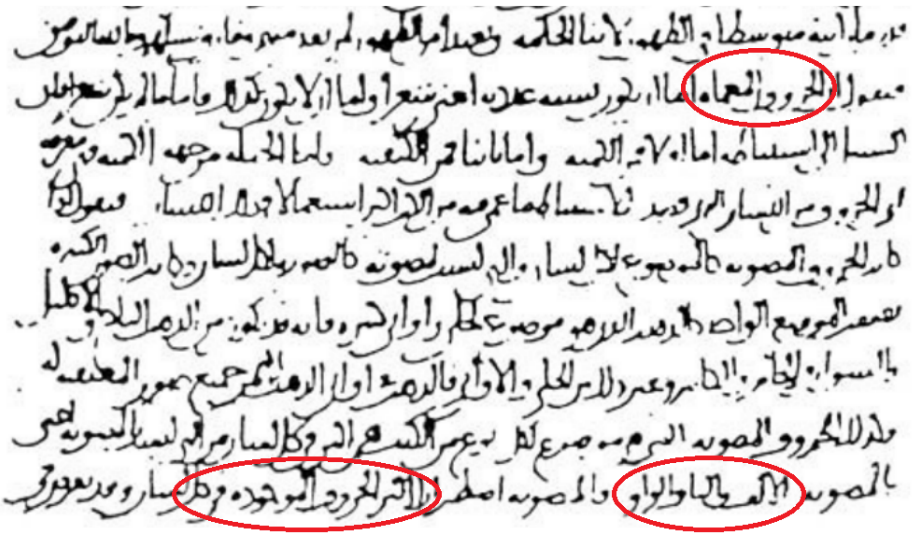
APPLICATIONS DE L'ALGORITHME LLL EN CRYPTOGRAPHIE

Abderrahmane Nitaj

Laboratoire de Mathématiques Nicolas Oresme
Université de Caen, France

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

abderrahmane.nitaj@unicaen.fr



Contents

Préface

L'invention de la cryptographie à clé publique en 1976 par Diffie et Hellman ainsi que celle du cryptosystème RSA en 1978 par Rivest, Shamir et Adleman ont contribué au développement de la théorie algorithmique des nombres. La sécurité de la plupart des cryptosystèmes modernes est basée sur des problèmes difficiles issus de la théorie de nombres, en particulier la difficulté de factoriser les grands nombres et le problème du logarithme discret. Le lien entre ces deux problèmes difficiles est la notion de nombres premiers. L'algorithmique des nombres premiers s'est alors considérablement développée et ce cours tourne autour de cette thématique. On propose ici d'étudier, tester et implémenter un grand nombre d'algorithmes nécessaires pour l'assimilation des techniques cryptographiques. Cette étude commence par les opérations arithmétiques sur les grands nombres et leurs complexités, puis continue avec les principaux tests de primalité ainsi que les principaux protocoles et cryptosystèmes modernes, en particulier RSA, Diffie-Hellman, El Gamal, NTRU et Knapsack. Toutes les implémentations sont détaillées et commentées à l'aide du system de calcul Maple.

قَالَ الشَّنْفَرَى

وَلِي دُونَكُمْ أَهلُونَ، سَيِّدٌ عَمَلَّسٌ وَ أَرْقَطُ زُهْلُولٌ وَ عَرْفَاءُ جَيْأَلُ

هُمُ الأَهْلُ لَا مُستَودَعُ السِّرِّ ذَائِعٌ لَدَيْهِمْ وَ لَا الجَانِي بِمَا جرَّ يُخْذَلُ

http://www.khayma.com/salehzayadneh/qasayed/shanfara.htm

Chapter 1

Introduction to Maple

In this chapter, we give a short introduction to the algebra system Maple and describe some basic algorithms that we will use later. Maple is a symbolic algebra system which allows computations in many fields of mathematics and engineering. We will use log to indicate the logarithm in base 2 and ln for the natural logarithm.

1.1 Basic operations

MAPLE is a symbolic algebra system which allows computations in many fields of mathematics and engineering.

```
      Maple:    Basic operations

> 10+10;
> 5*10;
> a:=790/6+3/4;
> evalf(a);
> evalf(a,8);
> evalf(a,4);
> round(a);
> floor(a);
> 2^6;
```

```
      Maple:    Digits

> Digits; evalf(1/3);
> Digits:=2;evalf(1/3);
> Digits:=50;evalf(1/3);
```

```
      Maple:    sum and product

> s:=sum(x,x=1..n);
> factor(s);
> t:=sum(x^2,x=1..n);
> factor(t);
> v:=sum(x^3,x=1..n);
> factor(v);
> product(x,x=1..10);
> w:=product(x,x=1..n);
> factor(w);
```

```
      Maple:    Trigonometry

> cos(Pi);
> sin(Pi);
> sin(Pi/4);
> tan(Pi/4);
> (cos(x))^2+(sin(x))^2;
> simplify((cos(x))^2+(sin(x))^2);
> a:=expand(sin(2x));
> combine(a);
```

```
      Maple:    Equations

> solve(x^2=4);
> solve(x^2=10);
> a:=expand((x+1)*(x^2+x+1));
> solve(a);
> isolve(a);
```

```
      Maple:    Derivatives

> diff(cos(x),x);
> diff(sin(x),x);
> diff(exp(x^2cos(x)),x);
```

```
      Maple:    Integrals

> int(cos(x),x);
> int(cos(x),x=0..Pi/2);
> int(x^2cos(x),x);
> int(x^2cos(x),x=0..Pi/2);
```

```
      Maple:    igcd

> igcd(13,10);
> igcd(15,25);
> igcd(35,42);#{what is the role of igcd(a,b)?}
```

```
      Maple:    igcdex

> igcdex(15,25,'u','v');u;v;
> 15*u+25*v;#{what is the role of igcdex?}
> igcd(151,250009);
> igcdex(151,250009,'u','v');u;v;
> 151*u+250009*v;
```

```
      Maple:    Modulo and powers

> with(numtheory):
> 10 mod 4;
> u:=7;v:=1/7 mod 10;
> u*v;
> u*v mod 10;
> 70000002^254000000;
> (70000002^254000000) mod 10;
> 70000002&^254000000 mod 10;
> modp(power(70000002, 254000000), 10);
```

1.2 Factorization and prime numbers

```
      Maple:    Factorization of polynomials

factor(x^5-1);
factor(x^12-1);
```

```
      Maple:    Factorization, details

?factor
```

```
      Maple:    Integer factorization, details

?ifactor
```

```
      Maple:    Integer factorization


ifactor(25);
ifactor(10^20-1);
```

```
      Maple:    Integer factorization, details

showstat(ifactor);
```

```
      Maple:    igcd, igcex iquo, irem

igcd(1005, 205);
igcdex(1005, 205, 'u', 'v');
u; v;
1005*u+205*v;
iquo(1005,205);
irem(1005,205);
```

```
      Maple:    Prime numbers

restart;
with(numtheory);
```

```
      Maple:    Prime numbers

nextprime(10);
nextprime(20);
prevprime(10);
prevprime(100);
isprime(12);
isprime(19);
```

```
      Maple:    Prime numbers

> restart;
> with(numtheory);
> nextprime(10);
> nextprime(20);#{what is the role of nextprime?}
> prevprime(10);
> prevprime(100);#{what is the role of prevprime?}
> isprime(12);
> isprime(19);#{what is the role of isprime?}
> ithprime(1);ithprime(2);ithprime(3);ithprime(4);ithprime(5);ithprime(6);
> ithprime(10);#{what is the role of ithprime?}
> S:=[]: for i to 100 do S:=[op(S),ithprime(i)] end do: S;
> nops(S);
> S[1];
> S[50];
> T:=[seq(ithprime(i),i=1..100)]:T;
> S-T;
> i:='i';
> sum(T[i],i=1..nops(T));
> product(T[i],i=1..nops(T));
```

Exercice 1.2.1. Find the list S of all prime numbers of the form $5i+2$ for $i = 1, \ldots, 1000$.

```
      Maple:    igcd, igcex iquo, irem

S:=[];
for i from 1 to 1000 do
    p:=5*i+2;
    if isprime(p) then
        S:=[op(S),p];
    end if;
end do;
S;
```

1.3 Lists and arrays

```
      Maple:    Lists with brackets

> list1 := [1, 2, 3, 4, 5,99];
> list2 := [10, 20, 30, 40, 50,99];
> list1+list2;
> list1-list2;
> list1*list2;
> op(list1);
> [op(list1),op(list2)];
> nops(list1);
> [op(list1),op(list2)];
> [op(list1),1000];
> [1000, op(list1)];
> list2[1..3];
> [op(list2[1..3]),1000,op(list2[4..nops(list2)])];
> subsop(1 = a, list1);
> subsop(2 = a, list1);
> subsop(1 = NULL, list1);
> subsop(3 = NULL, list1);
```

```
      Maple:    Lists with braces

> list1 := {1, 2, 3, 4, 5,10,20};
> list2 := {10, 20, 30, 40, 50};
> list1+list2;
> list1 union list2;
> op(list1);
> {op(list1),op(list2)};
> nops(list1);
> [op(list1),op(list2)];
> list1 intersect list2;
> list1 minus {20};
> list2[1..3];
> {op(list2[1..3]),1000,op(list2[4..nops(list2)])};
> subsop(1 = a, list1);
> subsop(2 = a, list1);
> subsop(1 = NULL, list1);
> subsop(3 = NULL, list1);
```

Exercice 1.3.1. Give the list of prime numbers of the form $ai + bi$ for $i, j = 1, \ldots, n$.

```
      Maple:    list of prime numbers

testprim:=proc(a,b)
local i,j;
global S;
S:=[];
for i from 1 to 10 do
    for j from 1 to 10
        p:=a*i+b*j;
        if isprime(p) then
            S:=[op(S),p];
        end if;
    end do;
end do;
return S;
end proc:
```

```
      Maple:    list of prime numbers

testprim2:=proc(a,b,n)
local i,j,p;
global S;
S:={};
for i from 1 to n do
    for j from 1 to n
        p:=a*i+b*j;
        if isprime(p) then
            S:=S union {p}
        end if;
    end do;
end do;
return S;
end proc:
```

1.4 Vectors and matrices

```
      Maple:    Modulo and power

> u := vector([1, b, c, d]);
> v := vector([1, b, c, d]);
> 2*u;
> 2*v;
> M:=matrix([[1,2],[3,4],[5,6]]);
> N:=Matrix([[1,2],[3,4],[5,6]]);
> 2*M;
> 2*N;
> M := matrix([[1, 2], [3, 4]]);
> N := matrix([[10, 20], [30, 40]]);
> multiply(M,N);
> M := Matrix([[1, 2], [3, 4]]);
> N := Matrix([[10, 20], [30, 40]]);
> multiply(M,N);
> with(LinearAlgebra);
> Multiply(M,N);
> 10*M;
> Determinant(M);
> MatrixInverse(M);
```

1.5 Procedures

```
      Maple:    If
> x:=1:y:=2:
> if (x>y) then maxi:=x else maxi:=y: end if:maxi;

> x:=5:y:=2:
> if (x>y) then maxi:=x else maxi:=y: end if:maxi;
```

```
      Maple:    Procedure with if

maxim:=proc(x,y)
    local maxi;
    if (x>y) then maxi:=x else maxi:=y:
    end if:
    return(maxi);
end proc:

maxim(10,2);
maxim(20,100);
```

```
      Maple:    Procedure

ss:=proc(n)
local i,S;
S:=0;
for i from 1 to n do
    S:=S+i^2;
end do;
return(S);
end proc:

ss(10);
ss(1000);
```

Exercice 1.5.1. 1. Copy and execute the following Maple procedure.

```
      Maple:    A procedure

count:=proc(n)
local i,S;
if n<2 then
    return( "Try n>1");
else
S:=0;
for i from 2 to n do
    if isprime(i) then
        S:=S+1;
    end if;
end do;
end if;
return S;
end proc:
```

2. Find the list of the prime numbers less than 10.
3. Execute count(10).
4. Find the list of the prime numbers less than 30.
5. Execute count(30).
6. What is the output of count(n).

Exercice 1.5.2. 1. Copy and execute the following Maple procedure.

```
      Maple:    A procedure

lprimes:=proc(n)
local i,L;
if n<2 then
    return( "Try n>1");
else
L:={};
for i from 2 to n do
    if isprime(i) then
        L:=L union {i};
    end if;
end do;
end if;
return L;
end proc:
```

2. Execute lprimes(10);
3. Execute lprimes(20);
4. What is the output of lprimes(n)?
5. Find the list of the prime numbers less than 1000.

Exercice 1.5.3. 1. Find the meaning of the following functions.

(a) nops(L)
(b) op(L)
(c) [op(L),a]
(d) [a,op(L)]
(e) op(i,L)

(f) subsop(i = a,L)

(g) subsop(1 = NULL, L)

2. What is the output of L:=lprimes(20)?
3. Find the number operands in the list L.
4. Find the second operand of the list L.
5. Add 23 at the end of the list L.
6. Add -1 at the beginning of the list L.
7. Substitute the first operand of L by 0.

Exercice 1.5.4.

1. Find the definition of **maxdigits**.
2. Execute kernelopts(maxdigits);
3. Find the number of digits of a^n with $n = 10^{10}$ and $a := 5$. Compare the result with kernelopts(maxdigits).

Chapter 2

Le Cryptosystème RSA

2.1 The Modular Exponentiation

Let a, m and n be positive integers. A practical concern in implementing many cryptographic protocols is the computation of the modular exponentiation $a^m \pmod n$. It can be performed efficiently with the repeated square-and-multiply algorithm. Let the binary representation of m be

$$m = \sum_{i=0}^{k} m_i 2^i, \quad m_i \in \{0, 1\},$$

where $k = \lfloor \log m \rfloor$. Then

$$\begin{aligned} a^m &= \left(a^{2^k}\right)^{m_k} \cdot \left(a^{2^{k-1}}\right)^{m_{k-1}} \cdots \left(a^{2^1}\right)^{m_1} \cdot \left(a^{2^0}\right)^{m_0}, \\ a^m &= \left(\left(\left((a^{m_k})^2 \cdot a^{m_{k-1}}\right)^2 \cdot a^{m_{k-2}}\right)^2 \cdots a^{m_1}\right)^2 \cdot a^{m_0}. \end{aligned}$$

For example, for $m = 100_{10} = 1100100_2$, we get

$$a^m \equiv \left(\left(\left(\left(\left((a)^2 \cdot a\right)^2\right)^2 \cdot a\right)^2\right)^2,$$

which gives rise to the sequence of exponents $2, 3, 6, 12, 24, 25, 50, 100$. From the above formulations, we deduce the repeated square-and-multiply algorithm for modular exponentiation. A direct consequence is that calculating $a^m \pmod n$ takes $O(\log m)$ multiplications.

The following algorithm computes the modular exponentiation.

Algorithm 1 Modular Exponentiation

Input : Integers a, m and n.

Output : The modular exponentiation $b \equiv a^m \pmod{n}$.

If $a \pmod{n} = 0$ **then**
    $b = 0$.
Else If m=0 **then**
    $b = 1$.
Else
    $b = 1$.
    $s = a$.
    $u = m$.
    While $u > 0$ **do**
        If $u \pmod{n} = 1$ **then**
            $b = bs \pmod{n}$.
        End If
        $s = s^2 \pmod{n}$.
        $u = \left\lfloor \frac{u}{2} \right\rfloor$.
    End While
    Return b.
End If

Complexity of Modular Exponentiation: $\mathcal{O}(\log m)$**.**

Maple code 2.1.1.

```
      Maple: Modular Exponentiation

modexp:=proc(a, m, n)
local s,b,u;
if (a mod n)=0 then
    b:=0;
    else if m=0 then
            b:=1;
            else
            b:=1;s:=a;u:=m;
            while u>0 do
                if u mod 2=1 then b:=b*s mod n;
                fi;
                s:=s*s mod n;
                u:=trunc(u/2);
            od;
        fi;
fi;
return(b);
end proc:
```

Example 2.1.2.

Let $a = 17$, $m = 200$ and $n = 230$. Then performing **modexp(a,m,n)**, we get $17^{200} \equiv 151 \pmod{2}30$.

Maple code 2.1.3.

In Maple 12, the modular exponentiation $a^m \pmod{n}$ is represented by

power(a,m) mod n or by the expression

`a&^m mod n.`

```
      Maple: power(a,m) mod n

aa := 457;
mm := 77;
nn := 212781;
aa^mm mod  nn;
power(aa,mm) mod nn;
```

```
      Maple: Power(a,m) mod n, a large example

aa := 457;
mm := 7000000007;
nn := 212781;
power(aa,mm) mod nn;
aa^mm mod  nn;
```

2.2 Euler's Totient Function

Definition 2.2.1. For a positive integer n, the Euler ϕ function is defined by

$$\phi(n) = \#\left\{a \in \mathbb{N} : 1 \leq a < n, \gcd(a,n) = 1\right\}.$$

Exercice 2.2.2.

1. Compute $\phi(p)$.
2. Compute $\phi(p^2)$.
3. Compute $\phi(5^7)$.
4. Compute $\phi(p^k)$.

Corollary 2.2.3. *If p is a prime number and $k \geq 1$, then $\phi(p^k) = p^{k-1}(p-1)$.*

Proof. We have

$$\begin{aligned}
\phi(p^k) &= \#\left\{a \in \mathbb{N} : 1 \leq a < p^k, \gcd(a,p) = 1\right\} \\
&= p^k - \#\left\{a \in \mathbb{N} : 1 \leq a \leq p^k, \gcd(a,p) = p\right\} \\
&= p^k - \#\left\{a \in \mathbb{N} : 1 \leq a \leq p^k, p|a\right\} \\
&= p^k - \#\left\{p, 2p, \ldots, p^k\right\} \\
&= p^k - p^{k-1} \\
&= p^{k-1}(p-1).
\end{aligned}$$

□

An important property of the Euler ϕ function is the multiplication.

Theorem 2.2.4. *For a positive integer $n = n_1 n_2 \ldots n_k$ where $\gcd(n_i, n_j) = 1$, the Euler ϕ function of n satisfies*

$$\phi(n) = \phi(n_1)\phi(n_2)\ldots\phi(n_k).$$

Proof. Apply the Chinese Theorem. □

Maple code 2.2.5.

In Maple, the Euler Totient function is simply **phi(n)**. To compute $\phi(n)$, maple needs to factor n. Consequently, maple is not useful for computing $\phi(n)$ for large n.

```
      Maple: Euler totient function

aa:=2^300*3^200;
phi(aa);

aa:=nextprime(2^300)*nextprime(3^200);
phi(aa);
```

2.3 Principe de RSA

2.3.1 Le module de RSA

Below we describe in detail the initial schemes of the RSA Cryptosystem.

- **RSA Key Generation**
  INPUT: The bitsize k of the modulus.
  OUTPUT: A public key (N, e) and a private key (N, d).

  1. Generate two large random and distinct $(k/2)$-bit primes p and q.
  2. Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$.
  3. Choose a random integer e such that $3 \leq e < \phi(N)$ and $\gcd(e, \phi(N)) = 1$.
  4. Compute the unique integer d such that $1 \leq e < \phi(N)$ and $ed \equiv 1 \pmod{\phi(N)}$.
  5. Return the public key (N, e) and the private key (N, d).

- **RSA Encryption**
  INPUT: The public key (N, e) and the plaintext $\mathbf{m}$.
  OUTPUT: The ciphertext C.

  1. Represent the message $\mathbf{m}$ as an integer M with $1 \leq M \leq N-1$.
  2. Compute $C \equiv M^e \pmod{N}$.
  3. Return the ciphertext C.

