commentaires
`restart:`	
`with(numtheory):`	
`Digits:=100:`	
`x:=1+7^(1/3):`	
`s:=cfrac(x,100,'quotients'):`	`<--- liste des quotients partiels`
`nu:=i->nthnumer(s,i):`	`<--- Numérateurs des convergentes`
`de:=i->nthdenom(s,i):`	`<--- Dénominateurs des convergentes`
`n:=7;`	`<--- Un indice quelconque`
`r:=nu(n)*de(n+1)-nu(n+1)*de(n):`	`<--- Première égalité`
`(-1)^(n+1)-r;`	`<--- Différence (nulle)`
`n:=10;`	`<--- Un indice quelconque`
`r:=nu(n)*de(n+2)-nu(n+2)*de(n);`	`<--- Deuxième égalité`
`(-1)^(n+1)*s[n+3]-r;`	`<--- Différence (nulle)`

Magma 2.1.10.

On vérifie maintenant le théorème 2.1.8 avec les convergentes de $x = 1 + 7^{\frac{1}{3}}$ pour $n = 7$ avec magma. Attention au décalage de l'exposant de -1 puisque Magma commence les quotients partiels par a_1 et non par a_0.

Programme	Programme
`x:=1+7^(1/3);`	`<--- Valeur de x`
`c:=ContinuedFraction(x);`	`<--- Liste des quotients partiels`
`s:=[];`	`<--- Initiation`
`t:=[];`	`<--- Initiation`
`l:=20;`	`<--- Nombre de convergentes`
`for i in [1..l] do`	`<--- Boucle`
`  s:=Append(s,c[i]);`	`<--- Liste s`
`  t:=Append(t,Convergents(s));`	`<--- Liste des convergentes`
`end for;`	
`n:=7;`	`<--- n=7`
`p1:=t[n][1,1];p2:=t[n+1][1,1];`	`<--- p7 et p8`
`q1:=t[n][2,1];q2:=t[n+1][2,1];`	`<--- q7 et q8`
`p1*q2-p2*q1-(-1)^(n);`	`<--- Vérification, réponse nulle`
`n:=7;`	
`p1:=t[n][1,1];p3:=t[n+2][1,1];`	`<--- p7 et p9`
`q1:=t[n][2,1];q3:=t[n+2][2,1];`	`<--- q7 et q9`
`p1*q3-p3*q1-(-1)^(n+2)*c[n+2];`	`<--- Vérification, réponse nulle`

Corollaire 2.1.11. *Les convergentes $\frac{p_n}{q_n}$ d'un nombre réel x vérifient $pgcd(p_n, q_n) = 1$ pour tout $n \geq 0$.*

Démonstration. Supposons que $\frac{p_{n+1}}{q_{n+1}}$ existe. Alors, d'après Théorème 2.1.8, on a $p_n q_{n+1} - q_n p_{n+1} = (-1)^{n+1}$ et donc $\text{pgcd}(p_n, q_n) = 1$. □

Théorème 2.1.12. *La suite des convergentes d'indices pairs $\frac{p_{2n}}{q_{2n}}$ et d'indices impairs $\frac{p_{2n+1}}{q_{2n+1}}$ d'un nombre irrationnel positif x vérifient :*

$$\frac{p_{2n-2}}{q_{2n-2}} < \frac{p_{2n}}{q_{2n}} < x < \frac{p_{2n+1}}{q_{2n+1}} < \frac{p_{2n-1}}{q_{2n-1}}.$$

Démonstration. Démontrons d'abord que pour tout $n \geq 0$, on a $\frac{p_{2n}}{q_{2n}} < \frac{p_{2n+1}}{q_{2n+1}}$. En effet, en utilisant le théorème 2.1.8

$$\frac{p_{2n}}{q_{2n}} - \frac{p_{2n+1}}{q_{2n+1}} = \frac{p_{2n}q_{2n+1} - q_{2n}p_{2n+1}}{q_{2n}q_{2n+1}} = \frac{(-1)^{2n+1}}{q_{2n}q_{2n+1}} < 0.$$

Démontrons maintenant que pour tout $n \geq 0$, on a $\frac{p_{2n-2}}{q_{2n-2}} < \frac{p_{2n}}{q_{2n}}$. On a, en utilisant le théorème 2.1.8 :

$$\frac{p_{2n-2}}{q_{2n-2}} - \frac{p_{2n}}{q_{2n}} = \frac{p_{2n-2}q_{2n} - p_{2n}q_{2n-2}}{q_{2n-2}q_{2n}} = \frac{(-1)^{2n-1}a_{2n}}{q_{2n-2}q_{2n}} < 0.$$

On a de même

$$\frac{p_{2n+1}}{q_{2n+1}} - \frac{p_{2n-1}}{q_{2n-1}} = \frac{p_{2n+1}q_{2n-1} - p_{2n-1}q_{2n+1}}{q_{2n+1}q_{2n-1}} = \frac{-(-1)^{2n}a_{2n+1}}{q_{2n+1}q_{2n-1}} < 0,$$

ce qui donne $\frac{p_{2n+1}}{q_{2n+1}} < \frac{p_{2n-1}}{q_{2n-1}}$.

Il reste à démontrer que les deux inégalités $\frac{p_{2n}}{q_{2n}} < x$ et $x < \frac{p_{2n+1}}{q_{2n+1}}$. En écrivant $x = [a_0, a_1, a_2, \cdots, a_n, x_{n+1}]$ avec $x_{n+1} = [a_{n+1}, \cdots]$, on obtient, en utilisant Théorème 2.1.8, pour tout $n \geq 0$:

$$x - \frac{p_n}{q_n} = \frac{x_{n+1}p_n + p_{n-1}}{x_{n+1}q_n + q_{n-1}} - \frac{p_n}{q_n} = \frac{p_{n-1}q_n - p_nq_{n-1}}{q_n(x_{n+1}q_n + q_{n-1})} = \frac{(-1)^n}{q_n(x_{n+1}q_n + q_{n-1})}.$$

Ainsi, on obtient $x - \frac{p_{2n}}{q_{2n}} > 0$ et $x - \frac{p_{2n+1}}{q_{2n+1}} < 0$. □

Le corollaire suivant est une conséquence de Théorème 2.1.12

Corollaire 2.1.13. *Si $[a_0, a_1, a_2, \cdots]$ est la fraction continue d'un nombre x, alors les convergentes $\frac{p_n}{q_n}$ de x vérifient :*

1. *Pour tout $n \geq 0$, $(q_nx - p_n)(q_{n+1}x - p_{n+1}) < 0$.*
2. *Pour tout $n \geq 0$, $|q_{n+1}x - p_{n+1}| < |q_nx - p_n|$.*

Démonstration. En écrivant $x = [a_1, a_2, a_3, \cdots, a_n, x_{n+1}] = [a_0, a_1, a_2, \cdots, a_{n+1}, x_{n+2}]$ avec $x_{n+1} = [a_{n+1}, \cdots]$ et $x_{n+2} = [a_{n+2}, \cdots]$, on obtient, comme dans la preuve du théorème 2.1.12, pour $n \geq 0$,

$$q_nx - p_n = \frac{(-1)^n}{x_{n+1}q_n + q_{n-1}}, \quad q_{n+1}x - p_{n+1} = \frac{(-1)^{n+1}}{x_{n+2}q_{n+1} + q_n}.$$

ceci montre que $q_nx - p_n$ et $q_{n+1}x - p_{n+1}$ sont de signes contraires et donc la propriété 2.

En écrivant $x_{n+1} = a_{n+1} + \frac{1}{x_{n+2}}$ avec $x_{n+2} > 1$, on a

$$a_{n+1} < x_{n+1} < a_{n+1} + 1.$$

Ainsi

$$x_{n+1}q_n + q_{n-1} < (a_{n+1} + 1)q_n + q_{n-1} = q_{n+1} + q_n < x_{n+2}q_{n+1} + q_n.$$

Donc

$$\frac{1}{x_{n+1}q_n + q_{n-1}} > \frac{1}{x_{n+2}q_{n+1} + q_n},$$

et par conséquent

$$|q_nx - p_n| > |q_{n+1}x - p_{n+1}|,$$

ce qui montre la propriété 2. □

Proposition 2.1.14. *Si $x = [a_1, a_2, a_3, \cdots]$, alors les convergentes $\frac{p_n}{q_n}$ de x vérifient pour tout $n \geq 0$,*

$$\left| x - \frac{p_n}{q_n} \right| < \frac{1}{q_n q_{n+1}}.$$

Démonstration. Puisque $x = [a_0, a_1, a_2, \cdots, a_n, x_{n+1}]$ avec $x_{n+1} = [a_{n+1}, \cdots]$, alors $x_{n+1} > a_{n+1}$. Comme dans la preuve de Théorème 2.1.12, pour $n \geq 0$, on obtient

$$\left| x - \frac{p_n}{q_n} \right| = \frac{1}{q_n(x_{n+1}q_n + q_{n-1})} < \frac{1}{q_n(a_{n+1}q_n + q_{n-1})} = \frac{1}{q_n q_{n+1}},$$

ce qui termine la preuve. □

On obtient alors la propriété suivante pour un nombre irrationnel.

Proposition 2.1.15. *Si $[a_0, a_1, a_3, \cdots]$ est la fraction continue d'un nombre irrationnel x, alors il existe une infinité de nombres rationnels $\frac{p}{q}$ tels que*

$$\left| x - \frac{p}{q} \right| < \frac{1}{q^2}.$$

Démonstration. Puisque $q_n < q_{n+1}$, alors, en utilisant la proposition 2.1.14, on obtient pour toutes les convergents $\frac{p_n}{q_n}$ de x on a

$$\left| x - \frac{p_n}{q_n} \right| < \frac{1}{q_n q_{n+1}} < \frac{1}{q_n^2}.$$

Donc, comme il y a une infinité de convergentes si x est irrationnel, il y a une infinité de nombres rationnels $\frac{p}{q}$ qui vérifient

$$\left| x - \frac{p}{q} \right| < \frac{1}{q^2}.$$

□

Nous avons aussi un théorème concernant les *meilleures* approximations.

Théorème 2.1.16 (Meilleurs approximations)**.** *Si $[a_1, a_2, a_3, \cdots]$ est la fraction continue d'un nombre irrationnel x. Soit $\frac{p_n}{q_n}$ une convergente de x avec $n \geq 0$ et $\frac{p}{q}$ un nombre rationnel.*

1. *Si $q < q_{n+1}$, alors $|q_n x - p_n| \leq |qx - p|$.*
2. *Si $q \leq q_n$, alors $\left| x - \frac{p_n}{q_n} \right| \leq \left| x - \frac{p}{q} \right|$.*

Démonstration. On suppose que $0 < q < q_{n+1}$. Montrons que $|q_n x - p_n| \leq |qx - p|$. Soit a et b deux entiers tels que

$$\begin{aligned} p &= ap_n + bp_{n+1}, \\ q &= aq_n + bq_{n+1}. \end{aligned}$$

L'existence de a et b est garantie par le théorème 2.1.8. En effet, on a $p_n q_{n+1} - p_{n+1} q_n = (-)^{n+1}$ et donc

$$a = (-1)^{n+1}(pq_{n+1} - qp_{n+1}), \qquad b = (-1)^{n+1}(qp_n - pq_n).$$

On peut discuter les valeurs de p et q suivant les valeurs de a et b.

