1 Cryptographie, exemple de l'algorithme RSA

1.1 Origine

Créateurs : Rivest, Shamir et Adleman d'où RSA. L'ensemble des résultats ci-dessous énoncés est présenté de manière détaillée et complète dans [Cormen *et al.*, 2001, pp. 849-905]. On y trouvera notamment toutes les preuves des propriétés ici mises en œuvre.

1.2 Nature et principe

Classe des algorithmes de cryptographie « basée clef » et asymétrique (aussi appelée cryptographie de clef publique/privée ou encore publique/secrète).

1.3 Notation mathématique, congruence

Par la suite on écrit de manière équivalente $(a * b) \bmod n = c$ et $(a * b) \equiv c \pmod n$. L'exemple qui précède concerne de façon anecdotique la multiplication mais nous sommes amenés dans la suite du chapitre à utiliser l'une ou l'autre des notations.

1.4 Fonction phi d'Euler

$n > 0$, la fonction Φ d'Euler est $\Phi(n) = n * \prod_{p_i}(1-1/p_i)$

où p_i (> 1) est la suite des nombres premiers[1] divisant n dont n si celui-ci est premier. En conséquence, si n est premier alors $\Phi(n) = n - 1$.

Exemples :

- $\Phi(45) = 45 * (1 - 1/3) * (1 - 1/5) = 24$
- $\Phi(47) = 47 * (1 - 1/47) = 46$

24 et 46 ne sont pas des chiffres quelconques. 24 signifie qu'il y a 24 nombres (le nombre 1 est inclus) qui sont premiers avec 45. Si x est un de ces 24 nombres, par définition pgcd(45,x) = 1.

1.5 Chiffrement

- $\text{Message}_{\text{codé}} = M^e \bmod n$ où M est le message à envoyer, considéré sous une forme numérique ;
- $n = p * q$ où p et q ($p \neq q$) sont des nombres premiers à choisir. On a par définition $\Phi(n) = p * q * (1 - 1/p) * (1 - 1/q) = (p - 1) * (q - 1)$;
- e un entier (impair[2] et « petit ») qui est premier avec $\Phi(n)$. On remarque que si l'on choisit un nombre premier, il n'y a qu'à vérifier que e ne divise pas $\Phi(n)$ pour qu'ils soient premiers entre eux ;
- d tel que $(e * d) \bmod \Phi(n) = 1$. Autrement dit, d est l'inverse de e modulo $\Phi(n)$.

1.6 Exemple numérique de chiffrement

- $p = 11$, $q = 23$
- $n = p * q = 253$
- $\Phi(n) = (p - 1) * (q - 1) = 220$
- $e = 29$ par exemple car pgcd(220,29) = 1
- $d = 129$ puisque $(29 * 129) \bmod 220 = 1$
- $\text{Message}_{\text{codé}} = M^{29} \bmod 253$

[1] Dans [Cormen *et al.*, 2001], l'expression « nombre premier » exclut le nombre 1 (voir page 851). En conséquence, « $p > 1$ » est sous-entendu dans le calcul de la fonction Euler.

[2] Voir justification ci-après.

On choisit par exemple M = ‘J’ c’est-à-dire la valeur décimale 74 d’où $74^{29} \bmod 253 = 40$

1.7 Déchiffrement

$M = Message_{codé}^{d} \bmod n$

1.8 Exemple numérique de déchiffrement

$40^{129} \bmod 253 = 74$

1.9 Principe d’infaillibilité

Il n’existe pas d’algorithme réellement efficace (réaliste) pour calculer la décomposition canonique ou « en facteurs premiers » d’un « grand » entier :

Soit n un entier positif, $n = p_1^{a1} * p_2^{a2} * \ldots * p_k^{ak}$

Où les p_i sont des nombres premiers tels que $p_1 < p_2 < \ldots < p_k \leq n$ et où les a_i sont des nombres positifs.

Exemple : $n = 100 = 2^2 * 5^2$

Par le fait que n dans la technique de chiffrement ci-dessus soit le produit de 2 nombres premiers, on voit que sa fonction Φ est très grande et que donc sa décomposition canonique d’autant plus complexe que p et q sont choisis grands.