- **RSA Decryption**
  INPUT: The private key (N, d) and the the ciphertext C.
  OUTPUT: The message $\mathbf{m}$.

1. Compute $M \equiv C^d \pmod{N}$.
2. Transform the number M to the message **m**.
3. Return the message **m**.

Algorithm 2 : Fabrication du module RSA

Input : Une taille $2t$ pour le module du cryptosytème RSA.
Output : Deux nombres premiers de tailles t et un module RSA N de taille $2t$.

1: Prendre un nombre premier aléatoire p dans l'intervalle $[2^t, 2^{t+1}]$.
2: Prendre un nombre premier aléatoire q dans l'intervalle $[2^t, 2^{t+1}]$.
3: **If** $p = q$ **then**
4: Aller à l'étape 2.
5: **Else**
6: $N = pq$.
7: **End If**

Exercice 2.3.1. 1. Ecrire une procédure pour produire deux nombres premiers p et q ayant chacun t bits avec $q < p$ ainsi qu'un module RSA $N = pq$.

2. Donner un module RSA de 1024 bits.

Maple code 2.3.2.

```
      Programme

rsa:=proc(t)
local m1,m2,p,q,N;
m1:=2^(t);m2:=2^(t+1);
p:=nextprime(rand(m1..m2)());
q:=nextprime(rand(m1..m2)());
while p=q or p>m2 or q>m2 do
    p:=nextprime(rand(m1..m2)());
    q:=nextprime(rand(m1..m2)());
end do;
N:=p*q;
return min(p,q),max(p,q),N;
end proc:
```

```
      Programme

t:=512:
l:=rsa(t);
p:=l[1];q:=l[2];N:=l[3];
```

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

Maple code 2.3.3.

Programme	Commentaires
`with(numtheory):`	`<- Librairie "Théorie des Nombres"`
`N:=5*8*61:`	`<- un entier simple`
`phi(N);`	`<- calcul de phi(n)`
`N:=nextprime(10^26)*nextprime(10^27):`	
`st := time():phi(N);duree := time()-st;`	

2.3.2 Les clés publiques et privées

Algorithm 3 : Fabrication des clés e et d

Input : Deux nombres premiers p et q.
Output : Une une clé publique e et clé privée d.

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

Exercice 2.3.4.
1. Pour p et q donnés, écrire une procédure pour produire un une clé publique e telle que $\gcd(e, \phi(N)) = 1$ et son inverse d modulo $\phi(N)$.
2. Donner une clé publique et une clé privée pour un module RSA de 1024 bits.

Maple code 2.3.5.

Programme
`cles:=proc(p,q)`
`local N, phii,e,d;`
`phii:=(p-1)*(q-1):`
`e:=rand(3..phii-1)():`
`while(gcd(e,phii) <> 1) do`
`  e:=rand(3..phii)();`
`od:`
`d := 1/e mod  phii:`
`return e,d;`
`end proc:`

```
      Programme

l:=rsa(512):p:=l[1]:q:=l[2]:N:=l[3]:
l:=cles(p,q);
e:=l[1];d:=l[2];
e*d mod (p-1)*(q-1);
```

2.3.3 Envoi d'un message

Algorithm 4 : Chiffrement d'un message

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

Maple code 2.3.6.

Pogramme de chiffrement.

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

2.3.4 Déchiffrement d'un message

Algorithm 5 : Déchiffrement d'un message

Input : Un message chiffré C et la clé privée (N, d).
Output : Un message clair M.
1: Calculer $M \equiv C^d \pmod{N}$.
2: Transformer le nombre M en un message clair.

Maple code 2.3.7.

Pogramme déchiffrement.

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

2.3.5 Signature d'un message

Algorithm 6 : Signature d'un message

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

Output : 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 code 2.3.8.

Programme Signature

```
      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
```

Algorithm 7 : Vérification d'une signature

Input : Un message chiffré C, une signature S, la clé publique (N_A, e_A).

Output : Vérification d'une signature.

1: B calcule $C' \equiv S^{e_A} \pmod{N_A}$.
2: Si $C' = C$ la signature est authentifiée.

Maple code 2.3.9.

Programme vérification

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	

2.3.6 Un exemple d'utilisation de RSA

Chinguetti est la perle de la Mauritanie

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 code 2.3.10.
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)**.

Maple code 2.3.11.

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"

Maple code 2.3.12.

Tout d'abord, on choisit p, q, N, $\phi(N)$, e et d.

```
      Programme

t:=20;
p:=nextprime(rand(2^t..2^(t+1))());
q:=nextprime(rand(2^t..2^(t+1))());
e:=17;
N:=p*q;
ph:=(p-1)*(q-1);
d:=1/e mod ph;
```

Pogramme de vérification du cryptosystème RSA.

```
      Programme: Encryption

m:="Chinguetti est la perle de la Mauritanie";
m:=convert(m,bytes);
c:=[];
for i from 1 to nops(m) do
    c:=[op(c),power(m[i],e) mod N];
end do:
c;
```

```
      Programme: Decryption


m2:=[];
for i from 1 to nops(c) do
    m2:=[op(m2),power(c[i],d) mod N];
end do:
m2;
m2-m;
```

Maple code 2.3.13.

Pogramme de vérification.

```
      Programme Obtenir une liste claire en bytes


message:="Chinguetti est la perle de la Mauritanie":
listeclaire:= convert(message, bytes):
```

```
      Programme Obtenir une liste cryptée


listecrypte:=[];
for i from 1 to nops(listeclaire) do
    listecrypte:=[op(listecrypte),power(listeclaire[i],e) mod N];
end do:
listecrypte;
```

```
      Programme déchiffrement de la liste

listdecrypte:=[];
for i from 1 to nops(listecrypte) do
        listdecrypte:=[op(listdecrypte),power(listecrypte[i],d) mod N];
end do:
listdecrypte;
```

```
      Programme obtenir le message original

messageoriginale:=convert(listedecrypte, bytes);
listeclaire-listedecrypte;
```

2.4 Cryptanalyses élémentaires de RSA

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

Proposition 2.4.1. *Soit N un module RSA. Si on connait $\varphi(N)$, alors on peut factoriser N.*

Maple code 2.4.2.

Programme de résolution

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

Exercice 2.4.3. 1. Démontrer la proposition.

2. Pogrammer cette proposition à l'aide de Maple et déterminer la valeur de p et q sachant que (ne pas utliser **ifactor**)

$$\begin{aligned} N &= 102724374910493563184689283607289992214 9, \\ \phi(N) &= 102724374910493563169138068486294327806 0. \end{aligned}$$

2.4.2 Utilisation du même module et deux exposants différents

Proposition 2.4.4. *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.*

Exercice 2.4.5. Démontrer cette proposition.

Maple code 2.4.6.

Programme	Commentaires
`N:=22900724415936720487705353 07:`	`<--- 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`

2.4.3 Utilisation de modules différents pour le même message.

Theorem 2.4.7. *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* $M_i \equiv P_i^{-1} \pmod{N_i}$.

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

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

Maple code 2.4.8.

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

Proposition 2.4.9. *Soit* $k \geq 2$ *un nombre entier. On considère* 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.*

Proof. 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 code 2.4.10.

Programme	Commentaires
`for i from 1 to 4 do`	`<--- 4 modules RSA`
`N[i]:=nextprime(rand())*nextprime(rand()):`	`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`

2.4.4 Attaque cyclique (Cycling Attack)

Dans cette partie, on considère RSA avec un module $N = pq$, une clé publique e et un message m. Soit c le message crypté:

$$c \equiv m^e \mod N.$$

Theorem 2.4.11. *Soit $N = pq$ un module RSA, e une clé publique et c un message crypté d'un message claire m inconnu. On suppose qu'on connait le plus petit entier k tel que*

$$\gcd\left(c^{e^k} - c, N\right) > 1.$$

Alors on peut déterminer le message m ou factoriser le module N.

Proof. On suppose que

$$\gcd\left(c^{e^k} - c, N\right) > 1.$$

On distingue alors deux cas:

- Soit

$$\gcd\left(c^{e^k} - c, N\right) = p, \qquad \text{ou} \qquad \gcd\left(c^{e^k} - c, N\right) = q,$$

ce qui donne la factorisation de N.

- Soit

$$\gcd\left(c^{e^k} - c, N\right) = N,$$

alors

$$c^{e^k} \equiv c \pmod N,$$

et en posant

$$m' \equiv c^{e^{k-1}} \pmod{N},$$

on obtient $m'^e \equiv c \pmod{N}$, donc $m = m'$ car k est le plus petit entier vérifiant $c^{e^k} \equiv c \pmod{N}$. □

Maple code 2.4.12.

Programme	Commentaires

```
Digits := 100:
p := nextprime(round(7.5^12));
q := nextprime(round(7.5^11.8));
N := p*q:
e:=3;
c := 175406449694505885461:
x0:=c;
x:=modp(x0&^e,N);
g:=gcd(x-c,N);
while g =1 do
    x0:=x;
    x:=modp(x0&^e,N);
    g:=gcd(x-c,N);
end do;
if g<N then
    print('Un facteur premier de N est', g):
    else print('Le message clair est', x0):
end if:
```

2.4.5 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.

Lemma 2.4.13. *Let $N = pq$ be an RSA modulus with $|p-q| < cN^{1/4}$ where c is a positive constant. Then one can factor N in time polynomial in c.*

Proof. Fermat's method for factoring $N = pq$ consists in finding two integers x, y such that

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

If $x - y \neq 2$, then the factorization of N is given by

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

Observe that, if $p - q < c\sqrt{N}$, then $p \approx q \approx \sqrt{N}$ and $p + q \approx 2\sqrt{N}$. To find x, y, we consider the sequence of candidates for x and y defined by

$$x_i = \left\lfloor 2\sqrt{N} \right\rfloor + 1 + i, \quad \text{and} \quad y_i = \sqrt{x_i^2 - 4N}, \quad i = 0, 1, \cdots, k,$$

where $\lfloor x \rfloor$ is the integral part of x. We stop the process when $x_k^2 - 4N$ is a perfect square. Since $p = \frac{x_k + y_k}{2}$ and $q = \frac{x_k - y_k}{2}$, then $x_k = p + q$. Now, suppose that $|p - q| < cN^{1/4}$. Then, since $2\sqrt{N} \leq \left\lfloor 2\sqrt{N} \right\rfloor + 1$ and $p + q > 2\sqrt{N}$, we get

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

It follows that Fermat's method can factor N in less than $\frac{c^2}{4}$ steps, which is efficient for small values of c. □

Maple code 2.4.14.

```
            Programme

fermat:=proc(N)
local i,x,y,p,q;
x:=trunc(2*sqrt(N));
y:=sqrt(x^2-4*N);
while y<>round(y) do
    x:=x+1;
    y:=sqrt(x^2-4*N);
end do;
p:=(x+y)/2;
q:=(x-y)/2;
return p,q;
end proc;

fermat(43361741);
fermat(17090864954662304167505112728173352240867);
fermat(17772281672690136564678521442710498584272041857428470131 1);
```

Maple code 2.4.15.

A complete example

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:`	

2.4.6 Cryptanalyse de RSA en utilisant deux messages et un polynôme

Exercice 2.4.16. Démontrer le théorème suivant:

Theorem 2.4.17. *Soient $N = pq$ un module RSA, e une clé publique et f un polynôme. Soit M un message satisfaisant*

$$\begin{aligned} C_1 &\equiv M^e \pmod N, \\ C_2 &\equiv (f(M))^e \pmod N, \\ x - M &\equiv pgcd(x^e - C_1, (f(x))^e - C_2) \mod N. \end{aligned}$$

Connaissant N, e, f, C_1 et C_2, on peut déterminer M.

2) Pogrammer ce théorème à l'aide de Maple et déterminer la valeur de M sachant que

$$\begin{aligned} N &= 22900724415936720487705353 07, \\ e &= 7, \\ f(x) &= x^2 - 751, \\ C_1 &= 50898847680285003503506 6764, \\ C_2 &= 13247370961014078164695 6648. \end{aligned}$$

Chapter 3

Cryptanalyse de RSA par les Fractions Continues

3.1 Les fractions continues

3.1.1 Définitions et propriétés

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 $x = 1 + 7^{\frac{1}{3}}$.

Maple code 3.1.1.

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,....]`

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 $x = 1 + 7^{\frac{1}{3}}$.

Maple code 3.1.2.

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

Theorem 3.1.3. *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 $n \geq -2$*

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

Maple code 3.1.4.

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):	
long:=100:	
s:=cfrac(x,long,'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)

Programme	Commentaires

`for n from -2 to long-1 do`
`  r := nu(n)*de(n+1)-nu(n+1)*de(n);`
`  dif := (-1)^(n+1)-r;`
`  print(n, dif):`
`end do:`	

Corollary 3.1.5. *Les convergentes* $\frac{p_n}{q_n}$ *d'un nombre réel* x *vérifient* $pgcd(p_n, q_n) = 1$ *pour tout* $n \geq 0$.

Maple code 3.1.6.

Programme	Commentaires

`for n from 0 to long-1 do`
`  g := gcd(nu(n), de(n)):`
`  print(n, g):`
`end do:`	

Theorem 3.1.7. *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 :*

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

Maple code 3.1.8.

Programme	Commentaires

`for n from 0 to trunc((1/2)*long) do`
`  dif1:= nu(2*n-2)/de(2*n-2)-nu(2*n)/de(2*n):`
`  print(n, sign(dif1)):`
`end do:`
`for n from 0 to trunc((1/2)*long) do`
`  dif2 := nu(2*n)/de(2*n)-evalf(x):`
`  print(n, sign(dif2)):`
`end do:`
`for n from 0 to trunc((1/2)*long)-1 do`
`  dif3 := evalf(x)-nu(2*n+1)/de(2*n+1):`
`  print(n, sign(dif3)):`
`end do:`
`for n from 1 to trunc((1/2)*long)-1 do`
`  dif4:=nu(2*n+1)/de(2*n+1)-nu(2*n-1)/de(2*n-1):`
`  print(n, sign(dif4)):`
`end do`	

Corollary 3.1.9. *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 $n \geq 0$, $(q_n x - p_n)(q_{n+1}x - p_{n+1}) < 0$.*
2. *Pour tout $n \geq 0$, $|q_{n+1}x - p_{n+1}| < |q_n x - p_n|$.*

Maple code 3.1.10.

Programme	Commentaires

```
for n from 0 to long-1 do
    dif1 := de(n)*evalf(x)-nu(n);
    dif2 := de(n+1)*evalf(x)-nu(n+1);
    print(n, sign(dif1)*sign(dif2)):
end do:
for n from 0 to long-1 do
    dif3 := abs(de(n)*evalf(x)-nu(n)):
    dif4 := abs(de(n+1)*evalf(x)-nu(n+1)):
    dif := dif4-dif3:
    print(n, sign(dif)):
end do:
```

Proposition 3.1.11. *Si $x = [a_1, a_2, a_3, \cdots]$, alors les convergentes $\frac{p_n}{q_n}$ de x vérifient pour tout $n \geq 0$,*

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

Maple code 3.1.12.

Programme	Commentaires

```
for n from 0 to long-1 do
    dif := abs(evalf(x)-nu(n)/de(n))-1/(de(n)*de(n+1)):
    print(n, sign(dif)):
end do:
```

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

Theorem 3.1.13 (Meilleurs approximations)**.** *Soit $[a_1, a_2, a_3, \cdots]$ 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 $\left| x - \frac{p_n}{q_n} \right| \leq \left| x - \frac{p}{q} \right|$.*

Proof. 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 3.1.3. 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. □

Proposition 3.1.14. *Si $[a_0, a_1, a_2, \cdots]$ est la fraction continue d'un nombre x, alors pour $n \geq 0$, on a*

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

Maple code 3.1.15.

Programme	Commentaires

```
for n from 0 to long-1 do
    dif1:=abs(evalf(x)-nu(n)/de(n))-1/(2*de(n)^2):
    dif2:=abs(evalf(x)-nu(n+1)/de(n+1))-1/(2*de(n+1)^2):
    print(n,sign(dif1),sign(dif2)):
    if sign(dif1)+sign(dif2)>1 then
        print(FAUX)
    end if:
end do:
```

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

Theorem 3.1.16. *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.

Proof. 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 3.1.13 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

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

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

3.2 Cryptanalyse de RSA par les fractions continues

3.2.1 L'attaque de Wiener

Proposition 3.2.1. *Soit $N = pq$ un module RSA où les nombres premiers p et q vérifient $q < p < 2q$. Alors*

$$\frac{\sqrt{2}}{2}\sqrt{N} < q < \sqrt{N} < p < \sqrt{2}\sqrt{N}.$$

Exercice 3.2.2. Démontrer la proposition.

Proposition 3.2.3. *Soit $N = pq$ un module RSA où les nombres premiers p et q vérifient $q < p < 2q$. Alors*

$$2\sqrt{N} < p + q < \frac{3\sqrt{2}}{2}\sqrt{N}.$$

Exercice 3.2.4.

1. Démontrer la proposition.
2. Donner un encadrement de $\phi(N)$ en fonction de N.

Theorem 3.2.5. *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.*

Proof. 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 3.1.16, $\frac{k}{d}$ 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 2.4.1 on peut calculer p et q et donc trouver la factorisatio de N. □

Exercice 3.2.6.

1. Write a maple procedure wiener(N, e) that computes $k, d, \phi(N), p, q$ when $ed - k\phi(N) = 1$ and $d < \frac{1/3}{N}^{\frac{1}{4}}$.
2. Find the factorization of $N = 1235882707637$ with $e = 434466294109$ using wiener(N, e).

Maple code 3.2.7.

```
        Programme

wiener:=proc(N,e)
local cv,i,k,d,phin,q,delta;
cfrac(e/N,10,'cv'):
for i from 2 to nops(cv) do
    k:=numer(cv[i]);
    d:=denom(cv[i]);
    phin:=(e*d-1)/k;
    if round(phin)=phin then
        delta:=(phin-N-1)^2-4*N;
        if delta>0 then
            q:=(-(phin-N-1)-sqrt(delta))/2;
            if round(q)=q then
                return q,N/q,phin,d,k;
            end if;
        end if;
    end if;
end do;
end proc:

N:=1235882707637;
e:=434466294109;
wiener(n,e);
```

Maple code 3.2.8.

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=‘):p2;	<-- Affichage de p
print(‘q=‘):N/p2;	<-- Affichage de q

Exercice 3.2.9. Démontrer le théorème suivant:

Theorem 3.2.10. *Soit d un entier positif, qui n'est pas un carré parfait. Si x et y sont deux nombres entiers positifs qui vérifient l'équation :*

$$x^2 - dy^2 = 1,$$

alors $\frac{x}{y}$ est une convergente de $\sqrt{d}$.

Exercice 3.2.11. Programmer ce théorème et déterminer une solution (x, y) lorsque $d = 1237$.

Exercice 3.2.12. Démontrer le théorème suivant :

Theorem 3.2.13. *Soit a, b et N des entiers positifs. Si x et y sont deux entiers positifs qui vérifient $pgcd(x, y) = 1$ et*

$$ax^2 + by^2 = N,$$

alors il existe une solution t de la congruence $at^2 + b \equiv 0 \pmod{N}$ et un entier u tel $x = Nu - ty$. De plus, $\frac{u}{y}$ est une convergente de $\frac{t}{N}$.

Exercice 3.2.14. Programmer ce théorème et déterminer une solution (x, y) lorsque

$$a = 127, \qquad b = 1547, \qquad N = 324655759005538.$$

sachant que les solutions de la congruence $at^2 + b \equiv 0 \pmod{N}$ s'obtiennet en effectuant **msolve**$(a * t^2 + b = 0, N)$.