- Si $a = 0$, alors $q = bq_{n+1}$, donc $b \geq 1$, ce qui contredit $0 < q < q_{n+1}$.

- Si $b = 0$, alors $p = ap_n$ et $q = aq_n$, ce qui donne $\left| x - \frac{p_n}{q_n} \right| = \left| x - \frac{p}{q} \right|$.

- Supposons donc que $ab \neq 0$. Puisque $q < q_{n+1}$, alors $aq_n + bq_{n+1} < q_{n+1}$ et donc $aq_n < (1 - b)q_{n+1}$, ce qui montre que $b \geq 1$ et $a < 0$ ou $b \leq -1$ et dans ce cas, on doit avoir $q = aq_n + bq_{n+1} > 0$, donc $a > 0$. Dans les deux cas, $ab < 0$. On a alors

$$qx - p = (aq_n + bq_{n+1})x - (ap_n + bp_{n+1}) = a(q_n x - p_n) + b(q_{n+1} x - p_{n+1}),$$

où les termes $a(q_n x - p_n)$ et $b(q_{n+1} x - p_{n+1})$ sont de même signe. Alors

$$|qx - p| = |a(q_n x - p_n)| + |b(q_{n+1} x - p_{n+1})| \geq |q_n x - p_n|.$$

Ceci termine la preuve de 1.

De plus, si on a $q \leq q_n$, alors

$$\left| x - \frac{p}{q} \right| = \frac{|qx - p|}{q} \geq \frac{|q_n x - p_n|}{q_n} = \left| x - \frac{p_n}{q_n} \right|,$$

et termine la preuve de 2. □

On a déjà vu dans la proposition 2.1.15 que pour $n \geq 0$, les convergentes d'un nombre réel x vérifient

$$\left| x - \frac{p_n}{q_n} \right| < \frac{1}{q_n^2}.$$

En fait, le résultat suivant est encore plus précis.

Proposition 2.1.17. *Si $[a_0, a_1, a_2, \cdots]$ est la fraction continue d'un nombre x, alors pour $n \geq 0$, on a*

$$\left| x - \frac{p_n}{q_n} \right| < \frac{1}{2q_n^2} \quad ou \quad \left| x - \frac{p_{n+1}}{q_{n+1}} \right| < \frac{1}{2q_{n+1}^2}.$$

Démonstration. On sait par le corollaire 2.1.13 que $x - \frac{p_n}{q_n}$ et $x - \frac{p_{n+1}}{q_{n+1}}$ sont de signes contraires. Alors

$$\left|\frac{p_n}{q_n} - \frac{p_{n+1}}{q_{n+1}}\right| = \left|x - \frac{p_n}{q_n}\right| + \left|x - \frac{p_{n+1}}{q_{n+1}}\right|$$

Or, en utilisant le théorème 2.1.8 et l'inégalité $2q_nq_{n+1} < q_n^2 + q_{n+1}^2$, on obtient :

$$\left|\frac{p_n}{q_n} - \frac{p_{n+1}}{q_{n+1}}\right| = \frac{1}{q_nq_{n+1}} < \frac{1}{2q_n^2} + \frac{1}{2q_{n+1}^2}.$$

Ainsi

$$\left|x - \frac{p_n}{q_n}\right| + \left|x - \frac{p_{n+1}}{q_{n+1}}\right| < \frac{1}{2q_n^2} + \frac{1}{2q_{n+1}^2}.$$

Ceci montre alors que $\left|x - \frac{p_n}{q_n}\right| < \frac{1}{2q_n^2}$ ou $\left|x - \frac{p_{n+1}}{q_{n+1}}\right| < \frac{1}{2q_{n+1}^2}$. □

On termine avec le théorème suivant, très utile pour reconnaitre les convergentes d'un nombre réel.

Proposition 2.1.18. *Si x est un nombre réel. Si $\frac{p}{q}$ est un nombre rationnel qui vérifie*

$$\left|x - \frac{p}{q}\right| < \frac{1}{2q^2},$$

alors $\frac{p}{q}$ est une convergente de x.

Démonstration. Soit $\frac{p}{q}$ est un nombre rationnel. Puisque la suite des dénominateurs (q_n) des convergentes $\frac{p_n}{q_n}$ de x est strictement croissantes, alors $q_n \leq q < q_{n+1}$ pour un entier $n \geq 0$. On a alors, en utilisant le théorème 2.1.16 et l'hypothèse $\left|x - \frac{p}{q}\right| < \frac{1}{2q^2}$,

$$\left|\frac{p}{q} - \frac{p_n}{q_n}\right| \leq \left|\frac{p}{q} - x\right| + \left|x - \frac{p_n}{q_n}\right| \leq 2\left|x - \frac{p}{q}\right| < \frac{1}{q^2}$$

Il en résulte que

$$|pq_n - p_nq| < \frac{q_n}{q} \leq 1.$$

Ainsi, $pq_n - p_nq = 0$ et donc $\frac{p}{q} = \frac{p_n}{q_n}$. □

2.2 Cryptanalyse de RSA par les fractions continues

Dans cette partie, nous considérons deux attaques basées sur l'utilisation des fractions continues. La première est due à Wiener (1990).

2.2.1 L'attaque de Wiener

Théorème 2.2.1. *Soit $N = pq$ un module RSA où les nombres premiers p et q sont de même taille ($q < p < 2q$). Soit $e < \phi(N)$ un exposant public pour lequel la clé secrète d est assez petite : $d < \frac{1}{3}N^{\frac{1}{4}}$. Connaissant N et e, on peut calculer d et factoriser N.*

Démonstration. Supposons que $q < p < 2q$. Puisque $N = pq > q^2$, alors $q < \sqrt{N}$. D'autre part, on a

$$N - \varphi(N) = p + q - 1 < 2q + q - 1 < 3q < 3\sqrt{N}.$$

Si e est une clé publique et d la clé privée, alors $d \equiv e^{-1} \pmod{\phi(N)}$, où $\phi(N)$ est l'indicateur d'Euler. Donc il existe un entier k tel que $ed - k\varphi(N) = 1$. Puisque $e < \phi(N)$, on peut donc écrire

$$k = \frac{ed-1}{\phi(N)} < \frac{ed}{\phi(N)} < d.$$

Alors

$$\begin{aligned}
\left|\frac{e}{N} - \frac{k}{d}\right| &= \frac{|ed - kN|}{Nd} \\
&= \frac{|ed - k\varphi(N) - kN + k\varphi(N)|}{Nd} \\
&= \frac{|1 - k(N - \varphi(N))|}{Nd} \\
&< \frac{k(N - \varphi(N))}{Nd} \\
&< \frac{3k\sqrt{N}}{Nd} \\
&= \frac{3k}{d\sqrt{N}}.
\end{aligned}$$

Puisque $k < d < \frac{1}{3}N^{\frac{1}{4}}$, alors

$$\frac{3k}{d\sqrt{N}} < \frac{N^{\frac{1}{4}}}{d\sqrt{N}} = \frac{1}{dN^{\frac{1}{4}}} < \frac{1}{2d^2}.$$

Ainsi, en appliquant le théorème 2.1.5, $\frac{k}{d}$ est une convergente de $\frac{e}{N}$. Connaissant d et k, on peut alors calculer $\phi(N)$ par la relation

$$\phi(N) = \frac{ed-1}{k}.$$

Ainsi, par la proposition 1.3.1 on peut calculer p et q et donc trouver la factorisatio de N. □

L'attaque de Wiener peut donc être résumée dans l'algorithme suivant.

Algorithme 7 : L'attaque de Wiener

Entrée : Un module RSA N et un exposant publique e pour lequel la clé secrète vérifie $d < \frac{1}{3}N^{\frac{1}{4}}$.

Sortie : La factorisation de N et la clé secrète d.

1: Déterminer la liste des réduites de $\frac{e}{N}$.
2: Pour chaque réduite xy telle que $y < \frac{1}{3}N^{\frac{1}{4}}$:
3: Calculer $\psi = \frac{ey-1}{x}$, $\delta = (N+1-\psi)^2 - 4N$, et $p_2 = \left[\frac{\left(N+1-\psi+\sqrt{\delta}\right)}{2}\right]$.
4: **Si** $\text{pgcd}(p_2, N) > 1$ **alors**
5: Afficher $d = y$, $p = p_2$ et $q = \frac{N}{p_2}$.
6: **Fin Si**

Maple 2.2.2.

Programme	Commentaire
`n:=100:Digits:=100:with(numtheory)`	`<-- Initiations`
`x:=rand(2^(n/2-1)..2^(n/2))():`	`<-- Une Valeur aléatoires`
`p:=nextprime(x):`	`<-- Le nombre premier p`
`y:=rand(round(x/2)..x)():`	`<-- Condition q<p<2q`
`q:= nextprime(y):`	`<-- Le nombre premier q`
`evalf(p/q):`	`<-- La condition 1<p/q<2`
`N:=p*q:`	`<-- Le module RSA`
`ph:=(p-1)*(q-1):`	`<-- L'indicateur d'Euler`
`sn:=trunc(1/3*(N^(1/4))):`	`<-- La borne de Wiener`
`d:=rand(sn)():`	`<-- Une clé privée aléatoire`
`while gcd(d,ph)>1 do`	`<-- d est premier avec ph`
`d:=rand(sn)() end do:`	
`e:=1/d mod ph:`	`<-- La clé publique`
`convert(e/N,confrac,'pq' ):`	`<-- pq est la liste des réduites`
`g:=1:i:=2:`	`<-- Initiation`
`while g=1 do`	`<-- Boucle`
`y:=denom(pq[i]):`	`<-- Dénominateur de la réduite`
`x:=numer(pq[i]):`	`<-- Numérateur de la réduite`
`psi:=(e*y-1)/x:`	`<-- la valeur psi`
`delta:=(N+1-psi)^2-4*N:`	
`p2:=((N+1-psi)+sqrt(delta))/2:`	`<-- Solution de l'équation`
`p2:=round(p2):`	
`g:=gcd(p2,N):`	`<-- calcul de pgcd(p2,N)`

```
 i:=i+1:
od:
print(‘d=‘):y;     <-- Affichage de d
print(‘p=‘):p;     <-- Affichage de p
print(‘q=‘):N/p2;  <-- Affichage de q
```

Magma 2.2.3.

On vérifie maintenant l'attaque de Wiener avec un module RSA de $n = 80$ bits.