1.10 Euclide étendu

On pose pgcd(0,0) = 0 et on a par ailleurs pgcd(a,b) = pgcd(|a|,|b|), pgcd(a,0) = |a|

1.10.1.1 Euclide :

pgcd(a,b) = pgcd(b,a mod b) avec $a \geq 0$ et $b > 0$

1.10.1.2 Euclide étendu :

$d = pgcd(a,b) = a * x + b * y$ avec $a > 0$ et $b > 0$ et $(x,y) \in Z^2$

1.10.1.3 Récursivité :

d = pgcd(a,0) = a * 1 + b * 0
d’ = b * x’ + (a mob b) * y’ = d = a * x + b * y
x = y’ et $y = x' - \lfloor a/b \rfloor * y'$ est une solution car « a mod b = $a - \lfloor a / b \rfloor * b$ »
De manière évidente : $b * x' + (a - \lfloor a / b \rfloor * b) * y' = a * y' + b * (x' - \lfloor a/b \rfloor * y')$

1.11 Equation « a * x ≡ b (mod n) » avec a > 0 et n > 0

Cette équation a zéro, une ou plusieurs solutions, comme par exemple « 13 * x ≡ 1 (mod 18) » qui a une et une seule solution : l’inverse de 13 dans Z^*_{18}, en l’occurrence x = 7.

1.11.1.1 Propriétés remarquables [Cormen *et al.*, 2001, pp. 869-870] :

« a * x ≡ b (mod n) » a des solutions si et seulement si gcd = pgcd(a,n) divise b, autrement dit pgcd(gcd,b) = gcd

« a * x ≡ b (mod n) » possède « gcd » solutions ou « zéro » solution

Si gcd > 0 et gcd = a * x’ + n * y’ alors $x_0 = x' * (b / gcd) \bmod n$ est une solution (multiplication modulo n)

1.11.1.2 Exemple numérique :

13 * x ≡ 1 (mod 18)
pgcd(13,18) = 1 // 1 solution
1 = 13 * 7 + 18 * (-5) // voir Euclide étendu
$x_0 = 7 * (1 / 1) \bmod 18 = 7 * 1 \pmod{18} = 7$

1.11.1.3 Autre exemple numérique :

$14 * x \equiv 30 \pmod{100}$ // [Cormen *et al.*, 2001, p. 871]
pgcd(14,100) = 2
$2 = 14 * (-7) + 100 * 1$
$x_0 = (-7) * (30 / 2) \bmod 100 = (-7) * 15 \bmod 100 = 95$ ($[95]_{100} = [-5]_{100}$ car $-5 = 95 + (-1) * 100$)

1.11.1.4 Application à RSA

Dans le cadre de la détermination de d via la formule $(e * d) \equiv 1 \pmod{\Phi(n)}$, on a $d = x_0$

1.12 Arithmétique modulaire

L'expression « $Message_{codé} = M^e \bmod n$ » est basée sur l'existence d'un groupe abélien fini Z^*_n associé à un entier n positif. Cet ensemble est composé d'entiers positifs premiers avec n (propriété nécessaire mais non suffisante) et strictement inférieurs à n. L'important ici est l'existence d'un inverse (unique) pour la multiplication (*) définie sur ce groupe (1 est l'identité). Par exemple pour tout x appartenant à $Z^*_{18} = \{1, 5, 7, 11, 13, 17\}$, on a pgcd(18,x) = 1. On élimine néanmoins de Z^*_n les nombres congrus entre eux modulo n en ayant pour principe de conserver le plus petit comme représentant (représentant de sa classe d'équivalence en fait). Pour n = 18, les classes d'équivalence sont (entre []) :

Reste 0 : {1, 2, 3, 6, 9}	[1]
Reste 1 : {17}	[17]
Reste 2 : {4, 8, 16}	[4] mais éliminé car $pgcd(18,4) = 2 \neq 1$
Reste 3 : {5, 15}	[5]
Reste 4 : {7, 14}	[7]
Reste 5 : {13}	[13]
Reste 6 : {12}	[12] mais éliminé car $pgcd(18,12) = 3 \neq 1$
Reste 7 : {11}	[11]
Reste 8 : {10}	[10] mais éliminé car $pgcd(18,10) = 2 \neq 1$