Chapter 4

Réductions des Réseaux

4.1 Introduction aux Réseaux

L'algorithme LLL a été inventé en 1982 et porte les initiales de 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.

Definition 4.1.1. 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.

If $\mathcal{L} \subset \mathbb{R}^n$ is a lattice of dimension d, then it is a discrete additive subgroup of $\mathbb{R}^n$.

Proposition 4.1.2. *Let $\mathcal{L}$ be a lattice of dimension d and rank n. Then $\mathcal{L}$ can be written as the rows of an $d \times n$ matrix with real entries.*

Proof. Let $(b_1 \cdots, b_d)$ be a basis of $\mathcal{L}$ such that, for $1 \leq i \leq d$,

$$b_i = \begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{ni} \end{bmatrix}.$$

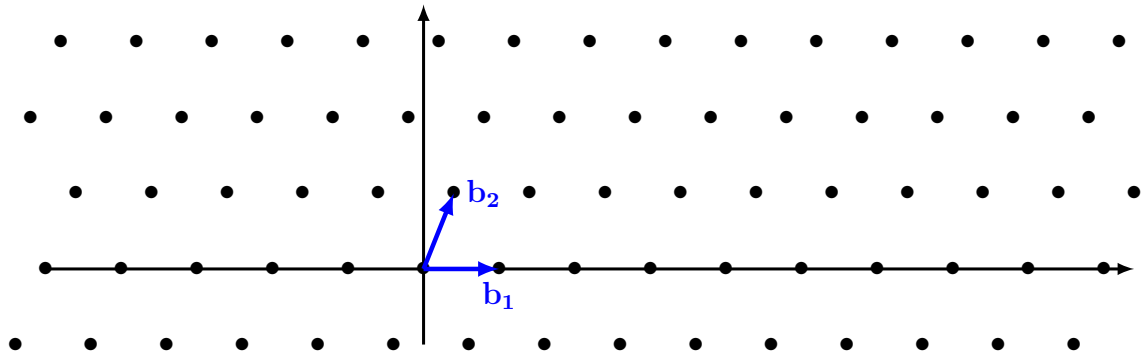


Figure 4.1: A lattice with the basis (b_1, b_2)

Let v be a vector of $\mathcal{L}$. Then $v = \sum_{i=1}^{d} x_i b_i$ for $x_i \in \mathbb{Z}$. Hence v can be rewritten as

$$v = x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{n2} \end{bmatrix} + \ldots + x_d \begin{bmatrix} a_{1d} \\ a_{2d} \\ \vdots \\ a_{nd} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nd} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}.$$

The involved matrix is constructed using the coordinates of the basis $(b_1 \cdots, b_d)$ as follows

$$\begin{bmatrix} b_1 & b_2 & \cdots & b_d \\ \downarrow & \downarrow & \cdots & \downarrow \\ a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nd} \end{bmatrix}.$$

□

The following result shows that in a lattice $\mathcal{L}$ with dimension $d \geq 2$, any two couples of bases are related with a unimodular matrix.

Proposition 4.1.3. *Let $\mathcal{L} \subset \mathbb{R}^n$ be a lattice of dimension d. Let $(b_1 \cdots, b_d)$ and $(b'_1 \cdots, b'_d)$ be two bases of $\mathcal{L}$. Then there exists a $d \times d$ matrix U with entries in $\mathbb{Z}$ and $\det(U) = \pm 1$ such that*

$$\begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_d \end{bmatrix} = U \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_d \end{bmatrix}.$$

Proof. Let $(b_1 \cdots, b_d)$ and $(b'_1 \cdots, b'_d)$ be two bases of $\mathcal{L}$. Since every vector $b'_i \in \mathcal{L}$, then

$$\begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_d \end{bmatrix} = \begin{bmatrix} u_{11}b_1 + u_{12}b_2 + \ldots + u_{1d}b_d \\ u_{21}b_1 + u_{22}b_2 + \ldots + u_{2d}b_d \\ \vdots \\ u_{d1}b_1 + u_{d2}b_2 + \ldots + u_{dd}b_d \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1d} \\ u_{21} & u_{22} & \cdots & u_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ u_{d1} & u_{d2} & \cdots & u_{dd} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_d \end{bmatrix} = U \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_d \end{bmatrix},$$

where U is a $d \times d$ matrix with integer entries. This can be rewritten as

$$(b'_1, b'_2, \ldots, b'_d)^t = U(b_1, b_2, \ldots, b_d)^t.$$

Similarly, there exist a $d \times d$ matrix U' with integer entries such that

$$(b_1, b_2, \ldots, b_d)^t = U'(b'_1, b'_2, \ldots, b'_d)^t.$$

Hence,

$$(b'_1, b'_2, \ldots, b'_d)^t = UU'(b'_1, b'_2, \ldots, b'_d)^t,$$

which implies $UU' = I_d$ where I_d is the $d \times d$ identity matrix. Taking determinant, we get $\det(U)\det(U') = 1$. Since the entries of U and U' are integers, then $\det(U), \det(U') \in \mathbb{Z}$ and $\det(U) = \det(U') = \pm 1$. □

Observe that any $d \times d$ triangular matrix U with diagonal entries equal to ± 1 satisfies $\det(U) = \pm 1$. This shows that a lattice $\mathcal{L}$ with dimension $d \geq 2$ has infinitely many bases.

Definition 4.1.4. Let $\mathcal{L}$ be a lattice with a basis $(b_1 \cdots, b_d)$. The volume or determinant of $\mathcal{L}$ is

$$\det(\mathcal{L}) = \sqrt{\det(BB^t)},$$

where B is the $d \times n$ matrix of formed by the rows of the basis.

Proposition 4.1.5. *Let $\mathcal{L}$ be a lattice of dimension d. Then the* $\det(\mathcal{L})$ *is independent of the choice of the basis.*

Proof. Let $(b_1 \cdots, b_d)$ and $(b'_1 \cdots, b'_d)$ be two bases of $\mathcal{L}$ with matrices B and B'. Then there exists a $d \times d$ matrix U with entries in $\mathbb{Z}$ and $\det(U) = \pm 1$ such that $B' = UB$. Then since $B'B'^t = UBB^tU^t$, we get

$$\det(B'B'^t) = \det\left(UBB^tU^t\right) = \det(U)\det\left(BB^t\right)\det\left(U^t\right) = \det\left(BB^t\right),$$

where we used $\det(UU^t) = \det(U)^2 = 1$. Hence $\sqrt{\det(B'B'^t)} = \sqrt{\det(BB^t)} = \det(\mathcal{L})$. □

When $d = n$, that is L is a full-rank lattice, the matrix of the basis is a $n \times n$ matrix and the following property holds.

Lemma 4.1.6. *Let $\mathcal{L}$ be a full-rank lattice of dimension n. If $(b_1 \cdots, b_n)$ is a basis of $\mathcal{L}$ with matrix B, then*

$$\det(L) = |\det(B)|.$$

Proof. Since $\det(B^t) = \det(B)$, then

$$\det(\mathcal{L}) = \sqrt{\det(BB^t)} = \sqrt{\det(B)\det(B^t)} = \sqrt{\det(B)^2} = |\det(B)|.$$

□

The determinant of a lattice can be considered as the volume of its fundamental domain.

Definition 4.1.7. Let $\mathcal{L}$ be a lattice with a basis $(b_1 \cdots, b_d)$. The fundamental domain or parallelepipede for $\mathcal{L}$ is the set

$$\mathcal{P}(b_1 \cdots, b_d) = \left\{ \sum_{i=1}^{d} x_i b_i, \ | \ 0 \leq x_i < 1 \right\}.$$

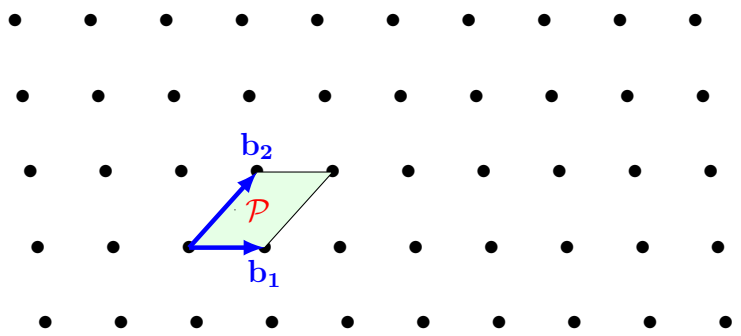


Figure 4.2: The fundamental domain for the basis (b_1, b_2)

Proposition 4.1.8. *Let $\mathcal{L}$ be a lattice with a basis $(b_1, \ldots, b_d)$. Then the volume $\mathcal{V}$ of the fundamental domain $\mathcal{P}(b_1, \ldots, b_d)$ satisfies*

$$\mathcal{V}(\mathcal{P}(b_1, \ldots, b_d)) = \det(\mathcal{L}).$$

The former result shows that any two bases of a lattice have the same volume $\mathcal{V}$ of the fundamental domain. This shows again that $\det(\mathcal{L})$ is an important invariant in a lattice.

When $d = n$, that is L is a full-rank lattice, the matrix of the basis is a $n \times n$ matrix and the following property holds. Lattices whose bases have integer coordinates are very convenient for various problems. Such lattices are called *integral lattices*. Also, many problems in lattice theory involve inner product of vectors and distance minimization and the most intuitive way to measure distance in a lattice is by using the Euclidean norm.

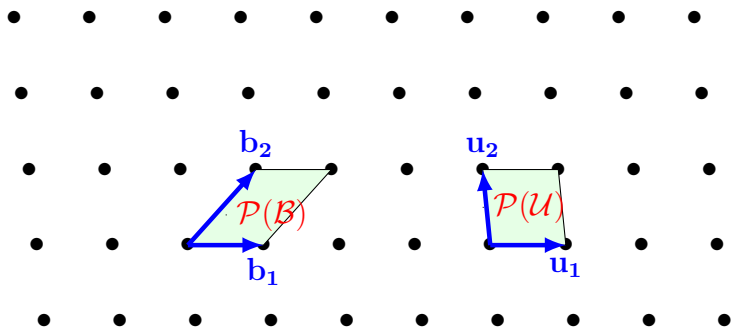


Figure 4.3: The fundamental domain for the bases (b_1, b_2) and (u_1, u_2)

4.2 The Gram-Schmidt Orthogonalization

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, la plus orthogonale possible dans un réseau. Pour cela, on considère le produit scalaire habituel de deux vecteurs et la norme euclidienne d'un vecteur.

Definition 4.2.1. 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

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

Exercice 4.2.2. 1. Programmer le produit scalaire à l'aide de Maple.

2. Calculer $\langle u, v \rangle$ avec $u = (1, 2, 3, 4)$ et $v = (-1, -2, -3, -4)$.

Maple code 4.2.3.

```
      maple: Produit Scalaire

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

```
      Maple: Exemple Produit Scalaire

u := [1, 2, 3, 4]; v := [-1, -2, -3, -4];
scal(u, v);
```

Dans Maple, le produit scalaire est la fonction **DotProduct(u,v)**.

Maple code 4.2.4.

```
      Maple: Exemple Produit Scalaire

with(LinearAlgebra):
u := [1, 2, 3, 4]; v := [-1, -2, -3, -4];
DotProduct(u, v);
```

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

Theorem 4.2.5 (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*

$$b_1^* = b_1, \quad \text{et pour } i \geq 2, \quad b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*,$$

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.

Proof. 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

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

La matrice U est est une matrice triangulaire dont la diagonale est formée de 1. Donc $\det(U) = 1$. 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 $1 \leq k \leq i-1$

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

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

Exercice 4.2.6. Orthogonaliser les vecteurs suivants :

1. $b_1 = (3, 1)$, $b_2 = (1, 2)$.

2. $b_1 = (3, 2, 5)$, $b_2 = (2, 4, -1)$, $b_3 = (-2, -1, 6)$.

Maple code 4.2.7.

```
      Maple: Gram-Schmidt

with(LinearAlgebra):
b1 := Vector([3, 1]);
b2 := Vector([1, 2]);
bb1 := b1;
u21 := DotProduct(b2, b1)/DotProduct(b1, b1);
bb2 := -b1*u21+b2;
```

Maple code 4.2.8.

```
      Maple: Gram-Schmidt

with(LinearAlgebra):
b1 := Vector([3, 2, 5]);
b2 := Vector([2, 4, -1]);
b3:=Vector([-2, -1, 6]);
bb1 := b1;
u21 := DotProduct(b2, b1)/DotProduct(b1, b1);
bb2 := -b1*u21+b2;
u31 := DotProduct(b3,bb1)/DotProduct(bb1, bb1);
u32 :=DotProduct(b3,bb2)/DotProduct(bb2, bb2);
bb3 := -bb1*u31-bb2*u32+b3;
DotProduct(bb1, bb2);
DotProduct(bb1, bb3);
DotProduct(bb2, bb3);
```

La méthode de Gram-Schmidt, décrite dans Theorem 8 peut être facilement mise en algorithme, comme dans l'algorithme 8.

Algorithm 8 : La méthode de Gram-Schmidt

Input : Une base $(b_1 \cdots, b_n)$.
Output : Une base orthogonale $(b_1^* \cdots, b_n^*)$.
1: Poser $b_1^* = b_1$.
2: **For** $i = 1, 2, \cdots n$, **do**
3: **For** $j = 1, 2, \cdots i-1$, **do**
4: Calculer $\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}$.
5: **End For**
6: Calculer $b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$.
7: **End For**

Example 4.2.9.

1. Programmer la méthode de Gram-Schmidt avec pour entrée une matrice carrée.

2. Orthogonaliser la base (b_1, b_2, b_3) avec $b_1 = (3, 2, 5)$, $b_2 = (2, 4, -1)$, $b_3 = (-2, -1, 6)$.

```
      La méthode de Gram-Schmidt

gramschmidt:=proc(B)
local i, j, n, M, P, u, k;
with(LinearAlgebra);
n:=nops(B);
for i to n do
    M[i]:=B[i];
    for j to i-1 do
        u[j]:= DotProduct(B[i],M[j])/DotProduct(M[j],M[j]);
        M[i]:= M[i]-M[j]*u[j];
    end do;
end do;
M :=seq(M[i], i = 1 .. n);
return M;
end proc:
```

Le programme suivant prend en entrée la base (b_1, b_2, b_3) dont les vecteurs sont $b_1 = (3, 2, 5)$, $b_2 = (2, 4, -1)$, $b_3 = (-2, -1, 6)$ et sort la base orthogonale $b_1^* = (3, 2, 5)$, $b_2^* = \left(\frac{49}{38}, \frac{67}{19}, -\frac{83}{38}\right)$, $b_3^* = \left(-\frac{1738}{717}, \frac{1027}{717}, \frac{632}{717}\right)$.

Maple code 4.2.10.

```
      La méthode de Gram-Schmidt

b1 :=[3, 2, 5];
b2 :=[2, 4, -1];
b3:=[-2, -1, 6];
B:=[b1,b2,b3];
gramschmidt(B);
```

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 $b_1 = (3, 2, 5)$, $b_2 = (2, 4, -1)$, $b_3 = (-2, -1, 6)$.

Programme	Commentaires
`with(LinearAlgebra):`	`<--- package LinearAlgebra`
`b1:=Vector([3,2,5]):`	`<--- Premier vecteur`
`b2 := Vector([2,4,-1]):`	`<--- Deuxième vecteur`
`b3 := Vector([-2,-1,6]):`	`<--- 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`

La base $(b_1^*, \cdots, b_n^*)$ a plusieurs propriétés remarquables, comme dans l'exemple ci-suivant.

Lemma 4.2.11. *Soient $(b_1 \cdots, b_n)$ une famille indépendante de vecteurs. On considère la famille de vecteurs $(b_1^* \cdots, b_n^*)$ produite par l'orthogonalisation de Gram-Schmidt. Alors*

1. $\|b_i^*\| \leq \|b_i\|$ *pour* $1 \leq i \leq n$.
2. $\langle b_i, b_i^* \rangle = \langle b_i^*, b_i^* \rangle$ *pour* $1 \leq i \leq n$.

Proof. 1. On a $\|b_1^*\| = \|b_1\|$ et pour $2 \leq i \leq n$,

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

avec pour $j < i$

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

Alors, puisque les vecteurs $b_1, \ldots, b_i$ sont deux à deux orthogonaux, on obtient

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

Ainsi $\|b_i^*\| \leq \|b_i\|$.

2. On a $\langle b_1, b_1^* \rangle = \langle b_1^*, b_1^* \rangle$ et pour $2 \leq i \leq n$,

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

□

Corollary 4.2.12 (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*

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

Proof. Par définition, on sait que $\det(L) = |\det(b_1 \cdots, b_n)|$. On sait aussi par Théorème 4.2.5 que $(b_1 \cdots, b_n)^t = U(b_1^*, \cdots, b_n^*)^t$ où U est une matrice triangulaire inférieure avec $\det(U) = 1$. Alors, puisque les vecteurs $b_1^*, \cdots, b_n^*$, sont orthogonaux deux à deux et $\|b_i^*\| \leq \|b_i\|$ pour $1 \leq i \leq n$, on obtient:

$$\det(L) = \det(U) |\det(b_1^* \cdots, b_n^*)| = \prod_{i=1}^{n} \|b_i^*\| \leq \prod_{i=1}^{n} \|b_i\|.$$

□

Exercice 4.2.13. On considère le réseau L engendré par la base $(b_1, \ldots, b_n)$. Ecrire une procédure Maple qui calcule $\det(L)$.

```
      Maple: procédure det

with(LinearAlgebra)
deter:=proc(B);
local GS,n,d,i;
GS:= GramSchmidt(B);
n:=nops(B);
d:=1;
for i from 1 to n do
    d:=d*sqrt(DotProduct(GS[i],GS[i]));
end do;
return d;
end proc:
```

Exercice 4.2.14. On considère le réseau L engendré par la base (b_1, b_2, b_3) avec $b_1 = (3, 2, 5)$, $b_2 = (2, 4, -1)$, $b_3 = (-2, -1, 6)$.

1. Calculer $\det(L)$.

2. Comparer $\det(L)$ et $\prod_{i=1}^{3} \|b_i\|$.

Maple code 4.2.15.