Programme	Commentaire
n:=80;	<-- Longeur du module RSA
a:=Round(2^(n/2-1));	<-- Borne inférieure
b:=Round(2^(n/2));	<-- Borne supérieure
x:=Random(a,b);	<-- Valeur aléatoire
p:=NextPrime(x);	<-- Nombre premier
a:=Round(x/2);	<-- Borne inférieure
b:=Round(x);	<-- Borne supérieure
y:=Random(a,b);	<-- Valeur aléatoire
q:=NextPrime(y);	<-- Nombre premier
N:=p*q;	<-- Module RSA
ph:=(p-1)*(q-1);	<-- Phonction d'Euler
sn:=Truncate(1/3*(N^(1/4)));	<-- Borne de Wiener
d:=Random(sn);	<-- valeur aléatoire
while (GCD(d,ph) gt 1) do d:=Random(sn); end while;	<-- d,phi(N) premiers entre eux
e:=Modinv(d,ph);	<-- Inverse de d
c:=ContinuedFraction(e/N);	<-- Liste des quotients partiels
s:=[];	<-- Initialisation
t:=[];	<-- Initialisation
l:=#c;	<-- Longuer de c
for i in [1..l] do	<-- Fabrication des listes
s:=Append(s,c[i]);	<-- quotients partiels
t:=Append(t,Convergents(s));	<-- convergentes
end for;	
i:=3;	<-- Initialisation
g:=1;	<-- Initialisation
while (g eq 1) do	<-- Boucle de Wiener
kk:=t[i][1,2];	<-- Valeur de k
dd:=t[i][2,2];	<-- Valeur de d
phn:=(e*dd-1)/kk;	<-- Valeur phn=(e*dd-1)/kk

```
 delta:=(N+1-phn)^2-4*N;          <-- delta=(N+1-phn)^2-4*N
  x:=((N+1-phn)+Sqrt(delta))/2;   <-- Candididat pour p
 p2:=Round(x);                    <-- Candididat pour p
 g:=GCD(p2,N);                    <-- PGCD
 i:=i+1;
end while;                        <-- Fin while
p2;                               <-- Affichage de p2
p2-p;                             <-- Différence nulle
```

Chapitre 3

Cryptanalyse de RSA par l'algorithme LLL

Les objectifs ne sont pas atteints par les souhaits, mais par le travail assidu.
Abu Alaa Al Maary

وَمَا نَيلُ المَطَالِبِ بِالتَّمَنّي وَ لَاكِن تُؤخَذُ الدُّنيَا غَلَابَا
ابُو العَلَاءِ المَعَرّي

3.1 L'algorithme LLL

3.1.1 Introduction aux réseaux

L'algorithme LLL a été inventé en 1982 et porte les initiales des ses inventeurs, A.K. Lenstra, H.W. Lenstra et L. Lovász. A l'origine, son but était de factoriser des polynômes à coefficients entiers. Depuis son invention, l'algorithme LLL a été utilisé dans de nombreux domaines, en particulier dans la résolution des équations diophantiennes et la cryptanalyse, y compris celle de certaines instances de RSA. Le domaine de l'algorithme LLL est la réduction de bases dans des sous ensembles de $\mathbb{R}^n$, appelés réseaux.

Définition 3.1.1. *Soit n et d deux un entiers positifs. Soit L une partie non vide de $\mathbb{R}^n$. On dit que L est un réseau s'il existe une famille libre $(b_1 \cdots, b_d)$ de $\mathbb{R}^n$ telle que*

$$L = \sum_{i=1}^{d} \mathbb{Z} b_i = \left\{ \sum_{i=1}^{d} x_i b_i \mid x_i \in \mathbb{Z} \right\}.$$

L'entier d est la dimension du réseau, et $(b_1 \cdots , b_d)$ est une base de ce réseau.

Dans la plupart des applications, on a $d = n$ et donc la matrice de la base $(b_1 \cdots , b_d)$ est une matrice carrée. On peut en particulier calculer son déterminant, qui ne dépend pas de la base choisie.

Définition 3.1.2. *Soit L un réseau de dimension n et $(b_1 \cdots , b_n)$ une base de L. Le déterminant de L est*

$$\det(L) = |\det(b_1 \cdots , b_n)|.$$

Proposition 3.1.3. *Soit L un réseau de dimension n. Le déterminant de L est indépendant de la base.*

Démonstration. Supposons que $\det(L) = |\det(b_1 \cdots , b_n)|$, où $B = (b_1 \cdots , b_n)$ est une base de L. Soit $B' = (b'_1 \cdots , b'_n)$ une autre base de L. Alors, il existe une matrice carrée U, de type $n \times n$ à éléments dans $\mathbb{Z}$ telle que

$$\det(U) = \pm 1, \quad B' = UB.$$

Alors

$$|\det(b'_1 \cdots , b'_n)| = |\det(U)\det(b_1 \cdots , b_n)| = |\det(b_1 \cdots , b_n)| = \det(L),$$

ce qui termine la preuve. □

Le plus souvent, la base d'un réseau n'a pas de bonnes propriétés et les calculs peuvent être compliqués. Un moyen pour rendre les calculs plus efficaces est de chercher une base orthogonale dans un réseau. Pour cela, on considère le produit scalaire habituelle de deux vecteurs et la norme euclidienne d'un vecteur.

Définition 3.1.4. *Soient $x = (x_1 \cdots , x_n)$ et $y = (y_1 \cdots , y_n)$ deux vecteurs de $\mathbb{R}^n$.*

1. Le produit scalaire de x et y est

$$\langle x, y \rangle = x^T y = \sum_{i=1}^{n} x_i y_i.$$

2. La norme de x est

$$\|x\| = (\langle x, x \rangle)^{\frac{1}{2}} = \left(\sum_{i=1}^{n} x_i y_i \right)^{\frac{1}{2}}.$$

Une des méthodes les plus utilisées pour produire une base orthogonale à partir d'une base quelconque est la méthode de Gram-Schmidt.

Théorème 3.1.5 (Gram-Schmidt)**.** *Soit V un sous-espace vectoriel de dimension n et $(b_1 \cdots, b_n)$ une base de V. On considère la famille de vecteurs $(b_1^* \cdots, b_n^*)$ définie par*

$$b_1^* = b_1, \quad b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*,$$

avec pour $j < i$

$$\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}.$$

Alors $(b_1^* \cdots, b_n^*)$ est une base orthogonale de l'espace de V.

Démonstration. On va commencer par démontrer que $(b_1^* \cdots, b_n^*)$ est une base de V. En écrivant

$$b_1 = b_1^*, \quad b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^*,$$

on en conclut que, sous forme matricielle, on a

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \mu_{2,1} & 1 & 0 & 0 & \cdots & 0 \\ \mu_{3,1} & \mu_{3,2} & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mu_{n_1,1} & \mu_{n-1,2} & \mu_{n-1,3} & \cdots & 1 & 0 \\ \mu_{n,1} & \mu_{n,2} & \mu_{n,3} & \cdots & \mu_{n,n-1} & 1 \end{bmatrix} \begin{bmatrix} b_1^* \\ b_2^* \\ b_3^* \\ \vdots \\ b_n^* \end{bmatrix} = U \begin{bmatrix} b_1^* \\ b_2^* \\ b_3^* \\ \vdots \\ b_{n-1}^* \\ b_n^* \end{bmatrix}.$$

La matrice U est est une matrice inférieure, dont la diagonale est formée de 1. Son déterminant est donc 1. Ainsi $(b_1^* \cdots, b_n^*)$ est aussi une base de V.

On démontre maintenant que $(b_1^* \cdots, b_n^*)$ est orthogonale. Par récurrence, puisque $b_1^* = b_1$ et $b_2^* = b_2 - \mu_{2,1} b_1$, alors

$$\langle b_1^*, b_2^* \rangle = \langle b_1, b_2 - \mu_{2,1} b_1 \rangle = \langle b_1, b_2 \rangle - \mu_{2,1} \langle b_1, b_1 \rangle = \langle b_1, b_2 \rangle - \frac{\langle b_2, b_1 \rangle}{\langle b_1, b_1 \rangle} \langle b_1, b_1 \rangle = 0.$$

Supposons maintenant que la famille $(b_1^* \cdots, b_{i-1}^*)$ est orthogonale avec $i \geq 3$. Alors

on a pour $1 \leq k \leq i-1$

$$\begin{aligned}
\langle b_k^*, b_i^* \rangle &= \left\langle b_k^*, b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \right\rangle \\
&= \langle b_k^*, b_i \rangle - \sum_{j=1}^{i-1} \mu_{i,j} \langle b_k^*, b_j^* \rangle \\
&= \langle b_k^*, b_i \rangle - \mu_{i,k} \langle b_k^*, b_k^* \rangle \\
&= \langle b_k^*, b_i \rangle - \frac{\langle b_i, b_k^* \rangle}{\langle b_k^*, b_k^* \rangle} \langle b_k^*, b_k^* \rangle \\
&= 0.
\end{aligned}$$

Ainsi $(b_1^* \cdots, b_i^*)$ est orthogonale, ce qui prouve la récurrence et termine la preuve. □

La méthode de Gram-Schmidt peut être facilement mise en algorithme 8.

Algorithme 8 : La méthode de Gram-Schmidt

Entrée : Une base $(b_1 \cdots, b_n)$.
Sortie : Une base orthogonale $(b_1^* \cdots, b_n^*)$.

1: Poser $b_1^* = b_1$.
2: **Pour** $i = 1, 2, \cdots n$, **faire**
3: **Pour** $j = 1, 2, \cdots i-1$, **faire**
4: Calculer $\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}$.
5: **Fin Pour**
6: Calculer $b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$.
7: **Fin Pour**

On va maintenant programmer directement l'algorithme 8.

Maple 3.1.6.

Programme

```
scal:= proc (u, v)
add(u[i]*v[i], i = 1 .. nops(u));
end proc:
```

Programme	Commentaires
`gramschmidt:=proc (M)`	`<-- Procédure`
`local i, j, n, B, P, u,k:`	`<-- Paramètres locaux`
`with(LinearAlgebra):`	`<-- Librairie LinearAlgebra`
`n := nops(M):`	`<-- Nombre de lignes de M`
`B := []: P := []:`	`<-- Initiation des listes`
`for i from 1 to n do`	`<-- i=1,...,n`
`u := [seq(0, k = 1 .. n)]:`	`<-- Initiation de u`
`for j from 1 to i-1 do`	`for j from 1 to i-1 do`
`u[j]:=scal(M[i],B[j])/scal(B[j],B[j]):`	`<-- Calcul de u[j]`
`end do:`	
`u[i] := 1:`	`<-- u[i]= 1`
`P := [op(P), u]:`	`<-- concaténation`
`B:=[op(B),[seq(M[i][k]-add(u[j]*B[j][k],`	`<-- Calcul de B`
`j = 1 .. i-1), k = 1 .. n)]]:`	
`end do:`	
`return(B, P):`	`<-- Sortie`
`end proc:`	

Le programme suivant prend en entrée la base (b_1, b_2, b_3) dont la matrice est

$$b_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 1 \\ 4 \\ -2 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 0 \\ -2 \\ -1 \end{bmatrix}.$$

Programme	Commentaires
`M:=[[2,1,0],[1,4,-2],[0,-2,-1]];`	`<-- Entrée matrice M`
`gramschmidt(M);`	`<-- procédure gramschmidt`
`A:=transpose(matrix(gramschmidt(M)[1]));`	`<-- Matrice A`
`P := transpose(matrix(gramschmidt(M)[2])):`	`<-- Matrice P`
`Q:=multiply(P, A);`	`<-- Produit PA`