On a par exemple $5 * 11 \pmod{18} = 1$ ce qui signifie que 5 et 11 sont inverses (mod 18). Une propriété remarquable est que $|Z^*_n| = \Phi(n)$

Note : par la suite, on utilise aussi Z_n qui correspond par convention à $\{0, 1, \ldots, n - 1\}$

1.13 Exponentiation modulo

On s'intéresse à $x^i \pmod{n}$ pour $x \in Z^*_n$ et $i > 0$. On a $x^0 \pmod{n} = 1$.

1.13.1.1 Exemple numérique :

$5^0 \pmod{18} = 1$
$5^1 \pmod{18} = 5$
$5^2 \pmod{18} = 7$
$5^3 \pmod{18} = 17$
$5^4 \pmod{18} = 13$
$5^5 \pmod{18} = 11$
$5^6 \pmod{18} = 1$

1.13.1.2 Autre exemple numérique :

$7^0 \pmod{18} = 1$
$7^1 \pmod{18} = 7$
$7^2 \pmod{18} = 13$
$7^3 \pmod{18} = 1$
$7^4 \pmod{18} = 7$
$7^5 \pmod{18} = 13$

1.13.1.3 Générateur, groupe cyclique

La différence entre les puissances de 5 et les puissances de 7, c'est que celles de 5 décrivent complètement Z^*_{18}. 5 est appelé générateur de Z^*_{18}. Par ailleurs un groupe ayant au moins un générateur est un groupe cyclique. Z^*_{18} est un groupe cyclique alors que Z^*_{12} par exemple, ne l'est pas (à vérifier expérimentalement).

1.13.1.4 Théorème d'Euler

$x^{\varphi(n)} \pmod n = 1$ pour tout $x \in Z^*_n$

1.13.1.5 Théorème de Fermat

$x^{n-1} \pmod n = 1$ si n est premier et pour tout $x \in Z^*_n$

Rappel : si n est premier alors $\Phi(n) = n - 1$

1.13.1.6 Théorème du logarithme discret (groupes cycliques)

Le logarithme discret ou index de 1, base 5 (ou générateur 5), dans Z^*_{18} est 0, c'est-à-dire l'exposant nécessaire à 5 pour obtenir 1. On note $ind_{18,5}(1) = 0$. On a $x^a \equiv x^b \pmod n$ avec x un générateur Z^*_n, a des solutions si et seulement si $a \equiv b \pmod{\varphi(n)}$.

L'idée est de trouver quels pourraient être les i tels que Z^*_i soit cyclique : en l'occurrence, si i = 2, i = 4, $i = p^e$ et $i = 2 * p^e$ (p > 2 et premier ainsi que e > 0) alors Z^*_i est cyclique.

Remarque : on va utiliser ce théorème pour l'exponentiation modulo. Ainsi $74^{29} \bmod 253$ revient à s'intéresser à 74^{16}, 74^8, 74^4 et 74^1 (voir programme Java).

1.14 Attention !

M < n ! Il est nécessaire que le message vu sous une forme numérique soit inférieur à n. En effet, les propriétés ci-dessus sont vérifiées pour $x \in Z_n$ impliquant x < n donc $M \in \{0, 1, \ldots, n-1\}$

1.15 Processus général de RSA

La clef publique est par définition la paire P = (e,n)
La clef secrète est par définition la paire S = (d,n)
$P(A) = A^e \bmod n$
$S(B) = B^d \bmod n$
De manière plus générique, $P(S(X) = = X^{d*e} \pmod n = S(P(X)) = X^{e*d} \pmod n$

1.16 Démonstration

Il faut montrer que $X^{e*d} \pmod p = X * (X^{p-1})^{k*(q-1)} \pmod p$ en s'appuyant sur le fait que $e * d = 1 + k * (p-1) * (q-1)$ avec k entier, *i.e.* e et d sont inverses modulo $\varphi(n) = (p-1) * (q-1)$.