```
      Maple: exemple

b1:=Vector([3,2,5]):
b2 := Vector([2,4,-1]):
b3 := Vector([-2,-1,6]):
B := [b1, b2, b3]:
d1:= simplify(deter(B));
d2:=Determinant(Matrix(B));
d3:=1;
for i from 1 to 3 do
    d3:=d3*sqrt(DotProduct(B[i],B[i]));
end do:
d3;
```

4.3 Les vecteurs courts d'un réseau

Lattices are used as a fundamental tool for cryptanalysis of various public key cryptosystems such as knapsack cryptosystems, RSA, NTRU and GGH. On the other hand, lattices are used as a theoretical tool for security analysis of several cryptosystems such as NTRU and LWE. These cryptosystems are related to hard computational problems in the theory of lattices such shortest nonzero vectors and minimal distances.

Definition 4.3.1. Let L be a lattice. The minimal distance λ_1 of $\mathcal{L}$ is the length of the shortest nonzero vector of $\mathcal{L}$:

$$\lambda_1 = \inf\{\|v\| \in \mathcal{L} \mid v \in \mathcal{L}\backslash\{0\}\} = \inf\{\|v-u\| \in \mathcal{L} \mid v, u \in \mathcal{L},\ v \neq u\}.$$

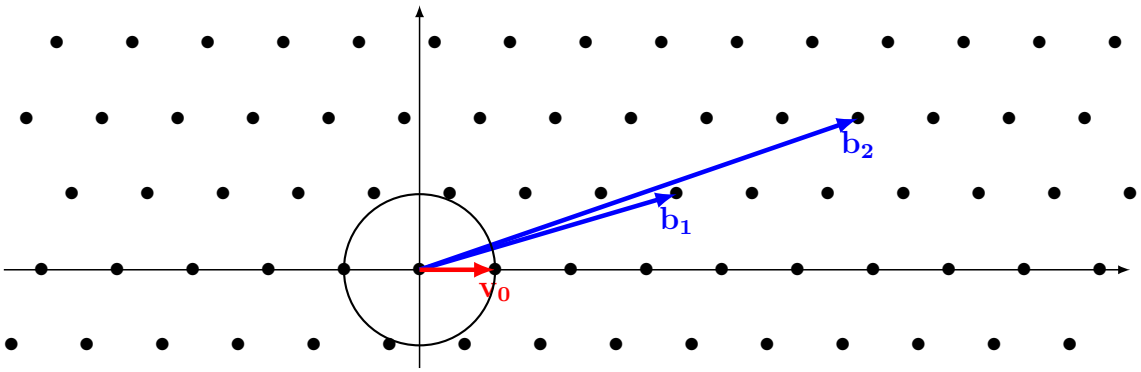


Figure 4.4: The shortest vectors are v_0 and $-v_0$

Example 4.3.2.

In dimension 2, finding the shortest vector in a lattice is very easy. Let $\mathcal{L}$ be a lattice with a basis (b_1, b_2) with

$$b_1 = \begin{bmatrix} 19239 \\ 2971 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 22961 \\ 3546 \end{bmatrix}.$$

Then the shortest vector is in the form

$$v_0 = x_1 b_1 + x_2 b_2 = \begin{bmatrix} 19239x_1 + 22961x_2 \\ 2971x_1 + 3546x_2 \end{bmatrix},$$

for some integers $(x_1, x_2) \neq (0, 0)$ for which the norm

$$\|v_0\| = \left((19239x_1 + 22961x_2)^2 + (2971x_1 + 3546x_2)^2\right)^{1/2},$$

is as small as possible. We write this as $v_0 \approx 0$, that is

$$\begin{cases} 19239x_1 + 22961x_2 \approx 0 \\ 2971x_1 + 3546x_2 \approx 0 \end{cases}$$

Assume that $x_2 \neq 0$. Then

$$\frac{x_1}{x_2} \approx -\frac{22961}{19239}, \text{ and } \frac{x_1}{x_2} \approx -\frac{3546}{2971}.$$

This means that $\frac{x_1}{x_2}$ is a good approximation for $-\frac{22961}{19239}$ and $-\frac{3546}{2971}$, which can be found among their common convergents. Indeed, the common convergents are

$$-2, -1, -\frac{6}{5}, -\frac{31}{26}, -\frac{37}{31}.$$

Plugging the common convergents in $x_1b_1 + x_2b_2 = v_0$ and computing $\|v_0\|$, we get

$$\begin{aligned}
&(x_1, x_2) = (2, -1), \quad v_0 = [15517, 2396], \quad \|v_0\| = \sqrt{246518105}, \\
&(x_1, x_2) = (1, -1), \quad v_0 = [-3722, -575], \quad \|v_0\| = \sqrt{14183909}, \\
&(x_1, x_2) = (6, -5), \quad v_0 = [629, 96], \quad \|v_0\| = \sqrt{404857}, \\
&(x_1, x_2) = (31, -26), \quad v_0 = [-577, -95], \quad \|v_0\| = \sqrt{341954}, \\
&(x_1, x_2) = (37, -31), \quad v_0 = [52, 1], \quad \|v_0\| = \sqrt{2705}.
\end{aligned}$$

Using some extra algebra, one can show that $v_0 = 37b_1 - 31b_2$ is the shortest vector in the lattice $\mathcal{L}$.

Programme

```
with(numtheory):
m1:=-22961/19239:
cfrac(m1,20,'cv1'):cv1;
m1:=-3546/2971:
cfrac(m2,20,'cv2'):cv2;
```

Exercice 4.3.3. Let $\mathcal{L}$ be a lattice with a basis (b_1, b_2) with $b_1 = (412157, -14578)$, $b_2 = (21571, 695743)$. Find the shortest non-zero vector of $\mathcal{L}$.

Example 4.3.4.

In dimension $n \geq 3$, finding the shortest vector in a lattice is not so easy. For example, let $\mathcal{L}$ be a lattice with a basis (b_1, b_2, b_3) with

$$b_1 = \begin{bmatrix} 124797 \\ 2971 \\ 4781 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 95874 \\ 3546 \\ 7895 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 56871 \\ 35462 \\ 16539 \end{bmatrix}.$$

Then the shortest vector is in the form

$$v_0 = x_1b_1 + x_2b_2 + x_3b_3 = \begin{bmatrix} 124797x_1 + 95874x_2 + 56871x_3 \\ 2971x_1 + 3546x_2 + 35462x_3 \\ 4781x_1 + 7895x_2 + 16539x_3 \end{bmatrix},$$

for some integers $(x_1, x_2, x_3) \neq (0, 0, 0)$ for which the $\|v_0\|^2$ is as small as possible. We have

$$\begin{aligned}\|v_0\|^2 =&(124797x_1 + 95874x_2 + 56871x_3)^2 + (2971x_1 + 3546x_2 + 35462x_3)^2 \\ &+ (4781x_1 + 7895x_2 + 16539x_3)^2.\end{aligned}$$

As we will see below, finding the shortest vector in a lattice is very hard in general. The short vector in the lattice $\mathcal{L}$ is

$$v_0 = -3b_1 + 4b_2 = \begin{bmatrix} 9105 \\ 5271 \\ 17237 \end{bmatrix}.$$

Definition 4.3.5. Let L be a lattice of dimension n. For $i = 1, \dots n$, the ith successive minimum of the lattice is

$$\lambda_i = \min\{\max\{\|v_1\|, \dots, \|v_i\|\} \mid v_1, \dots, v_i \in \mathcal{L} \text{ are linearly independent}\}.$$

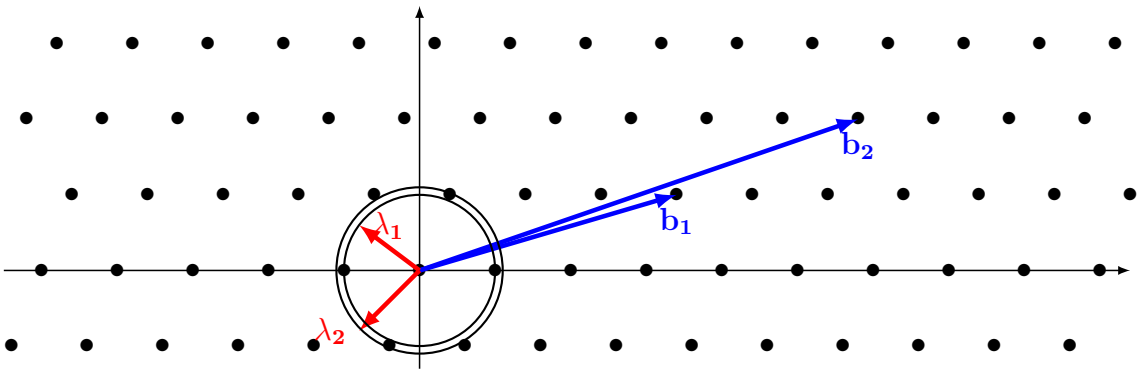


Figure 4.5: The first minima λ_1 and the second minima λ_2

Finding a vector v such that $\|v\| = \lambda_1$ is very hard in general. Nevertheless, in low dimension, the problem can be solved. For example, in dimension 2, Gauss'algorithm finds a basis (b_1, b_2) such that $\|b_1 = \lambda_1$ and $\|b_2\| = \lambda_2$ (see Subsection 4.3.1).

In the following, we list some computational problems that seem to be hard in general and on which some cryptographic systems have been based. An overview of many hard lattice problems and their interconnections is presented in [26].

Definition 4.3.6. Let $\mathcal{L}$ be a full rank lattice of dimension n in $\mathbb{Z}^n$.

1. **The Shortest Vector Problem (SVP):** Given a basis matrix B for $\mathcal{L}$, compute a non-zero vector $v \in \mathcal{L}$ such that $\|v\|$ is minimal, that is $\|v\| = \lambda_1(\mathcal{L})$.
2. **The Closest Vector Problem (CVP):** Given a basis matrix B for $\mathcal{L}$ and a vector $v \notin \mathcal{L}$, find a vector $u \in \mathcal{L}$ such that $\|v-u\|$ is minimal, that is $\|v-u\| = \mathrm{d}(v,\mathcal{L})$ where $\mathrm{d}(v,\mathcal{L}) = \min_{u\in\mathcal{L}} \|v-u\|$.
3. **The Shortest Independent Vectors Problem (SIVP):** Given a basis matrix B for $\mathcal{L}$, find n linearly independent lattice vectors $v_1, v_2, \ldots, v_n$ such that $\max_i \|v_i\| \leq \lambda_n$, where λ_n is the nth successive minima of $\mathcal{L}$.
4. **The approximate SVP problem (γSVP):** Fix $\gamma > 1$. Given a basis matrix B for $\mathcal{L}$, compute a non-zero vector $v \in \mathcal{L}$ such that $\|v\| \leq \gamma\lambda_1(\mathcal{L})$ where $\lambda_1(\mathcal{L})$ is the minimal Euclidean norm in $\mathcal{L}$.
5. **The approximate CVP problem (γSVP):** Fix $\gamma > 1$. Given a basis matrix B for $\mathcal{L}$ and a vector $v \notin \mathcal{L}$, find a vector $u \in \mathcal{L}$ such that $\|v-u\| \leq \gamma\lambda_1\mathrm{d}(v,\mathcal{L})$ where $\mathrm{d}(v,\mathcal{L}) = \min_{u\in\mathcal{L}} \|v-u\|$.

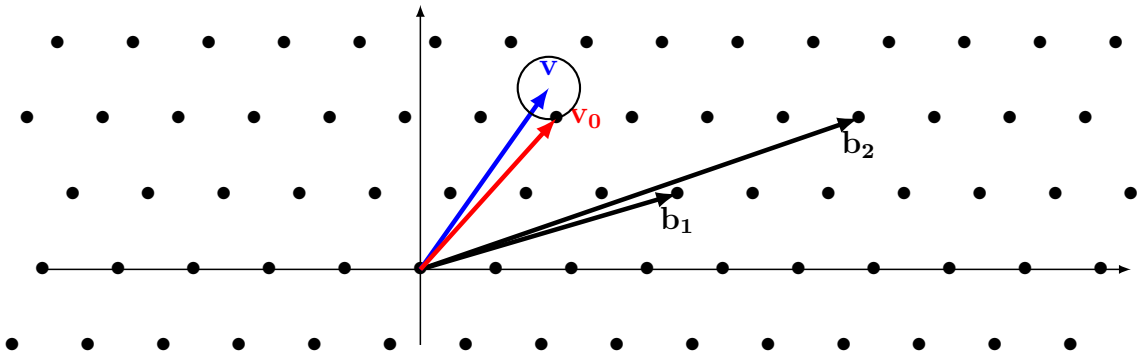


Figure 4.6: The closest vector to v is v_0

Some of such problems have been shown to be NP-hard, and in general, are known to be hard when the dimension is sufficiently large. No efficient algorithm is known to find the shortest vector nor the closest vector in a lattice. The next result, due to Minkowski gives a theoretical explicit upper bound in terms of $\dim(\mathcal{L})$ and $\det(\mathcal{L})$.

Theorem 4.3.7 (Minkowski)**.** *Let $\mathcal{L}$ be a lattice with dimension n. Then there exists a nonzero vector $v \in \mathcal{L}$ satisfying*

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

On the other hand, the Gaussian Heuristic implies that the expected shortest non-zero vector in a lattice $\mathcal{L}$ is approximately $\sigma(\mathcal{L})$ where

$$\sigma(\mathcal{L}) = \sqrt{\frac{\dim(\mathcal{L})}{2\pi e}} \left(\det(\mathcal{L})\right)^{\frac{1}{\dim(\mathcal{L})}}.$$

Exercice 4.3.8. On considère le réseau L muni de la base (b_1, b_2) avec $b_1 = (19239, 2971)$, $b_2 = (22961, 3546)$ et le vecteur le plus court $v_0 = 37b_1 - 31b_2$.

1. Vérifier l'inégalité de Minkowski.
2. Vérifier l'inégalité de l'heuristique de Gauss.

Maple code 4.3.9.

```
      Maple: Minkowski and Gauss heuristic

with(LinearAlgebra):
b1 := [19239, 2971];
b2 := [22961, 3546];
v0 := 37*b1-31*b2;
d:=Determinant(Matrix([b1, b2]));
Mink:=evalf(sqrt(DotProduct(v0,v0))/(sqrt(2)*d^(1/2)));
Gauss:=evalf(sqrt(DotProduct(v0,v0))/((sqrt(2/(2*Pi*exp(1)))*d^(1/2))));
```

4.3.1 Gauss'algorithm

Suppose that $\mathcal{L}$ is a lattice with a basis (b_1, b_2). Gauss'algorithm is used to find an optimal basis for $\mathcal{L}$ which is short and almost orthogonal. More precisely, it outputs a basis with length λ_1 and λ_2.

Lemma 4.3.10. *Let b_1 and b_2 non-zero vectors. Define b_2^* by*

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

Then b_2^* is the projection of b_2 onto the orthogonal line of b_1.

Proof. Let

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

Then

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

This means that b_2^* is orthogonal to b_1. Moreover, since $\langle b_1, b_2\rangle = \|b_1\|\|b_2\|\cos(b_1, b_2)$, then

$$\begin{aligned}\|b_2^*\|^2 &= \|b_2\|^2 - \frac{2\langle b_1, b_2\rangle^2}{\|b_1\|^2} + \frac{\langle b_1, b_2\rangle^2}{\|b_1\|^2} \\ &= \|b_2\|^2 - \|b_2\|^2\cos^2(b_1, b_2) \\ &= \|b_2\|^2\left(1 - \cos^2(b_1, b_2)\right) \\ &= \|b_2\|^2\sin^2(b_1, b_2).\end{aligned}$$

Hence $|b_2^*\| = \|b_2\|\sin(b_1, b_2)|$. This means that b_2^* is the projection of b_2 onto the orthogonal line of b_1. □

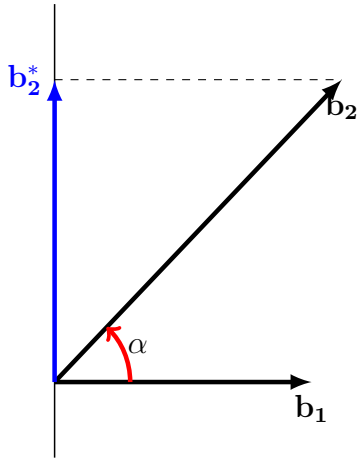


Figure 4.7: The orthogonal projection b_2^* of b_2 with $\alpha = (b_1, b_2)$

Since b_2^* is not necessary in $\mathcal{L}$, we replace the basis (b_1, b_2) by a basis (u_1, u_2) where

$$u_1 := b_1, \quad u_2 = b_2 - \left\lfloor \frac{\langle b_1, b_2\rangle}{\|b_1\|^2} \right\rceil b_1,$$

where $\lfloor x \rceil$ is the nearest integer to x.

Proposition 4.3.11. *Let b_1, b_2 be two linearly independent vectors. Define*

$$u_2 = b_2 - \left\lfloor \frac{\langle b_1, b_2\rangle}{\|b_1\|^2} \right\rceil b_1.$$

Then

$$-\frac{1}{2} < \frac{\langle b_1, u_2\rangle}{\|b_1\|^2} \leq \frac{1}{2}.$$

Proof. Let $m = \left\lfloor \frac{\langle b_1, b_2 \rangle}{\|b_1\|^2} \right\rceil$. Then

$$m - \frac{1}{2} < \frac{\langle b_1, b_2 \rangle}{\|b_1\|^2} \leq m + \frac{1}{2}.$$

Hence

$$-\frac{1}{2} < \frac{\langle b_1, b_2 \rangle}{\|b_1\|^2} - m \leq \frac{1}{2}.$$

We have

$$\frac{\langle b_1, b_2 \rangle}{\|b_1\|^2} - m = \frac{\langle b_1, b_2 \rangle - m\|b_1\|^2}{\|b_1\|^2} = \frac{\langle b_1, b_2 - mb_1 \rangle}{\|b_1\|^2} = \frac{\langle b_1, u_2 \rangle}{\|b_1\|^2}.$$

Consequently, we get

$$-\frac{1}{2} < \frac{\langle b_1, u_2 \rangle}{\|b_1\|^2} \leq \frac{1}{2}.$$

This terminates the proof. □

Exercice 4.3.12. Let b_1, b_2 be two linearly independent vectors. Define

$$u_1 = b_1, \qquad u_2 = b_2 - \left\lfloor \frac{\langle b_1, b_2 \rangle}{\|b_1\|^2} \right\rceil b_1.$$

Show that, if (b_1, b_2) is a basis of a lattice L, then $(u_1, u2)$ is also a basis of L.

If $\|u_1\| \geq \|u_2\|$, we swap u_1 and u_2 and continue the process.
The Gaussian reduction algorithm is described in Algorithm 9.

Algorithm 9 : Gauss' algorithm

Input : A basis (b_1, b_2) of a lattice $\mathcal{L} \subset \mathcal{R}^2$.
Output : A basis (u_1, u_2) such that $\|u_1\| = \lambda_1$ and $\|u_2\| = \lambda_2$.

1: Set $u_1 = b_1$ and $u_2 = b_2$.
2: **If** $\|u_1\| > \|u_2\|$ **then**
3: Swap u_1 and u_2.
4: **End If**
5: **While** $\|u_1\| < \|u_2\|$, **do**
6: Compute $\mu = \frac{\langle u_1, u_2 \rangle}{\|u_1\|^2}$.
7: Compute $u_2 = u_2 - \lfloor \mu \rceil u_1$.
8: Swap u_1 and u_2.
9: **End While**
10: Swap u_1 and u_2.
11: Return (u_1, u_2).

Exercice 4.3.13. Apply Gauss algorithm with the basis (b_1, b_2) with $b_1 = (6, 3)$ and $b_2 = (5, 1)$.

We get $u_1 = (1, 2)$ and $b_2 = (4, -1)$.

Maple code 4.3.14.

The algorithm 9 can be easily encoded using maple.