On obtient alors les matrices :

$$A = \begin{bmatrix} 2 & -\frac{7}{5} & \frac{10}{23} \\ 1 & \frac{14}{5} & -\frac{20}{23} \\ 0 & -2 & -\frac{35}{23} \end{bmatrix}, \quad P = \begin{bmatrix} 1 & \frac{6}{5} & -\frac{2}{5} \\ 0 & 1 & -\frac{6}{23} \\ 0 & 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 4 & -2 \\ 0 & -2 & -1 \end{bmatrix}.$$

Maple 3.1.7.
En fait, dans Maple, la méthode de Gram-Schmidt est programmée sous le nom de fonction **GramSchmidt** dans le package **LinearAlgebra**. Voici un exemple simple de son utilisation. On considère la base (b_1, b_2, b_3) avec

$$b_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 1 \\ 4 \\ -2 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 0 \\ -2 \\ -1 \end{bmatrix}.$$

Programme	Commentaires
`with(LinearAlgebra):`	`<--- package LinearAlgebra`
`b1:=Vector([2, 1, 0]):`	`<--- Premier vecteur`
`b2 := Vector([1, 4, -2]):`	`<--- Deuxième vecteur`
`b3 := Vector([0, -2, -1]):`	`<--- Troisième vecteur`
`B := GramSchmidt([b1, b2, b3]);`	`<--- La procédure de Gram-Schmidt`
`DotProduct(B[1], B[2]);`	`<--- Vérification du produit scalaire`
`DotProduct(B[1], B[3]);`	`<--- Vérification du produit scalaire`
`DotProduct(B[2], B[3]);`	`<--- Vérification du produit scalaire`

On obtient alors la base orthogonale (b_1^*, b_2^*, b_3^*) avec

$$b_1^* = b_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}, \quad b_2^* = \begin{bmatrix} -\frac{7}{5} \\ \frac{14}{5} \\ -2 \end{bmatrix}, \quad b_3^* = \begin{bmatrix} \frac{10}{23} \\ -\frac{20}{23} \\ -\frac{35}{23} \end{bmatrix}.$$

Magma 3.1.8.

Dans Magma, la méthode d'orthogonalisation de Gram-Schmidtst est programmée sous le **Orthogonalize(M)** où M est une matrice carrée. Elle retourne trois valeurs N, P et n où N est une matrice dont les colonnes représentent des vecteurs orthogonaux, P est une matrice de passage vérifiant $N = PM$ et n est le rang de M. Voici un exemple simple de son utilisation. On considère la même base (b_1, b_2, b_3) avec

$$b_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 1 \\ 4 \\ -2 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 0 \\ -2 \\ -1 \end{bmatrix}.$$

Programme	Programme
`Q:= RealField();`	`<--- Ensemble des réels`
`R:=MatrixRing(Q,3);`	`<--- Ensemble des matrices`
	`définie sur Q de type 3x3.`
`F:=elt<R|[2, 1, 0,`	`<--- F est la matrice`
`1, 4, -2,`	
`0, -2, -1]>;`	
`Orthogonalize(F) ;`	`<--- L'orthogonalisation de Gram-Shmidt`

On obtient alors la base orthogonale (b_1^*, b_2^*, b_3^*) avec

$$b_1^* = b_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}, \quad b_2^* = \begin{bmatrix} -\frac{7}{5} \\ \frac{14}{5} \\ -2 \end{bmatrix}, \quad b_3^* = \begin{bmatrix} \frac{10}{23} \\ -\frac{20}{23} \\ -\frac{35}{23} \end{bmatrix}.$$

Le résultat suivant montre qu'on peut calculer le déterminant d'un réseau si on connait une base orthogonale.

Corollaire 3.1.9 (Hadamard)**.** *Soit L un réseau de dimension n, $(b_1 \cdots, b_n)$ une base de L et $(b_1^* \cdots, b_n^*)$ la famille orthogonale au sens de Gram-Schmidt. Alors*

$$\det(L) = \prod_{i=1}^{n} \|b_i^*\| \leq \prod_{i=1}^{n} \|b_i\|.$$

Démonstration. On a $b_1 = b_1^*$, donc $\|b_1\| = \|b_1^*\|$. Pour $2 \leq i \leq n$, on a

$$b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^*,$$

et comme $\langle b_r^*, b_s^* \rangle = 0$ si $r \neq s$, alors

$$\|b_i\|^2 = \|b_i^*\|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 \|b_j^*\|^2,$$

et donc

$$\det(L)^2 = \prod_{i=1}^{n} \|b_i^*\|^2 \leq \prod_{i=1}^{n} \|b_i\|^2,$$

ce qui termine la preuve. □

Un problème très important en théorie des nombres et en cryptanalyse est la recherche d'un vecteur court dans un réseau. Deux sous-problèmes se posent alors.

Problème du vecteur le plus court (**The Shortest Vector Problem (SVP)**) :

Etant donné un réseau L. Déterminer un vecteur non nul v qui minimise la nome $\|v\|$.

Problème du vecteur le plus proche (**The Closest Vector Problem (CVP)**) :

Etant donné un réseau L et un vecteur v_0. Déterminer un vecteur $v \neq v_0$ qui minimise la nome $\|v - v_0\|$. Une réponse théorique dans le sens de ces deux problèmes est la suivante.

Théorème 3.1.10 (Hermitte)**.** *Soit L un réseau de dimension n. Alors il existe un vecteur v non nul de L tel que*

$$\|v\| \leq \sqrt{n}\det(L)^{\frac{1}{n}}.$$

Ce théorème est très théorique et la détermination d'un vecteur court est difficile en général. Un moyen pratique pour trouver des vecteurs assez courts est d'utiliser l'algorithme LLL.

3.1.2 L'algorithme LLL

L'algorithme LLL est lié à la méthode d'orthogonalisation de Gram-Schmidt. Il concerne des bases spéciales, appelées *bases réduites*.

Définition 3.1.11. *Une base $(b_1 \cdots, b_n)$ est LLL-réduite si, la base $(b_1^* \cdots, b_n^*)$ produite par la la méthode d'orthogonalisation de Gram-Schmidt vérifie*

$$|\mu_{i,j}| \leq \frac{1}{2}, \quad \textit{pour} \quad 1 \leq j < i \leq n, \tag{3.1}$$

$$\frac{3}{4}\|b_{i-1}^*\|^2 \leq \|b_i^* + \mu_{i,i-1}b_{i-1}^*\|^2, \quad \textit{pour} \quad 1 < i \leq n. \tag{3.2}$$

La deuxième condition est appelée *condition de Lovász*. Si $\mu_{i,j} = 0$ pour tout i et j, alors la base est orthogonale, et donc minimale d'après l'inégalité de Hadamard 3.1.9. Si $|\mu_{i,j}| \leq \frac{1}{2}$, la base est *presque orthogonale*, donc *presque minimale*. L'algorithme LLL original est le suivant

Maple 3.1.12.

On va maintenant programmer l'algorithme LLL. Tout d'abord, il faut programmer la procédure produit scalaire **scal(u,v)** et la procédure **gramschmidt(M)**. On programme ensuite la procédure **redfaible(M)**.

Algorithme 9 : Algorithme LLL

Entrée : Une base $(b_1, \cdots, b_n)$
Sortie : Une base LLL réduite $(b_1, \cdots, b_n)$

Pour $i = 1, \cdots, n$ **faire**
  $b_i^* = b_i$
  Pour $j = 1, \cdots, i-1$ **faire**
    $\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{B_j}$
    $b_i^* = b_i^* - \mu_{i,j} b_j^*$
  Fin Pour
  $B_i = \langle b_i^*, b_i^* \rangle$
Fin Pour
$k = 2$
Appliquer **redfaible**$(k, k-1)$
Si $\frac{3}{4} B_{k-1} > B_k + \mu_{k,k-1}^2 B_{k-1}$ **alors**
  Appliquer **lovasz**(k)
Fin Si
Aller à l'étape 10
Pour $l = k-2, \cdots, 1$ **faire**
  Appliquer **redfaible**(k, l)
Fin Pour
Si $k = n$ **alors**
  Stop
Sinon
  $k = k + 1$
  Aller à l'étape 10
Fin Si

Algorithme 10 : Procédure **redfaible**

Entrée : Un couple (k, l)
Sortie : Un vecteur b_k et des valeurs $\mu_{k,j}$ avec $j = 1, \cdots l-1$ avec $|\mu_{k,l}| \leq \frac{1}{2}$

Si $|\mu_{k,l}| > \frac{1}{2}$ **alors**
  $b_k = b_k - \lfloor \mu_{k,l} \rceil b_l$
  $\mu_{k,l} = \mu_{k,l} - \lfloor \mu_{k,l} \rceil$
  Pour $j = 1, \cdots, l-1$ **faire**
    $\mu_{k,j} = \mu_{k,j} - \lfloor \mu_{k,l} \rceil \mu_{l,j}$
  Fin Pour
Fin Si

Remarque : $\lfloor x \rceil$ est l'arrondi de x, c'est à dire l'entier le plus proche de x.

Algorithme 11 : Procédure **lovasz**

Entrée : Un entier k

Sortie : La condition de Lovasz $\frac{3}{4}B_{k-1} \leq B_k + \mu_{k,k-1}^2 B_{k-1}$ est satisfaite

1: $B = B_k + \mu_{k,k-1}^2 B_{k-1}$
2: $\mu_{k,k-1} = \frac{\mu_{k,k-1} B_{k-1}}{B}$
3: $B_k = \frac{B_{k-1} B_k}{B}$
4: $B_{k-1} = B$
5: Echanger b_k et b_{k-1}
6: **Pour** $j = 1, \cdots, k-2$ **faire**
7: Echanger $\mu_{k-1,j}$ et $\mu_{k,j}$
8: **Fin Pour**
9: **Pour** $i = k+1, \cdots, n$ **faire**
10: $\mu_{i,k-1} = \mu_{k,k-1}\mu_{i,k-1} + (1 - \mu_{k,k-1}\mu_{k,k-1})\mu_{i,k}$
11: $\mu_{i,k} = \mu_{i,k-1} - \mu_{k,k-1}\mu_{i,k}$
12: **Fin Pour**
13: **Si** $k > 2$ **alors**
14: $k = k - 1$
15: **Fin Si**

Procédure redfaible	Commentaires
`redfaible:=proc(M)`	`<-- Procédure redfaible`
`local i,j,k,U,N;`	`<-- paramètres locaux`
`N:=M;`	`<-- Initialisation`
`U:=gramschmidt(M)[2];`	`<-- Matrice de passage U`
`for i from 2 to nops(M) do`	`<-- i=2,...`
`for j from i-1 by -1 to 1 do`	`<-- j=i-1,...,1`
`N[i] := N[i]-round(U[i][j])*N[j];`	`<-- Nouveau N`
`for k from 1 to j do`	`<-- k=1,...,j`
`U[i][k]:=U[i][k]-round(U[i][j])*U[j][k]`	`<-- Nouveau U`
`end do`	`<--`
`end do`	`<--`
`end do:`	`<--`
`return N:`	`<-- renvoi de N`
`end proc:`	`<-- Fin de la procédure`

On programme maintenant la procédure **lovasz(M)**.

```
      Procédure lovasz

lovasz:=proc(M)
local i,n,G;
n:=nops(M);
G:=gramschmidt(M);
for i to n-1 do
 if evalb(scal(G[1][i+1],G[1][i+1])
  <(3/4-G[2][i+1][i]^2)*scal(G[1][i],G[1][i]))
 then return [false, i]
 end if:
 end do:
return [true, 0]:
end proc
```

Finalement, on donne la programmation de LLL sous la forme de procédure **myLLL(M)**.

```
      Procédure myLLL

myLLL:=proc(M)
local N,B,x:
N:=redfaible(M):
B:=lovasz(N):
while not(B[1]) do
 x := N[B[2]]:
 N:=redfaible(subsop(B[2] = N[B[2]+1], B[2]+1 = x, N)):
 B:=lovasz(N):
end do:
return N:
end proc:
```

Avec la matrice

$$M = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 4 & 7 & 9 & 4 \\ 6 & -7 & 2 & 3 \\ -1 & 2 & -1 & -1 \\ 2 & -1 & 0 & -3 \end{bmatrix},$$

on obtient en exécutant $myLLL(M)$ la matrice suivante.

$$N = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ -1 & 2 & -1 & -1 \\ 2 & 1 & -2 & -1 \\ -1 & 0 & 1 & -3 \\ 5 & 8 & 8 & 0 \end{bmatrix}.$$

Parmi les nombreuses propriétés d'une base LLL-réduite, on a les inégalités suivantes.