$X^{e*d} \pmod p = X * (1)^{k*(q-1)} \pmod p$ car en effet $X^{p-1} \pmod p = 1$ par le théorème de Fermat

On montre de manière symétrique que $X^{e*d} \pmod q = X$

Reste à montrer que $X^{e*d} \pmod n = X$. Cela découle du théorème du « reste chinois » [Cormen *et al.*, 2001, pp. 873-876]. Appliqué à RSA, cela donne : $x \equiv y \pmod p$ et $x \equiv y \pmod q$ si et seulement si $x \equiv y \pmod n$ et $n = p * q$ avec p et q premiers entre eux. Note : x et y sont des entiers quelconques.

2 Cryptographie, l'algorithme RSA en Java

2.1 Java

L'ensemble des fonctionnalités de base et évoluées de RSA sont disponibles en Java dans la librairie « java.security ». En outre, la classe java.math.BigInteger est une classe essentielle par le fait qu'elle

permet de manipuler de grands entiers et pré-implémente un grand nombre de fonctionnalités découlant de l'arithmétique modulaire. Nous nous intéressons ici à un programme Java pédagogique qui ignore l'ensemble des services disponibles.

2.2 Programme Java

Le programme travaille sur des long qui en Java occupent 64 bits. Le message M à chiffrer doit donc passer en format long. Par exemple, la transformation du message ('J','o','s','e','p','h') en format long s'opère comme suit :

0	1	0	0	1	0	1	0	0	1	1	0	1	1	1	1

'J' = 74 = $(01001010)_2$ est décalé à gauche, 'o' = 111 = $(01101111)_2$ rentre par la droite et ainsi de suite. La valeur exprimée ci-dessus est 19055 équivalent au message 'J','o'. Le programme ci-dessous ne traite que des messages de 8 caractères maxi. du fait que le format long en Java est de 64 bits.

```
  public Pedagogical_RSA_encryptor(char[] message) throws Exception {
         if(message.length > 8) throw new
Exception("Cryptography.Pedagogical_RSA_encryptor: Messages are limited to 64
bits"); // long uses a 64-bit format in Java

  // a bit slow:
  // long powersOfTwo = 0x0000000000000001L;
  // for(int i = message.length;i > 0;i--) {
  //     _message += message[i - 1] * powersOfTwo;
  //     powersOfTwo *= (2L * 2L * 2L * 2L * 2L * 2L * 2L * 2L);
  // }

  // faster:
         for(int i = 0;i < message.length;i++) {
             if(i > 0) _message <<= 8;
             _message += message[i];
         }
  }
```

2.3 Classe java.util.BitSet

L'exponentiation modulaire impose la transformation de e et de d sous forme binaire. On utilise la classe java.util.BitSet.

```
  public static java.util.BitSet toBitSet(long e) {
         java.util.BitSet bs = new java.util.BitSet();
         long powersOfTwo = 0x0000000000000001L;
         for(int i = 0;i < 64;i++) {
             if((e & powersOfTwo) == powersOfTwo) bs.set(i);
             powersOfTwo *= 0x0000000000000002L;
         }
         return bs;
  }
```

2.4 Synthèse

Deux classes servent à mettre en œuvre RSA :

```
  public class Euclid {
      private long _a,_b,_gcd,_x,_y;
      public Euclid(long a,long b) {…}
      private long extended_Euclid(long a,long b) {…}
      public long a() {…}
      public long b() {…}
      public long gcd() {…}
      public long x() {…}
      public long y() {…}
```

```
    public static long Basic_Euclid(long a,long b) {…}
    public long[] solver(long b) throws Exception {…}
    public static long Exponentiation(long message,long n,java.util.BitSet
 e) {…}
}

public class Pedagogical_RSA_encryptor {
    private long _message = 0L,_n = 0L,_d;
    public Pedagogical_RSA_encryptor(char[] message) throws Exception {…}
    public void encrypt(long n,Euclid euclid) throws Exception {…}
    public void decrypt() {}
}
```