```
      Maple: Gauss'algorithm

gauss:=proc(b1,b2)
local u1,u2,v,mu:
with(VectorCalculus):
u1:=convert(b1, Vector);
u2:=convert(b2, Vector);
if Norm(1.0*u1)>= Norm(1.0*u2) then
    v:=u1;
    u1:=u2;
    u2:=v;
end if;
while Norm(1.0*u1)<Norm(1.0*u2) do
    mu:=DotProduct(u1,u2)/Norm(u1)^2:
    u2:=u2-round(mu)*u1:
    v:=u1;
    u1:=u2;
    u2:=v;
end do;
v := u1;
u1 := u2;
u2 := v;
u1:=convert(u1, Vector);
u2:=convert(u2, Vector);
return(u1, u2):
end proc:
```

Example 4.3.15.

Let $\mathcal{L}$ be the lattice with the basis (b_1, b_2) with

$$b_1 = \begin{bmatrix} 19239 \\ 2971 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 22961 \\ 3546 \end{bmatrix}.$$

```
      Maple: Gauss'algorithm,  example1

b1 := [19239, 2971];
b2 := [22961, 3546];
gauss(b1, b2);
```

Then, applying Gauss'algorithm, we find a new basis (u_1, u_2) with

$$u_1 = \begin{bmatrix} -52 \\ -1 \end{bmatrix}, \quad u_2 = \begin{bmatrix} 5 \\ 84 \end{bmatrix}.$$

Then, we get

$$\lambda_1 = \|u_1\| = \sqrt{2705}, \quad \lambda_2 = \|u_2\| = \sqrt{7081}.$$

Observe that

$$u_1 = [b_1, b_2] \begin{bmatrix} -37 \\ 31 \end{bmatrix}, \quad u_2 = [b_1, b_2] \begin{bmatrix} -438 \\ 367 \end{bmatrix},$$

and that the fractions -$\frac{37}{31}$ and $-\frac{438}{367}$ are among the convergents of the continued fraction expansion of $-\frac{22961}{19239}$ as explained in Example 4.3.2.

Observe that the Gaussian algorithm can be applied to any two linearly independent vectors b_1, b_2 in the $\mathbb{R}^n$ for any $n \geq 2$.

Exercice 4.3.16. Apply Gauss algorithm with the basis (b_1, b_2) with $b_1 = (49, 70, -35)$ and $b_2 = (58, 89, -48)$.

We get $u_1 = (13, -6, 17)$ and $b_2 = (9, 19, -13)$.

4.4 L'algorithme LLL

L'algorithme de Gauss permet de réduire une base (b_1, b_2) et déterminer une base (u_1, u_2) avec de bonnes proprités, à savoir, u_1 et u_2 sont les plus courts possibles dans le réseau et sont presque orthogonaux. Pour généraliser la réduction à un nombre n de vecteurs indépendants $(b_1, \ldots, b_n)$, on utilise l'algorithme LLL de Lenstra, Lenstra et Lovász [15].

The LLL algorithm is the most useful tool in the algorithmic study of lattices. It provides a partial answer to SVP since it runs in polynomial time and approximates the shortest vector of a lattice of dimension n up to a factor of $2^{n/2}$. Babai gave an algorithm that approximates the CVP problem by a factor of $\left(3/\sqrt{2}\right)^n$. In some cases, LLL gives extremely striking results both in theory and practice that are enough to solve the problem. LLL uses the well known Gram-Schmidt orthogonalization method. The Gram-Schmidt process is an iterative method to orthonormalize the basis of a vector space. The LLL algorithm is connected to the Gram-Schmidt orthogonolization process and produces a basis that satisfies the LLL-reduction notion as in the following definition.

Definition 4.4.1. A basis $(b_1 \cdots, b_n)$ of a lattice $\mathcal{L}$ is said to be LLL-reduced if the Gram-Schmidt orthogonalization $(b_1^* \cdots, b_n^*)$ satisfies

$$|\mu_{i,j}| = \frac{|\langle b_i, b_j^* \rangle|}{\langle b_j^*, b_j^* \rangle} \leq \frac{1}{2}, \quad \text{for} \quad 1 \leq j < i \leq n, \tag{4.1}$$

$$\frac{3}{4}\|b_{i-1}^*\|^2 \leq \|b_i^* + \mu_{i,i-1} b_{i-1}^*\|^2, \quad \text{for} \quad 1 < i \leq n. \tag{4.2}$$

Using

$$|\mu_{i,j}| = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} = \frac{\|b_i\|}{\|b_j^*\|} |\cos(b_i, b_j^*)|,$$

we see that if $\mu_{i,j} = 0$ for all i and j, then the basis is orthogonal, and consequently is minimal according to Hadamard's inequality as in Corollary 4.2.12.
Also, the condition (4.1) means that the norm of b_i onto the line spanned by b_j^* is less than half of the norm of b_j^* and that the vector b_i is almost orthogonal the space spanned by the vectors $b_1, \ldots, b_{i-1}$.
The condition (4.2) is called Lovász' condition. It means that the norm of the projection of b_i onto the orthogonal space spanned by $(b_1, \cdots, b_{i-1})$ is larger than $\frac{3}{4}$ times the norm of the projection of b_{i-1} onto the orthogonal space spanned by $(b_1, \cdots, b_{i-2})$. Also, it can be transformed into the inequality

$$\|b_i^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) \|b_{i-1}^*\|^2.$$

Since a lattice has infinitely many basis, some basis are better than others. A *good basis* is generally a basis with short and almost orthogonal vectors. Consequently, a LLL-reduced basis is a candidate for a good basis.

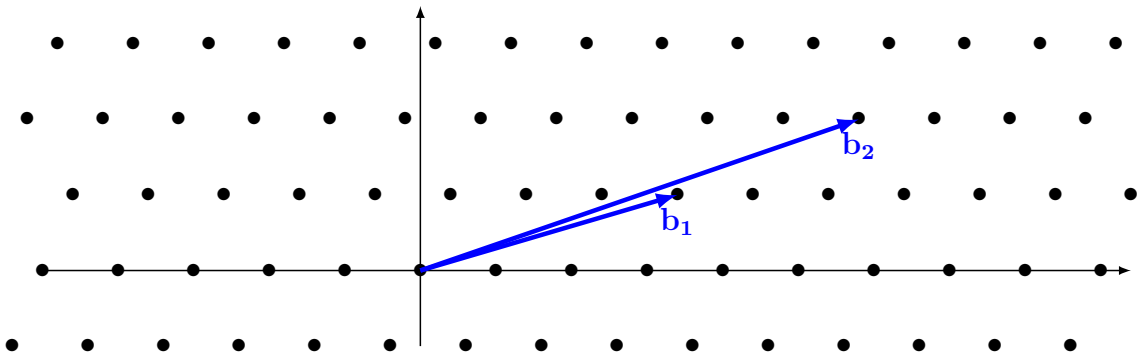


Figure 4.8: A lattice with *a bad* basis (b_1, b_2)

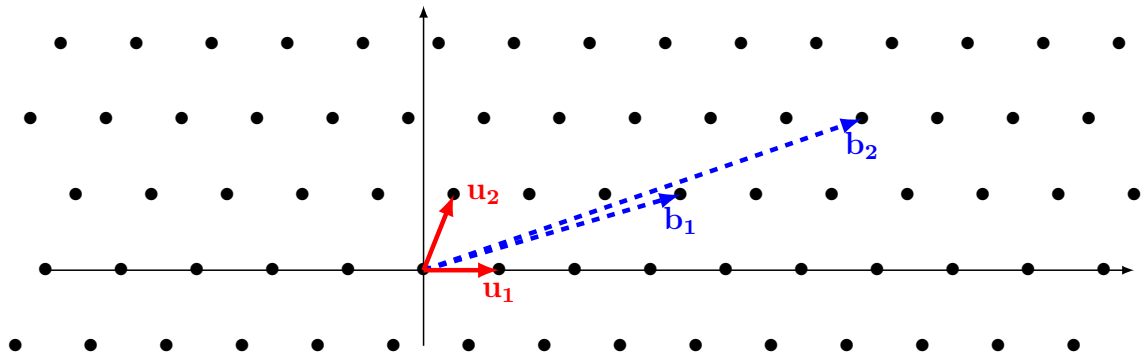


Figure 4.9: The same lattice with *a good* basis (u_1, u_2)

The original version of the LLL algorithm is presented in Algorithm (10).

Maple code 4.4.2.

Using the description of the LLL algorithm in Algorithm (10) and the procedures 11 and 12 , we can encode it using maple. The corresponding code needs the procedure **scal(u,v)** for the inner product and the procedure **gramschmidt(M)** for the Gram-Schmidt orthogonolization process. We start by encoding the procedure **redfaible(M)**.

Procedure 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`

Maple code 4.4.3.

Algorithm 10 : LLL Algorithm

Input : A basis $(b_1, \cdots, b_n)$

Output : A LLL reduced basis $(b_1, \cdots, b_n)$

For $i = 1, \cdots, n$ **do**
  $b_i^* = b_i$
  For $j = 1, \cdots, i-1$ **do**
    $B_j = \langle b_j^*, b_j^* \rangle$
    $\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{B_j}$
    $b_i^* = b_i^* - \mu_{i,j} b_j^*$
  End For
  $B_i = \langle b_i^*, b_i^* \rangle$
End For
$k = 2$
Apply the procedure **redfaible**$(k, k-1)$ [11]
If $\frac{3}{4} B_{k-1} > B_k + \mu_{k,k-1}^2 B_{k-1}$ **then**
  Apply **lovasz**(k) [12]
End If
Goto 10
For $l = k-2, \cdots, 1$ **do**
  Apply the procedure **redfaible**(k, l) [11]
End For
If $k = n$ **then**
  Stop
Else
  $k = k + 1$
  Goto 10
End If

Algorithm 11 : Procedure **redfaible**

Input : Two integers k and l.

Output : A vector b_k and $l-1$ values $\mu_{k,j}$ with $j = 1, \cdots l-1$ such that $|\mu_{k,l}| \leq \frac{1}{2}$

If $|\mu_{k,l}| > \frac{1}{2}$ **then**
  $b_k = b_k - \lfloor \mu_{k,l} \rceil b_l$ where $\lfloor x \rceil$ is the nearest integer to x.
  $\mu_{k,l} = \mu_{k,l} - \lfloor \mu_{k,l} \rceil$
  For $j = 1, \cdots, l-1$ **do**
    $\mu_{k,j} = \mu_{k,j} - \lfloor \mu_{k,l} \rceil \mu_{l,j}$
  End For
End If

Algorithm 12 : Procedure **lovasz**

Input : An integer k

Output : Test if the condition $\frac{3}{4}B_{k-1} \leq B_k + \mu_{k,k-1}^2 B_{k-1}$ is satisfied.

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: Swap b_k and b_{k-1}.
6: **For** $j = 1, \cdots, k-2$ **do**
7: Swap $\mu_{k-1,j}$ and $\mu_{k,j}$
8: **End For**
9: **For** $i = k+1, \cdots, n$ **do**
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: **End For**
13: **If** $k > 2$ **then**
14: $k = k - 1$
15: **End If**

Next, we encode the procedure **lovasz(M)**.

```
      Procedure 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:
```

Maple code 4.4.4.

Finally, we encode the LLL algorithm as the procedure **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:
```

Example 4.4.5.

We will need the scalr product of two vectors.

```
      Produit Scalaire

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

Also, we will need the Gram-Schmidt orthogonalization.

```
      La méthode de Gram-Schmidt

gramschmidt:=proc (M)
local i, j, n, B, P, u,k:
with(LinearAlgebra):
n := nops(M):
B := []: P := []:
for i from 1 to n do
    u := [seq(0, k = 1 .. n)]:
    for j from 1 to i-1 do
        u[j]:=scal(M[i],B[j])/scal(B[j],B[j]):
    end do:
    u[i] := 1:
    P := [op(P), u]:
    B:=[op(B),[seq(M[i][k]-add(u[j]*B[j][k],
        j = 1 .. i-1), k = 1 .. n)]]:
end do:
return(B, P):
end proc:
```

```
      Commentaires

<-- Procédure
<-- Paramètres locaux
<-- Librairie LinearAlgebra
<-- Nombre de lignes de M
<-- Initiation des listes
<-- i=1,...,n
<-- Initiation de u
 for j from 1 to i-1 do
<-- Calcul de u[j]

<-- u[i]= 1
<-- concaténation
<-- Calcul de B



<-- Sortie
```

Example 4.4.6.

Define the lattice $\mathcal{L}$ by the basis (b_1, b_2, b_3, b_4) with

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

Maple code 4.4.7.

The corresponding maple code is as follows.

```
      Programme

M1 := [4, 7, 9, 4];
M2 := [6, -7, 2, 3];
M3 := [-1, 2, -1, -1];
M4 := [2, -1, 0, -3];
M := [M1, M2, M3, M4];
M5 := Matrix(myLLL(M));
```

Then, the procedure **myLLL(M)** outputs the basis (u_1, u_2, u_3, u_4) with the matrix

$$N = \begin{bmatrix} -1 & 2 & -1 & -1 \\ 2 & 1 & -2 & -1 \\ -1 & 0 & 1 & -3 \\ 5 & 8 & 8 & 0 \end{bmatrix}.$$

Maple code 4.4.8.

The LLL algorithm is implemented in maple with the procedure **LLL** in the package **IntegerRelations**.
Using the same lattice $\mathcal{L}$ with the basis (b_1, b_2, b_3, b_4) as in Example 4.4.6, we can use the following code.

```
      Programme

with(IntegerRelations):
M := [M1, M2, M3, M4]:
N:=LLL(M,'integer');
```

This outputs the LLL-reduced basis (u_1, u_2, u_3, u_4) with the matrix

$$N = \begin{bmatrix} -1 & 2 & -1 & -1 \\ 2 & 1 & -2 & -1 \\ -1 & 0 & 1 & -3 \\ 5 & 8 & 8 & 0 \end{bmatrix}.$$

Using the lattice as in Example 4.4.6, we can check that the bases (b_1, b_2, b_3, b_4) and (u_1, u_2, u_3, u_4) have the same determinant and that u_1 is shorter than the other vectors of the basis (u_1, u_2, u_3, u_4).

Maple code 4.4.9.

The following procedure tests whether $\det(b_1, b_2, b_3, b_4) = \det(u_1, u_2, u_3, u_4)$ and compares the norms of the vectors u_1, u_2, u_3, u_4.