Théorème 3.1.13. *Soit $(b_1 \cdots, b_n)$ une base LLL-réduite et $(b_1^*, \cdots, b_n^*)$ la base orthogonale associée par la méthode de Gram-Schmidt. Alors*

1. $\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2$ *pour* $1 \leq j \leq i \leq n$.
2. $\prod_{i=1}^n \|b_i\| \leq 2^{\frac{n(n-1)}{4}} \det(L)$.
3. $\|b_j\| \leq 2^{\frac{i-1}{2}}\|b_i^*\|$ *pour* $1 \leq j \leq i \leq n$.
4. $\|b_1\| \leq 2^{\frac{n-1}{4}}(\det(L))^{\frac{1}{n}}$.
5. *Pour tout vecteur non nul* $v \in L$, $\|b_1\| \leq 2^{\frac{n-1}{2}}\|v\|$.

Démonstration.
`Preuve de 1.` Supposons que $(b_1 \cdots, b_n)$ une base LLL-réduite. Alors, en développant l'inégalité 3.2 et en utilisant 3.1, on a

$$\frac{3}{4}\|b_{i-1}^*\|^2 \leq \|b_i^*\|^2 + \mu_{i,i-1}^2\|b_{i-1}^*\|^2 \leq \|b_i^*\|^2 + \frac{1}{4}\|b_{i-1}^*\|^2,$$

ce qui donne

$$\|b_{i-1}^*\|^2 \leq 2\|b_i^*\|^2.$$

Ainsi, pour $j \leq i$, on obtient

$$\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2$$

ce qui démontre la propriété 1.

`Preuve de 2.` Dans le processus de l'orthogonalisation de Gram-Schmidt, on a $b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{i,j}b_j^*$,. On obtient ainsi, en utilisant 3.1 :

$$\|b_i\|^2 = \|b_i^*\|^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2\|b_j^*\|^2 \leq \|b_i^*\|^2 + \frac{1}{4}\sum_{j=1}^{i-1}\|b_j^*\|^2.$$

En utilisant $\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2$, on obtient

$$\|b_i\|^2 \leq \|b_i^*\|^2 + \frac{1}{4}\sum_{j=1}^{i-1} 2^{i-j}\|b_i^*\|^2 = \left(2^{i-2} + 2^{-1}\right)\|b_i^*\|^2 \leq 2^{i-1}\|b_i^*\|^2.$$

En passant aux produits, on a alors

$$\prod_{i=1}^{n}\|b_i\|^2 \leq \prod_{i=1}^{n} 2^{i-1}\|b_i^*\|^2 = 2^{\frac{n(n-1)}{2}}\prod_{i=1}^{n}\|b_i^*\|^2 = 2^{\frac{n(n-1)}{2}}\det(L),$$

ce qui donne l'inégalité 2 en prenant les racines carrées.

`Preuve de 3.` En remplaçant i par j dans l'inégalité $\|b_i\|^2 \leq 2^{i-1}\|b_i^*\|^2$ (ci-dessus), on obtient $\|b_j\|^2 \leq 2^{j-1}\|b_j^*\|^2$. En combinant avec l'inégalité 1, on obtient

$$\|b_j\|^2 \leq 2^{j-1}2^{i-j}\|b_i^*\|^2 = 2^{i-1}\|b_i^*\|^2,$$

ce qui démontre l'inégalité 3.

`Preuve de 4.` En prenant $j = 1$ dans l'inégalité 3, on obtient $\|b_1\|^2 \leq 2^{i-1}\|b_i^*\|^2$ pour $1 \leq i \leq n$. Alors

$$\|b_1\|^{2n} \leq \prod_{i=1} n2^{i-1}\|b_i^*\|^2 = 2^{\frac{n(n-1)}{2}}\prod_{i=1} n\|b_i^*\|^2 = 2^{\frac{n(n-1)}{2}}(\det(L))^2,$$

ce qui donne $\|b_1\| \leq 2^{\frac{n-1}{4}}(\det(L))^{\frac{1}{n}}$ et démontre l'inégalité 4.

`Preuve de 5.` Soit $v \in L$ un vecteur non nul. Alors, en exprimant v dans les bases $(b_1, b_2, \cdots, b_n)$ et $(b_1^*, b_2^*, \cdots, b_n^*)$, on obtient

$$v = \sum_{i=1}^{n}\alpha_i b_i = \sum_{i=1}^{n}\alpha_i^* b_i^*, \quad \alpha_i \in \mathbb{Z} \quad \alpha_i \in \mathbb{R}.$$

En prenant la définition de b_i dans l'orthogonalisation de Gram-Schmidt, on obtient

$$v = \sum_{i=1}^{n}\alpha_i\left(b_i^* + \sum_{j=1}^{i-1}\mu_{i,j}b_j^*\right).$$

Soit k, $1 \leq k \leq n$, le plus grand indice pour lequel $\alpha_k \neq 0$. Alors en exprimant le produit $\langle v, b_k^*\rangle$ de deux façons et en tenant compte de l'orthogonalité de $(b_1^*, b_2^*, \cdots, b_n^*)$, on obtient :

$$\langle v, b_k^*\rangle = \alpha_k\|b_k^*\|^2 = \alpha_k^*\|b_k^*\|^2.$$

Ainsi $\alpha_k = \alpha_k^*$, avec $|\alpha_k| \geq 1$. En calculant $\|v\|^2$, on obtient

$$\|v\|^2 = \sum_{i=1}^{n}(\alpha_i^*)^2\|b_i^*\|^2 \geq (\alpha_k^*)^2\|b_k^*\|^2 \geq \|b_k^*\|^2.$$

En prenant $j = 1$ et $i = k$ dans l'inégalité 3, on obtient $\|b_k^*\|^2 \geq 2^{1-k}\|b_1\|^2$. Ainsi

$$\|v\|^2 \geq 2^{1-k}\|b_1\|^2 \geq 2^{1-n}\|b_1\|^2,$$

ce qui donne $\|b_1\| \leq 2^{\frac{n-1}{2}}\|v\|$ et termine la preuve. □

Le théorème précédent concerne souvent le vecteur b_1. Plus généralement, on a le résultat suivant.

Théorème 3.1.14. *Soit $(b_1, \cdots, b_n)$ une base LLL-réduite d'un réseau. Alors pour tout j, $1 \leq j \leq n$, on a*

$$\|b_j\| \leq 2^{\frac{n(n-1)}{4(n-j+1)}} (\det L)^{\frac{1}{n-j+1}}.$$

Démonstration. Soit $(b_1^*, \cdots, b_n^*)$ la base orthogonale associée par la méthode de Gram-Schmidt à $(b_1, \cdots, b_n)$. Par le résultat (3) de la proposition 3.1.13, on a, pour $1 \leq j \leq i \leq n$

$$\|b_j\| \leq 2^{\frac{i-1}{2}}\|b_i^*\|.$$

En appliquant cette inégalité pour chaque $i = j, j+1, \cdots, n$ et en multipliant, on obtient

$$\|b_j\|^{n-j+1} \leq \prod_{i=j}^{n} 2^{\frac{i-1}{2}}\|b_i^*\|.$$

On obtient alors en commençant à $i = 1$:

$$\|b_j\|^{n-j+1} \leq \prod_{i=1}^{n} 2^{\frac{i-1}{2}}\|b_i^*\| = \prod_{i=1}^{n} 2^{\frac{i-1}{2}} \prod_{i=1}^{n} \|b_i^*\| = 2^{\frac{n(n-1)}{4}} \det(L),$$

ce qui donne le résultat recherché. □

On donne maintenant la complexité de l'algorithme LLL.

Théorème 3.1.15. *Soit $(b_1, \cdots, b_n)$ une base d'un réseau L et $B = \max_i \|b_i\|$. L'algorithme LLL produit une base réduite en un temps polynômial :*

$$\mathcal{O}\left(n^4 \log^3 B\right).$$

Maple 3.1.16.

Dans Maple 12, l'algorithme LLL est en fait déjà programmé sous le nom de procédure **LLL** dans le package **IntegerRelations**. On donne, par exemple la base (b_1, b_2, b_3, b_4) avec pour matrice

$$M = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 4 & 7 & 9 & 4 \\ 6 & -7 & 2 & 3 \\ -1 & 2 & -1 & -1 \\ 2 & -1 & 0 & -3 \end{bmatrix}.$$

Programme	Commentaires
`with(IntegerRelations):`	`<--- Package IntegerRelations`
`with(LinearAlgebra):`	`<--- Package LinearAlgebra`
`l1 := [4, 7, 9, 4];`	`<--- Premiere ligne`
`l2 := [6, -7, 2, 3];`	`<--- Deuxième ligne`
`l3 := [-1, 2, -1, -1];`	`<--- Troisième ligne`
`l4 := [2, -1, 0, -3];`	`<--- Quatrième ligne`
`N:=LLL([l1, l2, l3,l4],'integer');`	`<--- Procédure LLL, paramètres entiers`

On obtient alors la base (b_1, b_2, b_3, b_4) dont la matrice est

$$N = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ -1 & 2 & -1 & -1 \\ 2 & 1 & -2 & -1 \\ -1 & 0 & 1 & -3 \\ 5 & 8 & 8 & 0 \end{bmatrix}.$$

On peut alors vérifier que les deux bases on le même déterminant et que b'_1 est plus court que les autres vecteurs de la base.

Programme	Commentaires
`M:=Matrix([b1, b2, b3, b4]):`	`<--- Matrice M`
`N:=Matrix([m[1], m[2], m[3], m[4]]):`	`<--- Matrice N`
`Determinant(M)-Determinant(N);`	`<--- Comparaison`
`for i to 4 do`	`<--- Liste des modules`
`  DotProduct(Vector(m[i]),Vector(m[i]))`	`  et vérification que`
`end do;`	`  b1 est le plus court.`

Magma 3.1.17.