```
      Programme

B := Matrix([M1, M2, M3, M4]):
U := Matrix(N):
Determinant(B)-Determinant(U):
for i to 4 do
    print(U[i], "has norm", DotProduct(Vector(U[i]), Vector(U[i]))):
end do;
```

The LLL algorithm outputs a reduced basis that has many interesting properties such as the following ones.

Theorem 4.4.10. *Let $(b_1 \cdots, b_n)$ be an LLL-reduced basis with Gram-Schmidt orthogonolization $(b_1^*, \cdots, b_n^*)$. Then*

1. $\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2$ *for* $1 \leq j \leq i \leq n$.
2. $\det(L) \leq \prod_{i=1}^{n} \|b_i\| \leq 2^{\frac{n(n-1)}{4}} \det(L)$.
3. $\|b_j\| \leq 2^{\frac{i-1}{2}}\|b_i^*\|$ *for* $1 \leq j \leq i \leq n$.
4. $\|b_1\| \leq 2^{\frac{n-1}{4}}(\det(L))^{\frac{1}{n}}$.

Proof.
`Proof of 1.` Suppose that $(b_1 \cdots, b_n)$ is an LLL-reduced basis. Then, expanding the inequality (4.2) and using (4.1), we get

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

from which we deduce

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

Hence, for $j \leq i$, we get

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

which proves the property 1.
`Proof of 2.` In the Gram-Schmidt orthogonalization process, we have $b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$. Combining with (4.1), we get

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

Using $\|b_j^*\|^2 \leq 2^{i-j}\|b_i^*\|^2$, we get

$$\|b_i\|^2 \leq \|b_i^*\|^2 + \frac{1}{4} \sum_{j=1}^{i-1} 2^{i-j} \|b_i^*\|^2 = \left(1 + 2^{i-2} + 2^{-1}\right) \|b_i^*\|^2 \leq 2^{i-1}\|b_i^*\|^2. \tag{4.3}$$

Taking the product and using Corollary 4.2.12, we get

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

Taking square roots, this proves the property 2.
`Proof of 3.` Replacing i by j in the inequality 4.3, we get $\|b_j\|^2 \leq 2^{j-1}\|b_j^*\|^2$. Combining with property 1, we get

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

which proves the property 3.
`Proof of 4.` If we take $j = 1$ in the inequality 3, we get $\|b_1\|^2 \leq 2^{i-1}\|b_i^*\|^2$ for $1 \leq i \leq n$. Then

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

which gives $\|b_1\| \leq 2^{\frac{n-1}{4}} (\det(L))^{\frac{1}{n}}$ and proves the inequality in 4. □

Theorem 4.4.11. *Let $(b_1 \cdots, b_n)$ be an LLL-reduced basis with Gram-Schmidt orthogonolization $(b_1^*, \cdots, b_n^*)$. Then for any non zero vector $v \in L$, $\|b_1\| \leq 2^{\frac{n-1}{2}}\|v\|$.*

Proof. Let $v \in L$ be a nonzero vector. Writing v simultaneously in the bases $(b_1, b_2, \cdots, b_n)$ and $(b_1^*, b_2^*, \cdots, b_n^*)$, we get

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

Let k be the greatest integer with $1 \leq k \leq n$ and $\alpha_k \neq 0$, that is $|\alpha_k| \geq 1$ and $\alpha_i = 0$ for $i = k+1, \ldots, n$. Using b_i as in the Gram-Schmidt orthogonalization process, we get

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

Then

$$\alpha_k^* = \alpha_k + \sum_{i=k+1}^{n} \alpha_i \mu_{i,k} = \alpha_k.$$

Expanding $\|v\|^2$ in $(b_1^*, b_2^*, \cdots, b_n^*)$, we get

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

Taking $j = 1$ and $i = k$ in the inequality in (3) of Proposition 4.4.10, we get

$$\|b_k^*\|^2 \geq 2^{1-k} \|b_1\|^2.$$

Hence

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

which gives $\|b_1\| \leq 2^{\frac{n-1}{2}} \|v\|$ and terminates the proof. □

Observe that, as shown in property 5 of Proposition 4.4.11, the LLL algorithm outputs a basis where the first vector is short. It is the shortest vector in the lattice up to the factor $2^{\frac{n-1}{2}}$. Frequently, in low dimensions, it is the shortest nonzero vector in the lattice.
The following result gives the size of the the vectors of an LLL-reduced basis when $\mathcal{L} \subset \mathbb{Z}^n$, that is when the coordinates of the basis vectors are integers.

Theorem 4.4.12. *Let $(b_1, \cdots, b_n)$ be an LLL-reduced basis of a lattice $\mathcal{L} \subset \mathbb{Z}^n$. Then for $1 \leq j \leq n$, we have*

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

Proof. Let$(b_1^*, \cdots, b_n^*)$ be the orthogonal basis associated to $(b_1, \cdots, b_n)$ in the Gram-Schmidt process. Using property (3) of Theorem 4.4.10, we get, for $1 \leq j \leq i \leq n$,

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

Applying this successively for $i = j, j+1, \cdots, n$ and multiplying all the inequalities, we get

$$\|b_j\|^{n-j+1} \leq \prod_{i=j}^{n} 2^{\frac{i-1}{2}} \|b_i^*\|. \tag{4.4}$$

Using Property (3) of Theorem 4.4.10 with $j = 1$, we get, for $1 \leq i \leq n$,

$$\|b_1\| \leq 2^{\frac{i-1}{2}} \|b_i^*\|.$$

Then, since $\mathcal{L} \subset \mathbb{Z}^n$, then, for $1 \leq i \leq n$,

$$2^{\frac{i-1}{2}} \|b_i^*\| \geq \|b_1\| \geq 1,$$

and

$$\prod_{i=1}^{j-1} 2^{\frac{i-1}{2}} \|b_i^*\| \geq \|b_1\| \geq 1,$$

Then, transforming (4.4), we get

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

which leads the inequality of the theorem. □

Note that the LLL algorithm provides a basis of reasonably short vectors and can be used to approximate the shortest vector problem. The next result shows that the LLL algorithm is a polynomial time algorithm.

Theorem 4.4.13. *Let $(b_1, \cdots, b_n)$ be a basis of a lattice $\mathcal{L}$. Define $B = \max_i \|b_i\|$. The LLL algorithm computes an LLL-reduced basis with running time*

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

Chapter 5

Cryptanalyse de RSA par la Méthode de Coppersmith

5.0.1 La méthode de Coppersmith : polynômes à une variable

An important application of lattice reduction found by Coppersmith [5] in 1996 is finding small roots of low-degree polynomial equations. This includes modular univariate polynomial equations, and bivariate integer equations.

Let M be some large integer of unknown factorization and

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

be a polynomial of degree d with integer coefficients. Consider the equation $f(x) \equiv 0 \pmod{M}$. In general there is no known efficient algorithm that find integer roots of the above equation. However, Coppersmith [5] introduced an efficient method for finding small integer solutions using the LLL algorithm. Suppose we know that there exists an integer x_0 such that $f(x_0) \equiv 0 \pmod{M}$ and that $|x_0| < N^{\frac{1}{d}}$. The problem is to find x_0. The main idea is that if the coefficients of f are small enough so that $|f(x_0)| = \sum_{i=1}^{d} |a_i x^i| < M$, then one might have $f(x_0)$ over the integers. Coppersmith's idea is to build from $f(x)$ a polynomial $h(x)$ which has small coefficients and the same solution x_0. Soon after Coppersmith proposed his method in 1996, Howgrave-Graham [**?**] proposed in 1997 a a new method for finding all small integer roots. Suppose we know an upper bound X such that $|x_0| < X$. The following theorem by Howgrave-Graham reformulates Coppersmith's idea of finding modular roots. We define the Euclidean norm of a polynomial $f(x)$ as

$$\|f(x)\| = \left(\sum_{i=0}^{d} a_i^2 \right)^{\frac{1}{2}}.$$

Theorem 5.0.1 (Howgrave-Graham)**.** *Let $h(x) \in \mathbb{Z}[x]$ be a polynomial of at most ω monomials satisfying*

(1) $|x_0| < X$, for some positive integer X.

(2) $h(x_0) \equiv 0 \pmod{M}$, for some positive integer M.

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

Then $h(x_0) = 0$ over $\mathbb{Z}$.

Proof. Let $h(x) = \sum_i^d a_i x^i$ with ω monomials. Suppose $|x_0| < X$. Then

$$|h(x_0)| = \left|\sum_i a_i x_0^i\right| \leq \sum_i \left|a_i x_0^i\right| < \sum_i \left|a_i X^i\right|. \tag{5.1}$$

Recall that Cauchy-Schwarz inequality asserts that for $\alpha, \beta \in \mathbb{R}$, we have

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

Using this, we get

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

If $\|h(xX)\| < \frac{M}{\sqrt{\omega}}$, then, using (5.1), we get

$$|h(x_0)| < \sum_i \left|a_i X^i\right| < \sqrt{\omega}\sqrt{\sum_i \left(a_i X^i\right)^2} = \sqrt{\omega}\|h(xX)\| < M.$$

Hence $|h(x_0)| < M$. Finally, if $h(x_0) \equiv 0 \pmod{M}$, then $h(x_0) = 0$ over $\mathbb{Z}$ which terminates the proof. □

To solve $f(x_0) \equiv 0 \pmod{M}$, Theorem 5.0.1 suggests we should look for a polynomial $h(x)$ of small norm satisfying $h(x_0) \in Z$. To do this we will build a lattice of polynomials related to f and use LLL to find short vectors in the lattice. The following result, as given below, is from May [17].

Theorem 5.0.2. *For every $\varepsilon > 0$ there exists an N_0 such that the following holds: Let $N > N_0$ be an integer with unknown factorization which has a divisor $b > N^\beta$. Let $f_b(x)$ be a monic univariate polynomial of degree δ. All solutions x_0 of the congruence $f_b(x) \equiv 0 \pmod{b}$, such that*

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

can be found in time polynomial in $\log(N)$.

Proof. Fix two positive integers m and t and consider the polynomials

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

where $\delta = \deg(f_b)$. Observe that all the polynomials share the root x_0 modulo N^m. Rewriting the polynomials explicitly, we get

	$j=0$	$j=1$	$j=2$	$\cdots$	$j=\delta-1$
$i=m$	N^m,	$N^m x$,	$N^m x^2$,	$\cdots$	$N^m x^{\delta-1}$,
$i=m-1$	$N^{m-1} f_b(x)$	$N^{m-1} x f_b(x)$	$N^{m-1} x^2 f_b(x)$	$\cdots$	$N^{m-1} x^{\delta-1} f_b(x)$
$i=m-2$	$N^{m-2} f_b(x)^2$	$N^{m-2} x f_b(x)^2$	$N^{m-2} x^2 f_b(x)^2$	$\cdots$	$N^{m-2} x^{\delta-1} f_b(x)^2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$i=2$	$N^2 f_b(x)^{m-2}$	$N^2 x f_b(x)^{m-2}$	$N^2 x^2 f_b(x)^{m-2}$	$\cdots$	$N^2 x^{\delta-1} f_b(x)^{m-2}$
$i=1$	$N f_b(x)^{m-1}$	$N x f_b(x)^{m-1}$	$N x^2 f_b(x)^{m-1}$	$\cdots$	$N x^{\delta-1} f_b(x)^{m-1}$

(5.2)

Observe that the maximal degree is $\delta - 1 + (m-1)\delta = m\delta - 1$. The details of the polynomials $h_i(x)$ are as follows

$$i = 0, \cdots, t-1 \Rightarrow f_b^m(x), x f_b^m(x), x^2 f_b^m(x), \cdots, x^{t-1} f_b^m(x). \tag{5.3}$$

Observe here that the maximal degree is $t - 1 + m\delta > m\delta - 1$. Replacing x by Xx in the rows $i = m, m-1, \ldots, 1$ of the table (5.2) and in the sequence (5.3) and expressing in the basis $\left(1, x, x^2, \cdots, x^{m\delta+t-1}\right)$, we get a sequence of matrices of the shape

$$M_m = \begin{bmatrix} N^m & & & \\ & N^m X & & \\ & & \ddots & \\ & & & N^m X^{\delta-1} \end{bmatrix},$$

$$M_{m-1} = \begin{bmatrix} - & - & - & - & N^{m-1}X^{\delta} & & & \\ - & - & - & - & - & N^{m-1}X^{\delta+1} & & \\ - & - & - & - & - & - & \ddots & \\ - & - & - & - & - & - & - & N^{m-1}X^{2\delta-1} \end{bmatrix},$$

$$\vdots = \vdots$$

$$M_1 = \begin{bmatrix} - & - & \cdots & - & NX^{(m-1)\delta} & & & \\ - & - & \cdots & - & - & NX^{(m-1)\delta+1} & & \\ - & - & \cdots & - & - & - & \ddots & \\ - & - & \cdots & - & - & - & - & NX^{(m-1)\delta+\delta-1} \end{bmatrix},$$

$$M_0 = \begin{bmatrix} - & - & \cdots & - & X^{m\delta} & & & \\ - & - & \cdots & - & - & X^{m\delta+1} & & \\ - & - & \cdots & - & - & - & \ddots & \\ - & - & \cdots & - & - & - & - & X^{m\delta+t-1} \end{bmatrix}.$$

Gathering the matrices, we get a triangular matrix of the form

$$M = \begin{bmatrix} M_m \\ M_{m-1} \\ \vdots \\ M_1 \\ M_0 \end{bmatrix}, \tag{5.4}$$

which generates a lattice $\mathcal{L}$. Obviously, we have

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

where $n = m\delta + t$. Using the LLL-algorithm, we can find a small element in $\mathcal{L}$ that corresponds to a polynomial $h(x)$ satisfying (4) of Theorem 4.4.10, namely

$$\|h(xX)\| \leq 2^{\frac{n-1}{4}} \det(L)^{\frac{1}{n}} = 2^{\frac{n-1}{4}} N^{\frac{m(m+1)\delta}{2n}} X^{\frac{1}{2}(n-1)}.$$

In order to apply Theorem 5.0.1 on $h(x)$, it is sufficient that $\|h(xX)\| \leq \frac{b^m}{\sqrt{n}}$, holds. This is satisfied if

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

Plugging $b > N^{\beta}$, we find

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

Solving for X, we get

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

Consider the exponent of N as a polynomial in m. The exponent is maximal for

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

which leads to the bound

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

This can be rewritten as

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

where

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

Observe that ε depends on n and satisfies $\lim_{n \to +\infty} \varepsilon = 0$. □

Example 5.0.3.

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

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

Maple code 5.0.4.

Programme	Commentaires
restart;	
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
f(79) mod N;	<-- Vérification

5.0.2 Factorisation de N

Theorem 5.0.2 has various applications in cryptography. We will now present an attack on RSA - also due to Coppersmith - that finds the factorization of $N = pq$, provided that one knows half of the bits of one of the factors.

Theorem 5.0.5. *Let $N = pq$ be an RSA modulus with $p > q$. If $\tilde{p}$ is an approximation of p with*

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

then N can be factored in polynomial time in $\log N$.

Proof. Suppose we know an approximation $\tilde{p}$ of p with $|\tilde{p} - p| < N^{\frac{1}{4}}$. Consider the polynomial $f_p(x) = x + \tilde{p}$. Then $f_p(p - \tilde{p}) = p \equiv 0 \mod p$. Hence, $x_0 = p - \tilde{p}$ satisfies

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

Since $p > N^{\frac{1}{2}}$, one can then apply Theorem 5.0.2 with $b = p$, $f_p(x) = x + \tilde{p}$, $\delta = 1$ and $\beta = \frac{1}{2}$. This gives explicitly x_0 which leads to $p = x_0 + \tilde{p}$. □

Example 5.0.6.

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 5.0.2 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 $p = p_0 + x_0 = 1125899906842679$.

Maple code 5.0.7.

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`

Chapter 6

Le Cryptosystème NTRU

6.1 Introduction au cryptosystème NTRU

Le cryptosystème NTRU utilise plusieurs sortes de paramètres, en particulier les paramètres N, p et q.

Soit N un nombre entier. Les opérations de NTRU ont pour domaine l'anneau des polynômes suivant:

$$\mathcal{P} = \mathbb{Z}[X]/\left(X^N - 1\right).$$

Ainsi, les éléments f de $\mathcal{P}$ peuvent être représentés sous la forme

$$f = (f_0, f_1, \cdots, f_{N-1}) = \sum_{i=0}^{N-1} f_i X^i.$$

L'addition de deux polynômes $f, g \in \mathcal{P}$ se fait de façon naturelle terme à terme de même degrés, alors que la multiplication se fait par un produit de convolution noté ici $*$. Si $f * g = h$ avec $f = \sum_{i=0}^{N-1} f_i X^i$ et $g = \sum_{i=0}^{N-1} g_i X^i$, alors $h = \sum_{i=0}^{N-1} h_i X^i$ avec pour tout $0 \leq k \leq N-1$,

$$h_k = \sum_{i+j \equiv k \mod N} f_i g_j = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{N-1} f_i g_{N+k-i}.$$

Different descriptions of NTRUEncrypt, and different proposed parameter sets, have been in circulation since 1996. The 2005 instantiation of NTRU is set up by six public integers N, p, q, d_f, d_g, d_r and four public spaces $\mathcal{L}_f$, $\mathcal{L}_g$, $\mathcal{L}_m$, $\mathcal{L}_r$.

- N is prime and sufficiently large to prevent lattice attacks.
- p and q are relatively prime numbers.

- q is much larger than p.
- $\mathcal{L}_f = \mathcal{B}(d_f)$ is a set of small polynomials from which the private keys are selected.
- $\mathcal{L}_g = \mathcal{B}(d_g)$ is a similar set of small polynomials from which other private keys are selected.
- $\mathcal{L}_m = \mathbb{Z}_p[X]/\left(X^N - 1\right)$ is the plaintext space. It is a set of polynomials $m \in \mathbb{Z}_p[X]/(X^N - 1)$ that represent encryptable messages.
- $\mathcal{L}_r = \mathcal{B}(d_r)$ is a set of polynomials from which the blinding value used during encryption is selected.

The key generation, encryption and decryption primitives are as follows:

1. **Key generation**
   - Randomly choose a polynomial $f \in \mathcal{L}_f$ such that f is invertible in $\mathcal{P}$ modulo p and modulo q.
   - Compute $f_p \equiv f^{-1} \pmod{p}$ and $f_q \equiv f^{-1} \pmod{q}$.
   - Randomly choose a polynomial $g \in \mathcal{L}_g$.
   - Compute $h \equiv g * f_q \pmod{q}$.
   - Publish the public key (N, h) and the set of parameters p, q, $\mathcal{L}_f$, $\mathcal{L}_g$, $\mathcal{L}_r$ and $\mathcal{L}_m$.
   - Keep the private key (f, f_p).
2. **Encryption**
   - Represent the message as a polynomial $m \in \mathcal{L}_m$.
   - Randomly choose a polynomial $r \in \mathcal{L}_r$.
   - Encrypt m with the public key (N, h) using the rule $e \equiv p * r * h + m \pmod{q}$.
3. **Decryption**
   - The receiver computes $a \equiv f * e \pmod{q}$.
   - Using a centering procedure, transform a to a polynomial with coefficients in the interval $\left[-\frac{q}{2}, \frac{q}{2}\right[$.
   - Compute $m \equiv f_p * a \pmod{p}$.

The decryption process is correct if the polynomial $p*r*g+f*m \pmod q$ is actually equal to $p*r*g+f*m \in \mathbb{Z}[X]/\left(X^N-1\right)$, that is without using modulo q. We have

$$\begin{aligned} a &\equiv f*e \pmod q \\ &\equiv f*(p*r*h+m) \pmod q \\ &\equiv f*r*(p*g*f_q)+f*m \pmod q \\ &\equiv p*r*g*f*f_q+f*m \pmod q \\ &\equiv p*r*g+f*m \pmod q. \end{aligned}$$

Hence, if $a=p*r*g+f*m$ in $\mathbb{Z}[X]/\left(X^N-1\right)$, then

$$a*f_p \equiv (p*r*g+f*m)*f_p \equiv m \pmod p.$$

We note that if the parameters are chosen properly, the decryption process never fails. A sufficient condition for this is to choose q much larger than p.

Maple code 6.1.1.

Avec Maple 12, la procédure suivante permet de calculer $f*g$ dans l'anneau $\mathcal{P}$.

Programme (Maple)	Commentaires
`star:=proc (N,f,g)`	`<--- Procedure star`
`local k, h, i, s;`	`<--- Variables locales`
`for k from 0 to N-1 do`	`<--- Création de la`
`  h[k] := 0;`	`  suite h_k`
`  for i from 0 to k do`	`<--- Partie i=0..k`
`h[k]:=h[k]+coeff(f,X,i)*coeff(g,X,k-i)`	
`  end do;`	`<--- Fin partie i=0..k`
`  for i from k+1 to N-1 do`	`<--- Partie i=k+1..N-1`
`h[k]:=h[k]+coeff(f,X,i)*coeff(g,X,N+k-i)`	
`  end do;`	`<--- Fin partie i=k+1..N-1`
`end do;`	
`s := 0;`	`<--- Formation du`
`for k from 0 to N-1 do`	`  polynôme h=f*g`
`  s := s+h[k]*X^k`	
`end do;`	`<--- Fin`
`return s;`	`<--- Sortie de h=f*g`
`end proc;`	`<--- Fin de la procédure`

Pour illustrer cette procédure, on considère $N=17$, $f=X^{16}-1$ et $g=X^{15}+1$. Le programme suivant permet de calculer $s=f+g$ et $h=f*g$ dans $\mathcal{P}=\mathbb{Z}[X]/\left(X^{17}-1\right)$.. On suppose donc que la procédure **star(N,f,g)** est déjà exécutée.

Programme (Maple)	Commentaires
N:=17;	<--- Valeur de N
f:=X^16-1;	<--- Polynôme f
g:=X^15+1;	<--- Polynôme g
h:=star(N,f,g);	<--- h=f*g=-1+X^14-X^15+X^16

Soient $p, q \in \mathcal{P}$ deux entiers premiers entre eux. Une notion importante dans NTRU est le calcul de l'inverse d'un polynôme f dans $\mathcal{P}_p$ et $\mathcal{P}_q$. L'inverse de f dans $\mathcal{P}_p$ est un polynôme $f_p \in \mathcal{P}_p$ tel que

$$f * f_p \equiv 1 \pmod{p}.$$

De même, l'inverse de f dans $\mathcal{P}_q$ est un polynôme $f_q \in \mathcal{P}_q$ vérifiant

$$f * f_q \equiv 1 \pmod{q}.$$

Maple code 6.1.2.

Avec Maple 12, la procédure suivante permet de calculer l'inverse f_p de f dans $\mathcal{P}_p$. Pour l'algorithme de calcul, on peut consulter [21]. On suppose que la procédure **star(N,f,g)** est déjà exécutée.

Programme (Maple)

```
invp := proc (N, p, a)
with(MatrixPolynomialAlgebra);
k := 0;
b := 1;
c := 0;
f := a;
g := X^N-1;
while true do
    f0 := coeff(f, X, 0);
    while f0 = 0 do
        f := expand(f/X);
        c := expand(c*X);
        k := k+1;
        f0 := coeff(f, X, 0)
    end do;
    if degree(f) = 0 then
        b := `mod`(b/f0, p);
        fp := star(N, b, X^(`mod`(-k, N)));
        break
    end if;
    if degree(f) < degree(g) then
        t := f; f := g; g := t;
        t := b; b := c; c := t
    end if;
    f0 := coeff(f, X, 0);
    g0 := coeff(g, X, 0);
    u := `mod`(f0/g0, p);
    f := `mod`(f-u*g, p);
    b := `mod`(b-u*c, p)
end do;
return fp;
end proc:
```

Pour illustrer cette procédure, on considère l'exemple suivant $N = 11$, $p = 3$ et $f = 1 + X - X^3 + X^5 - X^6$. Le programme suivant permet de calculer f_p dans $\mathcal{P}_p$.

Programme (Maple)	Commentaires
N:=11;	
p:=3;	
f:=1+X-X^3+X^5-X^6;	
fp:=invp(N,p,f);	<-- fp=2X^8+2X^7+2X^6+X^5+2X^4+2*X^3+2X^2+X+2
star(N,f,fp) mod p;	<-- vérification f*fp=1 mod p

De même, le programme ci-dessous permet de calculer l'inverse f_q de f dans $\mathcal{P}_q$. Pour l'algorithme de calcul, on peut aussi consulter [21]. On suppose que les procédures **star(N,f,g)** et **inv(N,p,f)** sont déjà exécutées.

```
      Programme  (Maple)

invq := proc (N,p,r,a)
q := p^r;
b := invp(N, p, a);
qq := p;
while qq < q do
    qq := qq*p;
    b := `mod`(star(N, b, 2-star(N, a, b)), q)
end do;
return b;
 end proc:
```

Pour illustrer cette procédure, on considère l'exemple suivant $N = 11$, $p = 2$, $r = 3$ et $f = 1 + X - X^3 + X^5 - X^6$. Le programme suivant permet de calculer f_q dans $\mathcal{P}_q$.

Programme (Maple)	Commentaires
N:=11;	
p:=2;	
r:=3;	
f:=1+X-X^3+X^5-X^6;	
fq:=invp(N,p,r,f);	<-- fq=X^10+7X^9+7X^8+7X^7+5X^6+2X^5+3X^4+4X^3+X+4
star(N,f,fq) mod q;	<-- vérification f*fq=1 mod q

Maple code 6.1.3.

The following procedure transforms a polynomial f modulo q to a centered polynomial f modulo q, that is with coefficients in the interval $\left[-\frac{q}{2}, \frac{q}{2}\right[$

```
   Programme  (Maple)

center:=proc(N,f,q)
local i,ci,di,g;
g:=0;
for i from 0 to N-1 do
 ci:=coeff(f,X,i);
 di:=ci mod q;
 if di>q/2 then
    di:=di-q;
 end if;
 g:=g+di*X^i;
end do;
return(g);
end proc:
```

Maple code 6.1.4.
Le programme ci-dessous permet de vérifier la validité de NTRU à l'aide de Maple. On suppose que les procédure **star**, **invp** et **invq** sont déjà initialisées.