Dans Magma, l'algorithme LLL est programmé sous le nom **LLL**. On donne, le

même exemple avec la base (b_1, b_2, b_3, b_4) qui a pour matrice

$$M = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 4 & 7 & 9 & 4 \\ 6 & -7 & 2 & 3 \\ -1 & 2 & -1 & -1 \\ 2 & -1 & 0 & -3 \end{bmatrix}.$$

Programme	Commentaires
`Z := IntegerRing();`	`<--- Ensemble des entiers`
`R:=MatrixRing(Z,4);`	`<--- Ensemble des matrices de type 4x4`
`M :=elt< R|[4, 7, 9, 4,`	`<--- Matrice`
`6, -7, 2, 3,`	
`-1, 2, -1, -1,`	
`2 , -1,0, -3] >;`	
`F:=LLL(M) ;`	`<--- Procédure LLL,`
`Determinant(M);`	`<--- Déterminant de M`
`Determinant(F);`	`<--- Déterminant de F`

On obtient alors la base (b'_1, b'_2, b'_3, b'_4) dont la matrice est

$$N = \begin{bmatrix} b'_1 & b'_2 & b'_3 & b'_4 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ -1 & 2 & -1 & -1 \\ 2 & 1 & -2 & -1 \\ -1 & 0 & 1 & -3 \\ 5 & 8 & 8 & 0 \end{bmatrix}.$$

On peut alors vérifier que les deux bases on le même déterminant et que b'_1 est plus court que les autres vecteurs de la base.

Programme	Programme
`Z :=RationalField();`	`<--- Z :=RationalField();`
`V := VectorSpace(Z, 4);`	`V := VectorSpace(Z, 4);`
`x := V![-1 ,2 ,-1 ,-1];`	`x := V![-1 ,2 ,-1 ,-1];`
`Norm(x);`	`Norm(x);`
`y := V![2 , 1 ,-2, -1];`	`y := V![2 , 1 ,-2, -1];`
`Norm(y);`	`Norm(y);`
`z:= V![-1 , 0,  1, -3];`	`z:= V![-1 , 0,  1, -3];`
`Norm(z);`	`Norm(z);`
`w := V![5 , 8 , 8 , 0];`	`w := V![5 , 8 , 8 , 0];`
`Norm(w);`	`Norm(w);`

3.2 Cryptanalyse de RSA par la réduction des réseaux

3.2.1 La méthode de Coppersmith : polynômes à une variable

En 1996, D. Coppersmith a décrit une méthode pour déterminer les petites racines modulaires d'un polynôme à une variable ainsi que les petites racines entières d'un polynôme à deux variables. Soit N un entier positif (sans factorisation connue) et f un polynôme de degré d à coefficient dans $\mathbb{Z}$.

$$f(x) = \sum_{i=1}^{n} a_i x^i.$$

Chaque terme x^i est appelé un monôme. La norme Euclidienne de f est définie par

$$||f(x)|| = \sqrt{\sum_{i=1}^{n} a_i^2}.$$

Le problème de la petite racine modulaire consiste à trouver un nombre entier x_0 qui vérifie

$$\begin{aligned} &f(x_0) \equiv 0 \pmod{N}, \\ &|x_0| < X, \end{aligned}$$

où X est une borne fixée.

L'idée de Coppersmith s'applique plus généralement dans le cas suivant :

Paramètres connus :

- Un entier N.

- L'existence d'un facteur inconnu b de N avec $b > N^\beta$.

- L'expression d'un polynôme $f_b(x)$, de degrés δ.

- Une borne X pour laquelle $f_b(x_0) \equiv 0 \pmod{b}$ avec $|x_0| < X$.

Paramètres inconnus :

- Le facteur b de $\mathbb{N}$.

- La racine x_0.

La méthode de Coppersmith permet de transformer l'équation modulaire $f_b(x_0) \equiv 0 \pmod{b}$ en une équation entière $f(x) = 0$, c'est à dire sur l'ensemble des entiers. Pour déterminer l'équation entière, on fixe un entier m et on construit une série de polynômes de $\mathbb{Z}[x]$:

$$g_{jk}(x) = x^j \left(f_b(x)\right)^k N^{m-k},$$

où les valeurs de j et k dépendent de m et du degré de f. Puisque $f_b(x_0) \equiv 0 \pmod{b}$, alors $f_b(x_0) = ab$ pour un entier a et donc

$$\begin{aligned} g_{jk}(x_0) &= x_0^j \left(f_b(x)\right)^k b^{m-k} \left(\frac{N}{b}\right)^{m-k} \\ &= a^k x_0^j b^k b^{m-k} \left(\frac{N}{b}\right)^{m-k} \\ &= a^k x_0^j b^m \left(\frac{N}{b}\right)^{m-k} \\ &\equiv 0 \pmod{b^m}. \end{aligned}$$

Ceci implique que toute combinaison linéaire $h(x)$ des polynômes $g_{jk}(x)$ vérifiera $h(x_0) \equiv 0 \pmod{b^m}$. Si en plus h vérifie $|h(x_0)| < b^m$, alors on a tout simplement $h(x_0) = 0$, ce qui est facile à résoudre. Le passage de l'équation modulaire $h(x_0) \equiv 0 \pmod{b^m}$ à l'équation entière $h(x_0) = 0$ peut être fixé par le théorème suivant.

Théorème 3.2.1 (Howgrave-Graham)**.** *Soit $h(x) \in \mathbb{Z}[x]$ un polynôme de degré d ayant au plus ω monômes et X un nombre positif. Si x_0 est un entier et M un nombre positif tels que :*

(1) $|x_0| < X$,

(2) $h(x_0) \equiv 0 \pmod{M}$,

(3) $\|h(xX)\| < \frac{M}{\sqrt{\omega}}$,

alors $h(x_0) = 0$ en tant qu'équation sur $\mathbb{Z}$.

Démonstration. Soit $h(x) = \sum_i^d a_i x^i$ ayant ω monômes. Supposons que $|x_0| < X$. Alors

$$|h(x_0)| = \left| \sum_i a_i x^i \right| \leq \sum_i \left| a_i x^i \right| < \sum_i \left| a_i X^i \right|.$$

D'autre part, l'inégalité de Cauchy-Schwarz donne

$$\left(\sum_i \alpha_i \beta_i \right)^2 \leq \left(\sum_i \alpha_i^2 \right) \left(\sum_i \beta_i^2 \right).$$

En utilisant cette inégalité, on obtient

$$\left(\sum_i \left| a_i X^i \right| \right)^2 \leq \left(\sum_i 1^2 \right) \left(\sum_i \left(a_i X^i \right)^2 \right) = \omega \sum_i \left(a_i X^i \right)^2.$$

Ainsi, si $\|h(xX)\| < \frac{M}{\sqrt{\omega}}$, alors

$$\sum_i \left| a_i X^i \right| < \sqrt{\omega} \sqrt{\sum_i \left(a_i X^i \right)^2} = \sqrt{\omega} \|h(xX)\| < M.$$

Donc $|h(x_0)| < M$. Si on suppose que $h(x_0) \equiv 0 \pmod{M}$, alors $h(x_0) = 0$. □

Le problème de la résolution de la congruence $f_b(x_0) \equiv 0 \pmod{b}$ peut donc être résolu par une équation $h(x_0) = 0$ où h vérifie le théorème 3.2.1. On observe d'abord que la condition (3) de ce théorème signifie que les coefficients de h sont assez petits et que la condition (2) avec $M = b^m$ est satisfaite par toute combinaison linéaire des polynômes $g_{jk}(x)$. Ceci peut être obtenu en appliquant la réduction d'un réseau de dimension n dont une base est formée à l'aide des coefficients des polynômes $g_{jk}(xX)$. Avec l'algorithme LLL, on peut obtenir un vecteur $h(xX)$ qui vérifie la condition (3). L'algorithme LLL donnera alors des vecteurs $v_1(xX), \cdots v_n(xX)$ où n est la dimension du réseau. On sait d'après le théorème 3.1.13, que le polynôme $v_1(xX)$ va vérifier

$$\|v_1(xX)\| \leq 2^{\frac{n-1}{4}} \det(L)^{\frac{1}{n}}.$$

Ainsi, pour satisfaire la condition (3) du théorème 3.2.1, il suffit d'avoir

$$2^{\frac{n-1}{4}} \det(L)^{\frac{1}{n}} < \frac{b^m}{\sqrt{n}}. \tag{3.3}$$

Le vecteur recherché $h(xX)$ sera donc le premier vecteur $v_1(xX)$ de la base réduite.

En effet, on considère le réseau formé par les vecteurs colonnes de la matrice définie par les polynômes

$$\begin{aligned} g_{i,j}(x) &= x^j N^i (f_b(x))^{m-i}, \quad j = 0, \cdots, \delta - 1, \quad i = m, \cdots, 1, \\ h_i(x) &= x^i (f_b(x))^m, \qquad i = 0, \cdots, t-1, \end{aligned}$$

où $\delta = \deg(f_b)$ et m et t sont des entiers fixés. En explicitant les polynômes $g_{i,j}(x)$, on obtient les valeurs

(3.4)

	$j=0$ $\downarrow$	$j=1$ $\downarrow$	$j=2$ $\downarrow$	$\cdots$ $\downarrow$	$j=\delta-1$ $\downarrow$
$i=m \rightarrow$	N^m,	$N^m x$,	$N^m x^2$,	$\cdots$	$N^m x^{\delta-1}$,
$i=m-1 \rightarrow$	$N^{m-1} f$	$N^{m-1} x f$	$N^{m-1} x^2 f$	$\cdots$	$N^{m-1} x^{\delta-1} f$
$i=m-2 \rightarrow$	$N^{m-2} f^2$	$N^{m-2} x f^2$	$N^{m-2} x^2 f^2$	$\cdots$	$N^{m-2} x^{\delta-1} f^2$
$\vdots \rightarrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$i=2 \rightarrow$	$N^2 f^{m-2}$	$N^2 x f^{m-2}$	$N^2 x^2 f^{m-2}$	$\cdots$	$N^2 x^{\delta-1} f^{m-2}$
$i=1 \rightarrow$	$N f^{m-1}$	$N x f^{m-1}$	$N x^2 f^{m-1}$	$\cdots$	$N x^{\delta-1} f^{m-1}$

On observe de cette façon que les degrés des polynômes sont croissants de la position $(i, j) = (m, 0)$ à la position $(i, j) = (1, \delta - 1)$. On peut continuer ce tableau avec les polynômes $h_i(x)$.