```
      Programme  (Maple)

N := 13: q := 8: p := 3:
f := X^12+X^11+X^10+X^9+X^8+X^7+1:
g := X^12+X^5-X^4+X^3-X^2+X-1:
fp := invp(N, p, f):
fq := invq(N, 2, 3, f):
h := `mod`(star(N, fq, g), q):
m := X^10+X^8+X^7+X^4+X^3+1:
r := X^12+X^11+X^8+X^7+1:
e := `mod`(star(N, p*r, h)+m, q):
a := `mod`(star(N, f, e), q):
a:=center(N,a,q);
m2 := `mod`(star(N, fp, a), p):
m2:=center(N,m2,p);
m-m2;
```

La différence entre le message d'origine m est le message $m2$, obtenu après décryptage est nulle, ce qui confirme l'exactitude de NTRU.

6.2 Application de LLL à NTRU

Juste après la publication de NTRU, Coppersmith et Shamir [5] ont proposé une attaque contre la clé publique pour les petites valeurs du paramètre N. Dans NTRU, la clé

publique $h \equiv f_q * g \pmod q$ vérifie $f * h \equiv g \pmod q$. Alors, il existe un polynôme $u \in \mathcal{P} = \mathbb{Z}/\left(X^N - 1\right)$ tel que

$$f * h - q * u = g.$$

On considère alors l'ensemble

$$\mathcal{L} = \left\{(f, g) \in \mathcal{P}^2 \mid \exists u \in \mathcal{P},\ f * h - q * u = g\right\}.$$

On vérifie facilement que $\mathcal{L}$ est un réseau et sous forme matricielle, ceci s'écrit sous la forme :

$$(f, -u) * \begin{bmatrix} 1 & h \\ 0 & q \end{bmatrix} = (f, g).$$

Avec les coordonnées de f, g et h, l'égalité ci-dessus prend la forme

$$\left[\begin{array}{c} f_0 \\ f_1 \\ \vdots \\ \hline\hline f_{N-1} \\ \hline -u_1 \\ -u_2 \\ \vdots \\ -u_{N-1} \end{array}\right] * \left[\begin{array}{cccc||cccc} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & 1 & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ \hline\hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right] = \left[\begin{array}{c} f_0 \\ f_1 \\ \vdots \\ \hline\hline f_{N-1} \\ \hline g_0 \\ g_1 \\ \vdots \\ g_{N-1} \end{array}\right]$$

Ainsi, la matrice

$$M(L) = \begin{bmatrix} I_N & h_N \\ 0 & qI_N \end{bmatrix}$$

du réseau vérifie $\det(M(L)) = q^N$. D'autre part, on suppose que $f \in \mathcal{T}(d_f, d_f - 1)$ et $g \in \mathcal{T}(d_g, d_g)$ avec $d_f = d_g \approx \frac{N}{3}$. Alors

$$\|(f, g)\| = \left(\sum_{i=0}^{N-1} f_i^2 + \sum_{i=0}^{N-1} g_i^2\right)^{1/2} \approx \sqrt{4d_f} \approx 2\sqrt{\frac{N}{3}}.$$

En appliquant l'algorithme LLL au réseau $\mathcal{L}$, on produit un vecteur v_1 assez court, plus précisement

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

Ainsi, on peut avoir $v_1 = (f, g)$ si la condition suivante est vérifiée:

$$2\sqrt{\frac{N}{3}} \leq 2^{\frac{2N-1}{4}} \det(L)^{\frac{1}{2N}} \quad \Longleftrightarrow \quad 2^{-\frac{2N-5}{2}} N \leq 3q.$$

Pour illustrer l'utilisation de LLL, on prend l'exemple ci-dessous:

$$\begin{aligned}
N &= 13, \quad p = 3 \quad q = 8, \\
f &= 1 - X + X^{12}, \\
f_p &= 1 + 2X + 2X^2 + 2X^4 + X^5 + X^6 + X^8 + 2X^9 + 2X^10 + 2X^12, \\
f_q &= 1 + X + X^3 + 7X^4 + 2X^5 + 5X^6 + 5X^7 + 5X^9 + 3X^10 + 2X^11 + X^12, \\
g &= 1 + X - X^8 + X^{11}, \\
h &= 6X + 3X^2 + 3X^3 + 3X^5 + 5X^6 + 6X^7 + 7X^8 + 6X^9 + X^{10} + 5X^11 + 5X^{12} \pmod{q}.
\end{aligned}$$

La matrice du réseau est alors

$$M(L) = \begin{bmatrix} I_N & h_N \\ 0_N & qI_N \end{bmatrix}$$

où 0_N est la matrice carrée nulle, I_N est la matrice unité et h_N est la matrice circulaire. En appliquant l'algorithme LLL à la matrice $M(L)$, on obtient une matrice sous la forme

$$\begin{bmatrix} M_f & M_g \end{bmatrix}.$$

Si l'algorithme LLL réussit, les coefficients de f sont parmi les lignes de M_f et ceux de g sont parmi les lignes de M_g. Les lignes 26 de M_f et de M_g nous donnent

$$\begin{aligned}
f(LLL) &= 1 - X + X^{12} = f, \\
g(LLL) &= 1 + X - X^8 + X^{11} = g.
\end{aligned}$$

On retrouve ainsi les clés privées f et g qui ont permi de fabriquer la clé plublique h.

Maple code 6.2.1.

Dans Maple 12, le programme ci-dessous permet de retrouver f et g à partir de h.

Programme (Maple)	Commentaires
`with(IntegerRelations):`	`<--- Package LLL:`
`N := 13;`	`<--- N= 13`
`q := 8;`	`<--- q = 8`
`P := 6*X+3*X^2+3*X^3+3*X^5+5*X^6+6*X^7`	`<--- Polynôme h`
`+7*X^8+6*X^9+X^10+5*X^11+5*X^12;`	
`for i from 0 to N-1 do`	`<--- Coefficients de h`
`h[i] := coeff(P, X, i)`	
`end do:`	
`delta := proc (i, j)`	`<--- delta(i,j)=1 si i=j`
`if i = j then return 1`	`delta(i,j)=0 sinon`
`else return 0`	
`end if end proc:`	
`for i from 0 to N-1 do`	`<--- Matrice I_N`
`l[i+1]:=seq(delta(i, j),j=0..N-1)`	
`end do:`	
`for i from 0 to N-1 do`	`<--- Matrice h_N`
`ll[i+1]:=seq(h[‘mod‘(N-i+j, N)],j=0..N-1)`	
`end do:`	
`for i from 0 to N-1 do`	`<--- Matrice [I_N H_N]`
`b[i+1]:=[l[i+1],ll[i+1]]`	
`end do:`	
`delta2 := proc (i, j)`	`<--- Matrice 0_N`
`if i = j then return q`	
`else return 0`	
`end if`	
`end proc:`	
`for i from 0 to N-1 do`	`<--- Matrice q_N`
`l[N+i+1] := seq(delta2(i, j),j=0..N-1)`	
`end do:`	
`for i from 0 to N-1 do`	`<--- Matrice [0_N q_N]`
`b[N+i+1]:=[seq(0,j=0..N-1),l[N+i+1]]`	
`end do:`	
`ML := seq(b[i+1], i = 0 .. 2*N-1):`	`<--- Matrice ML`
`MLL := LLL([ML], 'integer');`	`<--- Application de LLL`

Pour vérifier que toutes les lignes sont des solutions, on peut appliquer le programme ci-dessous et remarquer que $f * P - g \equiv 0 \pmod{q}$.

```
      Programme  (Maple)

for i from 1 to 2*N do
    f:=0;
    for j from 1 to N do
        f:=f+MLL[i,j]*X^(j-1);
    end do;
    g:=0;
    for j from N+1 to 2*N do
        g:=g+MLL[i,j]*X^(j-N-1);
    end do;
    dif:=(star(N, P, f)-g) mod q;
    if (dif=0) then print(i,dif);end if;
end do:
```

Chapter 7

Le Cryptosystème Knapsack de Merkle et Hellman

7.1 Introduction au problème du sac à dos

Soit $n \geq 2$ un nombre entier. Soient a_i, $1 \leq i \leq n$, et S des nombres entiers fixés. Le problème du sac à dos consiste en la recherche d'une solution $(x_1, \cdots, x_k) \in \{0,1\}^n$ de l'équation:

$$a_1 x_1 + \cdots + a_n x_n = S.$$

Definition 7.1.1. Une suite supercroissante est une suite $(a_1, \cdots, a_n)$ telle que pour tout k avec $2 \leq k \leq n$, on a $b_k > \sum_{i=1}^{k-1} b_i$.

Proposition 7.1.2. *Soit $(a_1, \cdots, a_n)$ une suite supercroissante et S un entier tel que $S = a_1 x_1 + \cdots + a_n x_n$ avec $x_i \in \{0,1\}$. Alors l'algorithme suivant permet de déterminer la solution $(x_1, \cdots, x_n) \in \{0,1\}^n$.*

Maple code 7.1.3.

Dans Maple 12, le programme suivant permet déterminer la solution de l'équation $S = a_1 x_1 + \cdots + a_n x_n$ avec $x_i \in \{0,1\}$ où $(a_1, \cdots, a_n)$ est une suite supercroissante. Dans ce programme, on pose $a = \{a_1, \cdots, a_n\}$.

Algorithm 13 : Résolutions du problème sac à dos pour une suite supercroissante

Input : La suite supercroissante $(a_1, \cdots, a_n)$ et un entier $S = a_1x_1 + \cdots + a_nx_n$ avec $x_i \in \{0, 1\}$.

Output : La solution $(x_1, \cdots, x_n) \in \{0, 1\}^n$.

1: $i = n$.
2: **While** $i \geq 1$ **do**
3: **If** $S \geq a_i$ **then**
4: $x_i = 1$,
5: $S = S - a_i$,
6: **Else**
7: $x_i = 0$.
8: **End If**
9: $i = i - 1$.
10: **End While**
11: Sortir $(x_1, \cdots, x_n)$.

Programme (Maple 12)	Commentaires
`sol := proc (a, S)`	`<--- procédure sol`
`n := nops(a);`	`<--- n:=nombre d'éléments de a`
`x := [seq(0, i = 1 .. n)];`	`<--- Changement de variable;`
`Z := S; i := n;`	`<--- initialisation de i`
`while 0 < i do`	`<--- Début de la boucle`
`if a[i] <= Z then`	`<--- Test`
`x[i] := 1;`	`<--- x[i] = 1`
`Z := Z-a[i]`	`<--- S := S-a[i]`
`else x[i] := 0`	`<--- Sinon x[i] := 0`
`end if;`	`<--- Fin du test`
`i := i-1`	`<--- Décrémentation de i`
`end do;`	`<--- Fin de la boucle`
`return x;`	`<--- Sortie de la solution`
`end proc:`	

On peut tester ce programme avec l'exemple suivant

$$a = \{4, 6, 11, 25, 50, 110\}, \quad S = a[1] + a[2] + a[3] + a[6] = 131.$$

Programme (Maple 12)	Commentaires
`a:={4, 6, 11, 25, 50, 110};`	`<--- Liste a`
`S:=a[1]+a[2]+a[3]+a[6];`	`<--- Expression de S`
`sol(a,S);`	`<--- procédure sol(a,S);`

On obtient alors la solution $(1, 1, 1, 0, 0, 1)$.

7.2 Le cryptosysteme de Merkle et Hellman

Pour utliser le cryptosystème Merkle et Hellman par deux entités **A** et **B**, ils doivent procéder comme suit, où on suppose que **B** veut envoyer un message M à **B**. Pour celà, **A** doit préparer ses clés secrètes et publiques.

Génération des clés par A:

1. Choisir une suite super croissante $(a_1, \cdots, a_n)$.
2. Choisir un entier N tel que $N > \sum_{i=1}^{n} a_i$.
3. Choisir un entier d tel que $\gcd(d, N) = 1$.
4. Calculer $e_i = da_i \pmod N$ pour $1 \leq i \leq n$.
5. Publier la clé publique $(e_1, \cdots, e_n)$.
6. Conserver la clé secrète $(N, d, (a_1, \cdots, a_n))$.

Maintenant, **B** peut coder et envoyer son message M.

Chiffrement par B:

1. Transformer le message M en une suite binaire $X = (x_1, \cdots, x_n) \in \{0, 1\}^n$.
2. Calculer $S = \sum_{i=1}^{n} x_i e_i$.
3. Envoyer le message S à **A**.

En recevant le message chiffré S, **A** peut alors le déchiffrer.

Déchiffrement par A:

1. Calculer $S' \equiv Sd^{-1} \equiv \pmod N$.
2. Résoudre l'équation $S' = \sum_{i=1}^{n} x_i a_i$.
3. Calculer $M = \sum_{i=1}^{n} x_i 2^i$.

Maple code 7.2.1.

Avec Maple 12, le programme suivant illustre une utilisation du cryptosystème de Merkle et Hellman. Dans ce programme, on pose $a = \{a_1, \cdots, a_n\}$, et on suppose que la procédure **sol** est à été exécutée.

Programme (Maple 12)	Commentaires
`#Génération des clés par A:`	`<--- Génération des clés par A:`
`a := [4, 6, 11, 25, 50, 110];`	`<--- Choix de la suite a`
`n := nops(a);`	`<--- n := #a`
`T := 0:`	`<--- Initialisation de T`
`for i from 1 to n do`	`<--- T=somme des termes de a`
`  T := T+a[i]`	
`end do:`	
`N := (rand(T .. 2*T))();`	`<--- Choix d'un nombre`
`d:=N:`	`  aléatoire dans [T,2T]`
`while gcd(d,N)>1 do`	`<--- Création d'un nombre d`
`  d:=rand(2..N-1)():`	
`end do:`	
`e:=[seq(mod(d*a[i],N),i=1..n)];`	`<--- Création de la suite e`
`# Chiffrement par B`	`<--- Chiffrement par B`
`M:=61;`	`<--- Choix du message M`
`a2:=[seq(2^i, i = 0 .. n-1)];`	`<--- Suite a2=(1,2,...2^{n-1}`
`X :=sol(a2, M);`	`<--- Décomposition de M dans a2`
`S := 0:`	`<--- Initialisation de S`
`for i from 1 to n do`	`<--- S=somme X*e`
`  S := S+X[i]*e[i];`	
`end do:`	
`# Déchiffrement par A`	`<--- Déchiffrement par A`
`S2:=(S*modp(1/d, N)) mod N;`	`<--- Calcul de S2=S/d mod N`
`X2 := sol(a, S2);`	`<--- X2=suite binaire de S2 dans a`
`M2 := 0:`	`<--- Initialisation de M2`
`for i from 0 to n-1 do`	`<--- Calcul du nombre`
`  M2 := M2+X2[i+1]*2^i`	`  M2=somme X2[i+1]*2^i`
`end do:`	
`M2-M;`	`<--- Vérification de l'égalité M2=M`

7.3 Application de LLL au problème du sac à dos

Supposons que les entités **A** et **B** sont entrain d'utliser le cryptosystème de Merkle et Hellman. Les paramètres publiques sont donc :

- La clé publique $(e_1, \cdots, e_n)$ de **A**.
- Le message S envoyé par **B** à **A**.

On sait aussi que ces paramètres vérifient une relation secrète $S = \sum_{i=1}^{n} x_i e_i$ pour une suite $(x_1, \cdots, x_n) \in \{0,1\}^n$. On considère le vecteur

$$U = \left(x_1, x_2, \cdots, x_n, \sum_{i=1}^{n} x_i e_i - S \right).$$

On peut écrire alors le vecteur U sous la forme

$$U = \sum_{i=1}^{n} x_i b_i + b_{n+1},$$

avec

$$\begin{array}{rcl} b_1 & = & (1,0,0,\cdots,0,e_1 C), \\ b_2 & = & (0,1,0,\cdots,0,e_2 C), \\ b_3 & = & (0,0,1,\cdots,0,e_3 C), \\ \vdots & \vdots & \vdots \\ b_n & = & (0,0,0,\cdots,1,e_n C), \\ b_{n+1} & = & (0,0,0,\cdots,0,-SC), \end{array}$$

où C est un nombre à déterminer plus tard pour garantir le succés de la méthode. Ceci nous amène à considérer le résau L défini par:

$$L := \left\{ \sum_{i=1}^{n+1} a_i b_i \mid a_i \in \mathbb{Z} \right\}.$$

La matrice associé à ce résau est alors:

$$M(L) = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & e_1 C \\ 0 & 1 & 0 & \cdots & 0 & e_2 C \\ 0 & 0 & 1 & \cdots & 0 & e_3 C \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & 0 & \vdots & 1 & e_n C \\ 0 & 0 & 0 & \vdots & 0 & -SC \end{pmatrix}.$$

En réduisant le réseau, on peut alors déterminer un vecteur court $V = (y_1, \cdots, y_n, y_{n+1})$. Pour que la suite $(y_1, \cdots, y_n)$ soit une solution pour le problème, on doit avoir $\sum_{i=1}^{n} y_i e_i = S$ et donc $y_{n+1} = 0$, ce qui donne les conditions:

$$\begin{cases} |y_{n+1}| = 0 \\ |y_i| \leq 1 \quad \text{pour } 1 \leq i \leq n \\ \sum_{i=1}^{n} y_i e_i = S. \end{cases} \tag{7.1}$$

Pour illustrer l'application de l'algorithme au problème du sac à dos, on prend l'exemple:

$$e = [205, 119, 281, 56, 112, 171], \qquad S = 825.$$

La matrice formée par les vecteurs de la base du réseau L est donc sous la forme

$$M(L) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 205C \\ 0 & 1 & 0 & 0 & 0 & 0 & 119C \\ 0 & 0 & 1 & 0 & 0 & 0 & 281C \\ 0 & 0 & 0 & 1 & 0 & 0 & 56C \\ 0 & 0 & 0 & 0 & 1 & 0 & 112C \\ 0 & 0 & 0 & 0 & 0 & 1 & 171C \\ 0 & 0 & 0 & 0 & 0 & 0 & 825C \end{pmatrix}.$$

En appliquant l'algorithme LLL sous Maple 12 avec C=10, on obtient la matrice

$$\begin{pmatrix} 0 & 0 & 0 & -2 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ -1 & -3 & 2 & 0 & 0 & 0 & 0 \\ 2 & -3 & 0 & 0 & -2 & 1 & 0 \\ 1 & -2 & -2 & 0 & 1 & -2 & 0 \\ -1 & -1 & -3 & 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & -1 & -2 & 0 & 10 \end{pmatrix}.$$

Seul le vecteur $(1, 0, 1, 1, 1, 1, 0)$ vérifie les conditions 7.1 et donne

$$(x_1, x_2, x_3, x_4, x_5, x_6) = (1, 0, 1, 1, 1, 1),$$

ce qui permet d'avoir $205 + 281 + 56 + 112 + 171 = 825$.

Maple code 7.3.1.

Avec Maple 12, l'exemple ci-dessus peut être réalisé comme ci-dessous.

Programme (Maple 12)	Commentaires
`restart;`	
`with(IntegerRelations):`	`<--- Package LLL et PSLQ:`
`with(LinearAlgebra):`	`<--- Matrices et vecteurs`
`n:=6;`	`<--- Nombre d'inconnues`
`S:=825;`	`<--- Valeur de S`
`C:=10;`	`<--- Valeur de C`
`e[1]:=205*C;e[2]:=119*C;e[3]:=281*C;`	`<--- Valeurs de la suite e`
`e[4]:=56*C;e[5]:=112*C;e[6]:=171*C;`	
`e[7]:=-S*C;`	
`for i to n+1 do`	`<--- Construction de`
`  for j to n+1 do`	`<--- la base b`
`  if i=j then b[i, j]:=1;`	
`  else b[i, j]:=0 ;`	
`  end if;`	
`  end do;`	
`end do;`	`<--- Fin`
`for i to n+1 do`	`<--- Construction de`
`  b[i, n+1]:=e[i]`	`  la base b`
`end do;`	
`for i to n+1 do`	
`  for j to n+1 do`	
`  b[i]:=[seq(b[i, j],j=1..n+1)];`	
`  end do;`	
`end do;`	`<--- Fin`
`ML:=[seq(b[i], i = 1 .. n+1)];`	`<--- La matrice M(L)`
`N:=LLL(ML,'integer');`	`<--- N=Application de LLL`
`i := 0;`	`<--- Initialisation`
`while i < n+1 do`	`<--- Vérification des`
`  i:=i+1;`	`<--- conditions:`
`  if N[i,n+1]=0 then`	`<--- y_{n+1}>0`
`  j := 0;`	
`  while j<n+1 do`	
`  j:=j+1;`	
`  if abs(N[i, j])>1`	`<--- abs(y_i)=0 ou 1`
`  then j:=n+1`	
`  else if j=n+1 then`	
`print("La solution est dans :",N[i]);`	`<--- La solution`
`  i := n+1`	
`  end if;`	
`  end if;`	
`  end do;`	
`  end if;`	
`end do;`	`<--- Réponse [1,0,1,1,1,1,0]`

Chapter 8

The Lattice Based Cryptosystems GGh and LWE

In this chapter, we give a short introduction to two lattice based cryptosystems, GGH and LWE. While GGH has been broken, LWE is a promising scheme since it is one of the candidates to post quantum cryptography. The chapter is illustrated with many examples and codes using the computer algebra system Maple.

In the following, we list some computational problems that seem to be hard in general and on which some cryptographic systems have been based. An overview of many hard lattice problems and their interconnections is presented in [26].

Definition 8.0.1. Let $\mathcal{L}$ be a full rank lattice of dimension n in $\mathbb{Z}^n$.

1. **The Shortest Vector Problem (SVP):** Given a basis matrix B for $\mathcal{L}$, compute a non-zero vector $v \in \mathcal{L}$ such that $\|v\|$ is minimal, that is $\|v\| = \lambda_1(\mathcal{L})$.

2. **The Closest Vector Problem (CVP):** Given a basis matrix B for $\mathcal{L}$ and a vector $v \notin \mathcal{L}$, find a vector $u \in \mathcal{L}$ such that $\|v - u\|$ is minimal, that is $\|v - u\| = \mathrm{d}(v, \mathcal{L})$ where $\mathrm{d}(v, \mathcal{L}) = \min_{u \in \mathcal{L}} \|v - u\|$.

3. **The Shortest Independent Vectors Problem (SIVP):** Given a basis matrix B for $\mathcal{L}$, find n linearly independent lattice vectors $v_1, v_2, \ldots, v_n$ such that $\max_i \|v_i\| \leq \lambda_n$, where λ_n is the nth successive minima of $\mathcal{L}$.

4. **The approximate SVP problem (γSVP):** Fix $\gamma > 1$. Given a basis matrix B for $\mathcal{L}$, compute a non-zero vector $v \in \mathcal{L}$ such that $\|v\| \leq \gamma\lambda_1(\mathcal{L})$ where $\lambda_1(\mathcal{L})$ is the minimal Euclidean norm in $\mathcal{L}$.

5. **The approximate CVP problem (γSVP):** Fix $\gamma > 1$. Given a basis matrix B for $\mathcal{L}$ and a vector $v \notin \mathcal{L}$, find a vector $u \in \mathcal{L}$ such that $\|v - u\| \leq \gamma\lambda_1\mathrm{d}(v, \mathcal{L})$ where $\mathrm{d}(v, \mathcal{L}) = \min_{u \in \mathcal{L}} \|v - u\|$.

8.1 GGH

In 1996, Goldreich, Goldwasser and Halevi [9] proposed an new way to use lattice theory to build a public key cryptosystem and digital signature scheme inspired by McEliece cryptosystem. This cryptosystem, called GGH was based on the appproximate Closest Vector Problem γCVP 8.0.1. GGH was brocken by Nguyen [19] in 1999 for low dimensions.

Algorithm 14 : GGH key generation

Input : A lattice $\mathcal{L}$ of dimension n.
Output : A public key B and a private key A.

1: Find a "good basis" A of $\mathcal{L}$.
2: Find a "bad basis" B of $\mathcal{L}$.
3: Publish B as the public key.
4: Keep A as the secret key.

Algorithm 15 : GGH encryption

Input : A lattice $\mathcal{L}$, a parameter $\rho > 0$, a public key B and a plaintext $m \in \mathbb{Z}^n$.
Output : A ciphertext c.

1: Compute $v = mB \in \mathcal{L}$.
2: Choose a small vector $e \in [-\rho, \rho]^n$.
3: The ciphertext is $c = v + e$.

Algorithm 16 : GGH decryption

Input : A lattice $\mathcal{L}$, a private key A and a ciphertext c.
Output : A plaintext $m \in \mathbb{Z}^n$.

1: Use an efficient reduction algorithm and the good basis A to find the closest vector $v \in \mathcal{L}$ of the ciphertext c.
2: Compute $m = vB^{-1}$.

The security of GGH is based on the following assumptions.

- Given a "good basis" A of $\mathcal{L}$, it is easy to find a "bad basis" B of $\mathcal{L}$.
- Given a "bad basis" B of $\mathcal{L}$, it is hard to find a "good basis" A of $\mathcal{L}$.
- Given a "good basis" A of $\mathcal{L}$, it is easy to solve the CVP.
- Given a "bad basis" B of $\mathcal{L}$, it is hard to solve the CVP.

Nevertheless, Ngyuen's attack [19] shows that GGH is vulnerable for lattices with small dimension ($\dim(\mathcal{L}) \leq 350$). Consequently, to ensure security, GGH requires lattices with higher dimension. This makes GGH not practical at all for implementations.

Maple code 8.1.1. To start with Maple, we need to initialize the worksheet with the following packages.