(3.5) $$i = 0, \cdots, t-1 \Rightarrow f^m, x f^m, x^2 f^m, \cdots, x^{t-1} f^m.$$

Une fois encore les degrés sont croissants. Globalement, les degrés remplissent totalement l'intervalle $\{0, 1, \cdots, m\delta + t - 1\}$. On peut alors considérer les matrices définies par les coordonnées des différents polynômes $g_{i,j}(Xx)$ et $h_i(Xx)$, lignes par

lignes, dans la base $\left(1, x, x^2, \cdots, x^{m\delta+t-1}\right)$.

$$
\begin{aligned}
M_m &= \begin{bmatrix} N^m & & & \\ & N^m X & & \\ & & \ddots & \\ & & & N^m X^{\delta-1} \end{bmatrix}, \\
M_{m-1} &= \begin{bmatrix} - & - & - & - & N^{m-1}X^{\delta} & & & \\ - & - & - & - & - & N^{m-1}X^{\delta+1} & & \\ - & - & - & - & - & - & \ddots & \\ - & - & - & - & - & - & - & N^{m-1}X^{2\delta-1} \end{bmatrix}, \\
\vdots &= \vdots \\
M_1 &= \begin{bmatrix} - & - & \cdots & - & NX^{(m-1)\delta} & & & \\ - & - & \cdots & - & - & NX^{(m-1)\delta+1} & & \\ - & - & \cdots & - & - & - & \ddots & \\ - & - & \cdots & - & - & - & - & NX^{(m-1)\delta+\delta-1} \end{bmatrix}, \\
M_0 &= \begin{bmatrix} - & - & \cdots & - & X^{m\delta} & & & \\ - & - & \cdots & - & - & X^{m\delta+1} & & \\ - & - & \cdots & - & - & - & \ddots & \\ - & - & \cdots & - & - & - & - & X^{m\delta+t-1} \end{bmatrix}.
\end{aligned}
$$

Avec ses matrices, on forme la matrice carrée $(n \times n)$ M, avec $n = m\delta + t$, qui définie le réseau L par les colonnes de la matrice :

$$
M = \begin{bmatrix} M_m \\ M_{m-1} \\ \vdots \\ M_1 \\ M_0 \end{bmatrix}. \tag{3.6}
$$

Cette matrice est diagonale est son déterminant est donc le produit des différentes diagonales :

$$
\det(L) = N^{m\delta} \cdot N^{(m-1)\delta} \cdot \ldots \cdot N^{\delta} X^{1+2+\cdots+n-1} = N^{\frac{1}{2}m(m+1)\delta} X^{\frac{1}{2}n(n-1)}.
$$

Alors l'inégalité 3.3 devient

$$
2^{\frac{n-1}{4}} N^{\frac{m(m+1)\delta}{2n}} X^{\frac{1}{2}(n-1)} < \frac{b^m}{\sqrt{n}}.
$$

Puisque $b > N^{\beta}$, il suffit alors d'avoir

$$
2^{\frac{n-1}{4}} N^{\frac{m(m+1)\delta}{2n}} X^{\frac{1}{2}(n-1)} < \frac{N^{m\beta}}{\sqrt{n}}.
$$

En isolant X, on obtient alors

$$X < 2^{-\frac{1}{2}} n^{\frac{-1}{n-1}} N^{\frac{2mn\beta - m(m+1)\delta}{n(n-1)}}.$$

En considérant l'exposant de N dans l'inégalité précédente, on observe qu'il atteint son maximum pour

$$m_0 = \frac{2n\beta - \delta}{2\delta},$$

ce qui donne la borne

$$X < 2^{-\frac{1}{2}} n^{\frac{-1}{n-1}} N^{\frac{\beta^2}{\delta} + \frac{\beta^2}{(n-1)\delta} + \frac{\delta}{4n(n-1)} - \frac{\beta}{n-1}},$$

ce qui peut être écrit sous la forme

$$X < 2^{-\frac{1}{2}} N^{\frac{\beta^2}{\delta} - \varepsilon},$$

avec

$$\varepsilon = \frac{\log n}{(n-1)\log N} + \frac{\beta}{n-1} - \frac{\beta^2}{(n-1)\delta} - \frac{\delta}{4n(n-1)},$$

qui dépend de n mais qui vérifie $\lim_{n \to +\infty} \varepsilon = 0$.

On peut maintenant résumer cette méthode dans le théorème suivant.

Théorème 3.2.2. *Pour tout $\varepsilon > 0$, il existe un entier N_0 qui vérifie la propriété : Soit $N > N_0$ un entier de factorisation inconnue qui a un diviseur $b > N^\beta$. Soit $f_b(x)$ un polynôme de degrés δ. Toutes les solutions x_0 de la congruence $f_b(x) \equiv 0 \pmod{b}$ vérifiant*

$$|x_0| < 2^{-\frac{1}{2}} N^{\frac{\beta^2}{\delta} - \varepsilon}$$

peuvent être déterminées en temps polynômial en $\log N$.

En appliquant ce théorème avec $b = N$ et donc $\beta = 1$, on obtient le théorème suivant :

Théorème 3.2.3. *Pour tout $\varepsilon > 0$, il existe un entier N_0 qui vérifie la propriété : Soit $N > N_0$ un entier de factorisation inconnue. Soit $f(x)$ un polynôme de degrés δ. Toutes les solutions x_0 de la congruence $f(x) \equiv 0 \pmod{N}$ vérifiant*

$$|x_0| < 2^{-\frac{1}{2}} N^{\frac{1}{\delta} - \varepsilon}$$

peuvent être déterminées en temps polynômial en $\log N$.

Exemple 3.2.4.

Soit $N = 29263868053$. On veut déterminer les solutions de la congruence

$$f(x) = -111111111 - 111111110x - 111111110x^2 + x^3 \equiv 0 \pmod{N}.$$

Le degrés est $\delta = 3$ et on peut donc déterminer, en théorie, toutes les solutions x_0 qui vérifient

$$|x_0| < 2^{-\frac{1}{2}} N^{\frac{1}{\delta}-\varepsilon},$$

soit $|x_0| < 2179$ avec $\varepsilon = 0$. On pose donc $X = 2179$. On choisissant $m = 3$ et $t = 1$, on doit calculer les polynômes définies par (3.4), soit

	$j=0$	1	2
$i=3$	N^3,	N^3xX,	$N^m(xX)^2$
$i=2$	$N^2f(xX)$	$N^2xf(xX)$	$N^2(xX)^2f(xX)$
$i=1$	$Nf^2(xX)$	$N(xX)f^2(xX)$	$N(xX)^2f^2(xX)$
$i=0$	$f^3(xX)$		

On pose

$$\begin{aligned} f(x) &= a_0 + a_1x + a_2x^2 + x^3, \\ f^2(x) &= b_0 + b_1x + b_2x^2 + b_3x^3 + b_4x^4 + b_5x^5 + x^6, \\ f^3(x) &= c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 + c_6x^6 + c_7x^7 + c_8x^8 + x^9. \end{aligned}$$

Puisque $n = \delta m + t = 10$, on forme alors la matrice carrée 10×10 définie par (3.6) en utilisant (3.4) avec $m = 3$, $\delta = 3$ et (3.5) avec $t = 1$,

$$\begin{bmatrix} 1 & x & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 & x^8 & x^9 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ N^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & XN^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & X^2N^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_0N^2 & a_1XN^2 & a_2X^2N^2 & X^3N^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_0XN^2 & a_1X^2N^2 & a_2X^3N^2 & X^4N^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_0X^2N^2 & a_1X^3N^2 & a_2X^4N^2 & X^5N^2 & 0 & 0 & 0 & 0 \\ b_0N & b_1XN & b_2X^2N & b_3X^3N & b_4X^4N & b_5X^5N & X^6N & 0 & 0 & 0 \\ 0 & b_0XN & b_1X^2N & b_2X^3N & b_3X^4N & b_4X^5N & b_5X^6N & X^7N & 0 & 0 \\ 0 & 0 & b_0X^2N & b_1X^3N & b_2X^4N & b_3X^5N & b_4X^6N & b_5X^7N & X^8N & 0 \\ c_0 & c_1X & c_2X^2 & c_3X^3 & c_4X^4 & c_5X^5 & c_6X^6 & c_7X^7 & c_8X^8 & X^9 \end{bmatrix}$$

Maple 3.2.5.
On ne trouve en fait aucune solution car la valeur prise par X est très grande. En effet, dans l'inégalité $|x_0| < 2^{-\frac{1}{2}} N^{\frac{1}{\delta}-\varepsilon}$, le rôle de ε peut être très important. En prenant la valeur $X = 500$, on trouve $x_0 = 79$.

Programme	Commentaires
`with(PolynomialTools):`	`<-- Package  Polynômes`
`with(LinearAlgebra):`	`<-- Package Algèbre Linéaire`
`with(linalg):`	`<-- Un autre package`
`with(IntegerRelations):`	`<-- Package LLL`
`N:=29263868053;`	`<-- La valeur de N`
`f:=x->x^3-111111110*x^2`	`<-- La fonction f`
`    -111111110*x-111111111;`	
`d:=degree(f(x),x):`	`<-- Le degré de f`
`X:=trunc((2^(-1/2)*N^(1/d))):`	`<-- Borne supérieure des solutions`
`X:=500;`	`<-- Une borne inférieure`
`m:=3:t:=1:`	`<-- m=3 et t=1`
`n:=d*m+t:`	`<-- La dimension du réseau`
`M:=Matrix(n):`	`<-- La matrice (n,n), formée de 0`
`line:=0:`	`<-- Initiation`
`for i from 1 to m do`	`<-- Boucle de formation de`
`    i2:=m+1-i:`	`    la matrice correspondante`
`    for j from 0 to d-1 do`	`    aux polynômes g(i,j),`
`        line:=line+1:`	`    qui formera`
`    cc:=CoefficientVector`	`    l'entrée pour LLL`
` (N^i2*X^j*x^(j)*(f(X*x))^(m-i2),x):`	
`    for k from 1 to Dimension(cc) do`	
`                M[line,k]:=cc[k]:`	
`    end do:`	
`    end do:`	
`end do:`	`<-- Fin de la boucle`
`for i from 0 to t-1 do`	`<-- Boucle de formation de`
`    line:=line+1:`	`    la matrice correspondante`
`    cc:=CoefficientVector`	`    aux polynômes h(i),`
`        (x^i*X^i*(f(x*X))^m,x):`	`    qui formera`
`    for k from 1 to Dimension(cc) do`	`    l'entrée pour LLL`
`        M[line,k]:=cc[k]:`	
`    end do:`	
`end do:`	`<-- Fin de la boucle`
`VM:=[row(M,1..n)]:`	`<-- Formation de la matrice`
`L := LLL(VM, 'integer'):`	`<-- Réduction de la matrice`
`h:=0:`	`<--`
`for i from 0 to n-1 do`	`<-- Formation du polynôme h`
`    h:=h+(L[1,i+1]/X^i)*x^i:`	
`end do:`	
`h:`	
`isolve(h);`	`<-- racines entière de h`

Magma 3.2.6.
Pour Magma, on peut alors utiliser le code suivant.

```
        Programme                                  Commentaires

F<x> := PolynomialRing (Integers());          <-- Ensemble des polynômes
G<z> := PolynomialRing(Rationals());          <-- Ensemble des polynômes
N:=29263868053;                               <-- Module
f:=-111111111-111111110*x                     <-- Fonction
   -111111110*x^2+x^3;
d:=Degree(f);                                 <-- Son degré
X:=Truncate((2^(-1/2)*N^(1/d)));X:=500;       <-- Borne de Coppersmith
cc:=Coefficients(f);                          <-- Liste des coefficients de f
f:=0;                                         <-- f=0
for i:=1 to #cc do                            <-- Polynômes f de Coppersmith
    f:=f+cc[i]*x^(i-1)*X^(i-1);
end for;
m:=3;t:=1;                                    <-- Choix m=3 et t=1
n:=d*m+t;                                     <-- Ordre du réseau
M:=RMatrixSpace(IntegerRing(),n,n)!0;         <-- Matrice carrée nxn
line:=0;                                      <-- line=0
for i:=1 to m do                              <-- Boucle de formation de
    i2:=m+1-i;                                    la matrice correspondante
    for j:=0 to d-1 do                            aux polynômes g(i,j),
     line:=line+1;                                qui formera
     cc:=Coefficients                             l'entrée pour LLL
     (N^i2*X^j*x^(j)*f^(m-i2));
     for k:=1 to #(cc) do
                M[line,k]:=cc[k];
        end for;
    end for;
end for;
for i:=0 to t-1 do                            <-- Boucle de formation de
    line:=line+1;                                 la matrice correspondante
    cc:=Coefficients(x^i*X^i*f^m);                aux polynômes h(i),
    for k:=1 to #(cc) do                          qui formera
        M[line,k]:=cc[k];                         l'entrée pour LLL
    end for;
end for;
L := LLL(M);                                  <-- Algorithme LLL
p:=0;                                         <-- p=0
pp:=0;                                        <-- pp=0
```

```
for i:=0 to n-1 do                  <-- Formation du polynôme h
    pp:=pp+L[1,i+1]*z^i;
    p:=p+(L[1,i+1]/X^i)*z^i;
end for;
p;Roots(p);                         <-- polynôme h et ses racines
```

On ne trouve en fait aucune solution si la valeur prise par X est très grande. En effet, dans l'inégalité

$$|x_0| < 2^{-\frac{1}{2}} N^{\frac{1}{\delta}-\varepsilon},$$

le rôle de ε peut être très important. En prenant $X = 500$ dans le code ci-dessus, on obtient alors le polynôme :

$$\begin{aligned} h(z/X) &= 35819 * z^9 - 8381646 * z^8 + 645386742 * z^7 - 6424486034022 * z^6 \\ &+ 999721772296884 * z^5 + 35322490662759665925 * z^4 \\ &+ 13772627569229621478757 * z^3 - 12921623624227964415085 68 * z^2 \\ &- 12921976839137380826459 97 * z - 13059349889858798130738 84, \end{aligned}$$

de racine 79.

En fait, dans Magma, la méthode de Coppersmith pour résoudre l'équation polynomiale

$$f(x_0) \equiv 0 \pmod{N}, \qquad |x_0|X < \frac{1}{2} N^{\frac{1}{d}},$$

est implémentée sous la forme **SmallRoots(f,N,X)**. Dans l'exemple c–dessous, on veut résoudre l'équation

$$f(x) = x^3 - 111111110 * x^2 - 111111110 * x - 111111111 \equiv 0 \pmod{29263868053},$$

avec $|x| < 500$.

Magma 3.2.7.

```
        programme                                  programme

F<x>:=PolynomialRing(Integers());   <-- Ensemble des polynômes
N:=29263868053;                     <-- Module
f:=x^3-111111110x^2                 <-- Fonction
    -111111110x-111111111;
X:=500;                             <-- Borne
sol:=SmallRoots(f, N, X);           <-- Procédure de Coppersmith
sol;                                <-- Ensemble des solutions
```

La solution obtenue 79.

3.2.2 Factorisation de N

Une autre application de la méthode de Coppersmith est la factorisation de $N = pq$ si on connait une fraction importante de p ou de q. Plus précisément, on a le résultat suivant :

Théorème 3.2.8. *Soit $N = pq$ un module RSA dont les facteurs premiers p et q sont inconnus et tels que $q < p$. Si on connait une valeur $\tilde{p}$ telle que*

$$|\tilde{p} - p| < N^{\frac{1}{4}},$$

alors on peut factoriser N en temps polynômial en $\log N$.

Démonstration. Supposons que $|\tilde{p} - p| < N^{\frac{1}{4}}$. On considère la fonction f_p définie par $f_p(x) = x + \tilde{p}$. Alors on a $f_p(p - \tilde{p}) = p \equiv 0 \mod p$. Ainsi, $x_0 = p - \tilde{p}$ est un entier qui vérifie

$$f_p(x_0) \equiv 0 \mod p, \quad |x_0| < N^{\frac{1}{4}}.$$

On peut donc appliquer le théorème 3.2.2 avec $b = p > N^{\frac{1}{2}}$, en prenant $\beta = \frac{1}{2}$. □

Maple 3.2.9.

Voici un exemple dans lequel $N = 25353012004566062958812027956 51$ est de la forme $N = pq$. On donne aussi une approximation $p_0 = 1125899907822525$ de p et on sait que $|p - p_0| < N^{\frac{1}{4}}$. Dans le programme maple ci-dessous, la borne du théorème 3.2.2 avec $\varepsilon = 0$ s'est avérée assez grande et ne donne aucune réponse. En prenant la borne

$$X < 2^{-\frac{3}{2}} N^{\frac{1}{4}}.$$

On obtient alors une réponse avec le choix $m = 2$ et $t = 2$ car le choix $t = 1$ ne donne pas de solution non plus. On obtient alors la réponse $x_0 = -979846$, ce qui donne $p = p_0 + x_0 = 1125899906842679$.

Programme	Commentaires
`with(PolynomialTools):`	`<-- Package  Polynômes`
`with(LinearAlgebra):`	`<-- Package Algèbre Linéaire`
`with(linalg):`	`<-- Un autre package`
`with(IntegerRelations):`	`<-- Package LLL`
`p := nextprime(2^50):`	`<-- Le nombre premier p`
`q := nextprime(2^51):`	`<-- Le nombre premier q`
`N:=p*q;`	`<-- Le module RSA`
`p0 := p+979846:`	`<-- Une approximation de p`
`f := x->x+p0:`	`<-- La fonction f`
`d:=degree(f(x),x):`	`<-- Son degré`
`b:=1/2:`	`<-- L'exposant beta=1/2`
`X:=trunc((2^(-3/2)*N^(b^2/d))):`	`<-- Borne supérieure des solutions`
`m:=2:t:=2:`	`<-- m=2 et t=2`
`n:=d*m+t:`	`<-- La dimension du réseau`
`M:=Matrix(n):`	`<-- La matrice (n,n), formée de 0`
`line:=0:`	`<-- Initiation`
`for i from 1 to m do`	`<-- Boucle de formation de`
`    i2:=m+1-i:`	`    la matrice correspondante`
`    for j from 0 to d-1 do`	`    aux polynômes g(i,j),`
`        line:=line+1:`	`    qui formera`
`    cc:=CoefficientVector`	`    l'entrée pour LLL`
`  (N^i2*X^j*x^(j)*(f(X*x))^(m-i2),x):`	
`    for k from 1 to Dimension(cc) do`	
`                M[line,k]:=cc[k]:`	
`end do:end do:end do:`	`<-- Fin de la boucle`
`for i from 0 to t-1 do`	`<-- Boucle de formation de`
`    line:=line+1:`	`    la matrice correspondante`
`    cc:=CoefficientVector`	`    aux polynômes h(i),`
`        (x^i*X^i*(f(x*X))^m,x):`	`    qui formera`
`    for k from 1 to Dimension(cc) do`	`    l'entrée pour LLL`
`        M[line,k]:=cc[k]:`	
`    end do:`	
`end do:`	`<-- Fin de la boucle`
`VM:=[row(M,1..n)]:`	`<-- Formation de la matrice`
`L := LLL(VM, 'integer'):`	`<-- Réduction de la matrice`
`h:=0:`	`<-- Formation du polynôme h`
`for i from 0 to n-1 do`	
`    h:=h+(L[1,i+1]/X^i)*x^i:`	
`end do:`	
`isolve(h);`	`<-- racines entière de h`

Magma 3.2.10.
On reprend le même exemple avec $N = pq = 25353012004566062958812027956 51$. On donne aussi une approximation $p_0 = 1125899907822525$ de p et on sait que $|p - p_0| < N^{\frac{1}{4}}$. Dans le programme ci-dessous, la borne du théorème 3.2.2 avec $\varepsilon = 0$ s'est avérée assez grande et ne donne aucune réponse. En prenant la borne

$$X < 2^{-\frac{3}{2}} N^{\frac{1}{4}}.$$

On obtient alors une réponse avec le choix $m = 2$ et $t = 2$ car le choix $t = 1$ ne donne pas de solution non plus. On obtient alors la réponse $x_0 = -979846$, ce qui donne $p = p_0 + x_0 = 1125899906842679$.

```
        Programme

F<x> := PolynomialRing (Integers());G<z> := PolynomialRing(Rationals());
p := NextPrime(2^50);q := NextPrime(2^51);N:=p*q;
p0 := p+979846;f:=x+p0;d:=Degree(f);
b:=1/2;X:=Truncate((2^(-3/2)*N^(b^2/d)));cc:=Coefficients(f);f:=0;
for i:=1 to #cc do f:=f+cc[i]*x^(i-1)*X^(i-1); end for;
m:=2;t:=2;n:=d*m+t;M:=RMatrixSpace(IntegerRing(),n,n)!0;line:=0;
for i:=1 to m do
i2:=m+1-i;
for j:=0 to d-1 do
line:=line+1;
cc:=Coefficients
(N^i2*X^j*x^(j)*f^(m-i2));
for k:=1 to #(cc) do
M[line,k]:=cc[k];
end for;end for;end for;
for i:=0 to t-1 do
line:=line+1;cc:=Coefficients(x^i*X^i*f^m);
for k:=1 to #(cc) do M[line,k]:=cc[k];end for;
end for;
L := LLL(M);h:=0;pp:=0;
for i:=0 to n-1 do pp:=pp+L[1,i+1]*z^i; h:=h+(L[1,i+1]/X^i)*z^i;end for;
h;Roots(h);
```

On obient le polynôme

$h(z) = 2z^3 + 2251799819564523z^2 + 4412934291333159103051z + 216204709865121228996097 8150,$

et les solutions

$$[< -1125899907822525, 1 >, < -1959781/2, 1 >, < -979846, 1 >].$$

Alors $p0 - 979846 = p$.

Bibliographie

[1] D. Boneh, G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$, Advances in Cryptology – Eurocrypt'99, Lecture Notes in Computer Science Vol. 1592, Springer-Verlag, 1—11 (1999).

[2] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, 10(4), 233—260 (1997).

[3] G.H. Hardy, E.M. Wright. **An Introduction to the Theory of Numbers**. Oxford University Press, London (1965).

[4] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. Factoring polynomials with rational coefficients, Mathematische Annalen, Vol. 261, pp. 513—534, 1982.

[5] A. May. New RSA Vulnerabilities Using Lattice Reduction Methods. PhD thesis, University of Paderborn (2003)
`http://wwwcs.upb.de/cs/ag-bloemer/personen/alex/publikationen/`

[6] R. Rivest, A. Shamir, L. Adleman. A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM, Vol. 21 (2), 120—126 (1978).

[7] M. Wiener. Cryptanalysis of short RSA secret exponents, IEEE Transactions on Information Theory, Vol. 36, 553—558 (1990).