```
      Maple:    Initialization

restart:Digits:=50:
with(numtheory):with(linalg):
with(LinearAlgebra):with(IntegerRelations):
```

Maple code 8.1.2. The following procedure creates a $n \times n$ matrix with large random integer entries and determinant 1.

```
      Maple:    Creating a matrix (n,n) with det=1


matinv:=proc(n)
local M1,M2,M;
M1 := randmatrix(n, n, unimodular):
M2 := transpose(randmatrix(n, n, unimodular)):
M := multiply(M1, M2):
return matrix(M);
end proc:
```

Then, we can check that $\det(M) = 1$.

```
      Maple:    Checking that det(M)=1


n:=4;
M:=matinv(n);
Determinant(M);
```

We get always 1.

Maple code 8.1.3. The following procedure creates a $n \times n$ matrix with nonzero determinant.

```
      Maple:    Creating a matrix (n,n) with det=1

mat:=proc(n)
local dd,M;
M:=randmatrix(n,n);
dd:=Determinant(Matrix(M));
while dd=0 do
    M:=randmatrix(n,n);
    dd:=Determinant(Matrix(M));
end do;
return Matrix(M);
end proc:

mat(5);
```

Maple code 8.1.4. The following procedure creates public key B and a private A as in the GGH key generation process.

```
      Maple:    Public and private keys for GGH

GGHkeys:=proc(n)
local B,A,U,i,b,j,M;
B:=mat(n);
M:=[seq(Row(B,i),i=1..n)];
A:=LLL(M, 'integer');
U:=matinv(n);
B:=multiply(U,A);
return Matrix(A),Matrix(B);
end proc:

n:=4;
key:=GGHkeys(n):
A:=key[1];
B:=key[2];
```

Maple code 8.1.5. The following procedure creates a a random plaintext message $m \in \mathbb{Z}^n$.

```
      Maple:    Random message

message:=proc(n)
local m;
m:=RandomMatrix(1, n, generator = -100 .. 100);
return m;
end proc:

m:=message(n);
```

Maple code 8.1.6. Let A be a GGH private key and B a GGH public key. Let $m \in \mathbb{Z}^n$ be a plaintext. The following procedure creates the ciphertext $c = mB+e$ where $e \in [-\rho, \rho]^n$ B for a fixed ρ.

```
      Maple:    GGH encryption

GGHencrypt:=proc(n,B,m,r)
local v,e,c;
v:=multiply(m,B);
e:=RandomMatrix(1, n, generator = -r .. r);
c:=matadd(v, e);
return Matrix(c);
end proc:

c:=GGHencrypt(n,B,m,2);
```

Maple code 8.1.7. Let A be a GGH private key and B a GGH public key. Let $c \in \mathbb{Z}^n$ be a ciphertext. The following procedure outputs the plaintext m such that $c = mB + e$.

```
      Maple:    GGH decryption

GGHdecrypt:=proc(n, A, B, c)
local T1, T, j, M1, M;
T1:=multiply(c, inverse(A));
T:=[seq(round(T1[1,i]), i = 1 .. n)];
M1:=multiply(T,A);
M:=multiply(M1,inverse(B));
return Matrix(M);
end proc:

mm:=GGHdecrypt(n,A,B,c);
m;
```

8.2 LWE

In this section we describe LWE, the Learning With Errors cryptosystem. LWE was invented by Regev [23] in 2005. It is a lattice-based cryptosystem and is supported by a theoretical proof of security. LWE is parameterized by integers n, m, l, t, r, q and a probability distribution χ over $\mathbb{Z}_q$, typically taken to be a rounded normal distribution.

Maple code 8.2.1. The following Maple code prepares a list LWE=$[n0, m0, l0, t0, r0, q]$ to be used throughout the LWE cryptosystem.

```
      LWE List

n0:=3:
m0:=3:
l0:=6:
t0:=10:
r0:=9:
q0:=23:
LWE:=[n0,m0,l0,t0,r0,q0];
```

The key generation, encryption and decryption algorithms are presented in Algorithm 17, Algorithm 18 and Algorithm 19.

Algorithm 17 : LWE Key Generation

Input : Integers n, m, l, q.
Output : A private key S and a public key (A, P).

1: Choose $S \in \mathbb{Z}_q^{n\times l}$ at random.
2: Choose $A \in \mathbb{Z}_q^{m\times n}$ at random.
3: Choose $E \in \mathbb{Z}_q^{m\times l}$ according to χ.
4: Compute $P = AS + E \pmod q$. Hence $P \in \mathbb{Z}_q^{m\times l}$.
5: The private key is S.
6: The public key is (A, P).

Maple code 8.2.2. The following Maple code computes a private key S and a public key (A, P) to be used by LWE.

```
      LWE Key Generation

lwekey:= proc(L);
local n,m,l,t,r,q,S,A,E,P;
n:=L[1]:m:=L[2]:l:=L[3]:t:=L[4]:r:=L[5]:q:=L[6]:
S:=randmatrix(n,l,entries=rand(0 .. q-1));
A:=randmatrix(m,n,entries=rand(0 .. q-1));
E:=randmatrix(m,l,entries=rand(0 .. 0));
P:=Matrix(multiply(A,S))+Matrix(E) mod q;
return matrix(S),matrix(A),P;
end proc:
```

We choose E to be nil for decryption reason. Here is an example for lwekey.

```
      LWE Key Generation, example

ll:=lwekey(LWE):
S0:=ll[1];
A0:=ll[2];
P0:=ll[3];
```

The output is

$$S = \begin{bmatrix} 17 & -6 & 9 & 8 \\ -9 & -4 & -18 & 16 \\ -16 & -18 & 8 & -21 \end{bmatrix},\ A = \begin{bmatrix} -5 & 6 & -18 \\ -3 & -10 & 15 \end{bmatrix},\ P = \begin{bmatrix} 11 & 6 & 3 & 18 \\ 4 & 18 & 22 & 7 \end{bmatrix}$$

Maple code 8.2.3. In LWE, the message is a vector $M \in \mathbb{Z}_t^{l\times 1}$. The following Maple code outputs a random message.

```
      LWE Random Message Generation

lwemessage:= proc(L)
local n,m,l,t,r,q,M;
n:=L[1]:m:=L[2]:l:=L[3]:t:=L[4]:r:=L[5]:q:=L[6]:
M:=randmatrix(l,1,entries=rand(0 .. t-1));
return matrix(M);
end proc:
```

Here is an example for lwemessage.

```
      LWE Random Message Generation, example

M0:=lwemessage(LWE);
```

The output is

$$M = [8, 1, 8, 4, 9, 6]^T.$$

Algorithm 18 : LWE Encryption

Input : Integers n, m, l, t, r, q, a public key (A, P) and a plaintext $M \in \mathbb{Z}_t^{l\times 1}$.
Output : A ciphertext (u, c).

1: Choose $a \in [-r, r]^{m\times 1}$ at random.
2: Compute $u = A^T a \pmod q \in \mathbb{Z}_q^{n\times 1}$.
3: Compute $c = P^T a + \left[\frac{Mq}{t}\right] \pmod q \in \mathbb{Z}_q^{l\times 1}$.
4: The ciphertext is (u, c).

Maple code 8.2.4. The LWE encryption process can be encoded in Maple as in the following.

```
      LWE Encryption

lweencrypt:=proc(A,P,M,L);
local n,m,l,t,r,q,a,u,c1,c2,c;
n:=L[1]:m:=L[2]:l:=L[3]:t:=L[4]:r:=L[5]:q:=L[6]:
a:=randmatrix(m,1,entries=rand(-r..r));
u:=evalm(transpose(A)&*a);
u:=Matrix(u) mod q;
c1:=evalm(transpose(P)&*a);
c2:=evalm(round(q*M/t));
c:=evalm(c1+c2);
c :=Matrix(c) mod q;
return evalm(u),evalm(c);
end proc:
```

Here is an example

```
      LWE Encryption, an example

ll:=lweencrypt(A0,P0,M0,LWE):
u0:=ll[1];
c0:=ll[2];
```

The output is

Algorithm 19 : LWE Decryption

Input : Integers n, m, l, t, r, q, a private key S and a ciphertext (u, c).
Output : A plaintext M.

1: Compute $v = c - S^T u$ and $M = \left[\frac{tv}{q}\right]$.

Maple code 8.2.5. The LWE encryption process can be encoded in Maple as in the following.

```
      LWE Decryption

lwedecrypt:=proc(S,L,u,c);
local n,m,l,t,r,q,a,c1,c2,v,M;
n:=L[1]:m:=L[2]:l:=L[3]:t:=L[4]:r:=L[5]:q:=L[6]:
c1:=evalm(transpose(S)&*u);
v:=evalm(c-c1);
v:=Matrix(v) mod q;
v :=matrix(v);
M:=evalm(round(t*v/q));
M:=Matrix(M) mod t;
return evalm(M);
end proc:
```

Here is an example for the decryption.

```
      LWE decryption, an example

ll:=lwedecrypt(S0,LWE,u0,c0);
```

The output is

The parameters of LWE are chosen to guarantee the correctness of the decryption. We have

$$\begin{aligned} v &= c - S^T u \\ &= (AS + E)^T a - S^T A^T a + \left[\frac{Mq}{t}\right] \\ &= E^T a + \left[\frac{Mq}{t}\right]. \end{aligned}$$

Hence

$$\left[\frac{tv}{q}\right] = \left[\frac{tE^T a}{q} + \frac{t}{q}\left[\frac{Mq}{t}\right]\right].$$

With suitable parameters, the term $\frac{tE^T a}{q}$ is negligible with high probability according to the form of the probability χ (see [16] for more details). Consequently

$$\left\lceil \frac{tv}{q} \right\rfloor = M.$$

This ensures the correctness of the decryption process.

The security of LWE is based on the following problem:

LWE Problem. Given $P \in \mathbb{Z}_q^{m \times l}$ and $A \in \mathbb{Z}_q^{m \times n}$, find $S \in \mathbb{Z}_q^{n \times l}$ and $E \in \mathbb{Z}_q^{m \times l}$ such that $P = AS + E \pmod{q}$.

In [23], it is shown that the LWE problem can be reduced to special worst-case lattice problems using a quantum algorithm. This means that the LWE problem is hard.

Bibliography

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. Annals of Mathematics, 160(2):781–793, 2004.

[2] J.P. Buhler, H.W. Lenstra, and C. Pomerance, The development of the number field sieve, Volume 1554 of Lecture Notes in Computer Science, Springer-Verlag, 1994, pp. 50–94.

[3] 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).

[4] H. Cohen, A Course in Computational Number Theory, Graduate Texts in Mathematics, Springer, 1993.

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

[6] D. Coppersmith and A. Shamir, Lattice attacks on NTRU. In Advances in cryptology—EUROCRYPT '97, volume 1233 of Lecture Notes in Comput. Sci., pages 52–61. Springer, Berlin, 1997.

[7] W. Diffie, E. Hellman, New Directions in Cryptography, IEEE Trans- actions on Information Theory, 22, 5 (1976), pp. 644–654.

[8] T. El Gamal, A public key cryptosystem and signature scheme based on discrete logarithms. IEEE Transactions on Information Theory IT-31, 496-473,1976.

[9] Goldreich, O., Goldwasser, S., Halevi, S.: Public-Key Cryptosystems from Lattice Reduction Problems. Tech. rep., Massachusetts Institute of Technology (1996).

[10] G. Hanrot, LLL: A Tool for Effective Diophantine Approximation, in: The LLL Algorithm Survey and Applications, Phong Q. Nguyen and Brigitte Vallée Ed. Springer, 2010.

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

[12] J. Hoffstein, J. Pipher, and J. H. Silverman, NTRU: A Ring Based Public Key Cryptosystem in Algorithmic Number Theory. Lecture Notes in Computer Science 1423, Springer-Verlag, pages 267–288, 1998.

[13] J. Hoffstein, J. Pipher, and J. H. Silverman, An Introduction to Mathematical Cryptography, Springer-Verlag, UTM, 2008.

[14] B. King, Mapping an Arbitrary Message to an Elliptic Curve when Defined over GF(2^n), International Journal of Network Security, Vol.8, No.2, PP.169–176, Mar. 2009

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

[16] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput., 37(1): pp. 267–302, 2007. Preliminary version in FOCS 2004.

[17] 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/`

[18] R. C. Merkle, M. E. Hellman, Hiding information and signatures in trap door knapsacks, IEEE Transactions on Information Theory, IT-24(5), pp. 525-530, 1978.

[19] P. Nguyen, Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from crypto'97. In M. Wiener, eds, Advances in Cryptology - CRYPTO'99, volume 1666 of Lecture Notes in Computer Science, pp. 790–790. Springer, 1999.

[20] 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).

[21] J. H. Silverman, Almost Inverses and Fast NTRU Key Creation," Tech. Rep. 14, NTRU Cryptosystems, Inc., March 1999. Version 1.

[22] D. Simon, Selected applications of LLL in number theory (2008) `www.math.unicaen.fr/~simon/maths/lll25_Simon.pdf`.

[23] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, STOC 2005, ACM (2005) pp. 84–93.

[24] P.W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Computing 26, pp. 1484-1509 (1997).

[25] J.H. Silverman, The Arithmetic of Elliptic Curves, Graduate Texts in Mathematics, Springer-Verlag, 106 (1986)

[26] T. Laarhoven, J. van de Pol, B. de Weger, Solving hard lattice problems and the security of lattice-based cryptosystems, Cryptology ePrint Archive, No. 2012/533 (2012).

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