Algorithmique 1
Algorithmes calculatoires

(un peu de cryptographie ...)

Hajer Akid & Sandrine Julia

Semestre 4

Algorithme RSA (en bref)

Préliminaires

- choisir 2 grands nombres premiers p et q de 100 digits
- multiplier p par q pour obtenir $N = p \cdot q$
- choisir un entier (relativement petit) e premier à $m = (p-1)(q-1)$
- trouver d, l'inverse multiplicatif de e modulo m

Chiffrement/déchiffrement

Soit un message M à chiffrer :

- la **clef publique** (N, e) permet de calculer le chiffré :
$$C = M^e \mod N$$
- la **clef privée** (N, d) permet de le déchiffrer :
$$M = C^d \mod N$$

Algorithme RSA (exemple)

Chiffrement/déchiffrement d'un message M

- la clef publique (N, e) permet de calculer le chiffré :
$$C = M^e \mod N$$
- la clef privée (N, d) permet de le déchiffrer :
$$M = C^d \mod N$$

Exemple

- avec $p = 11$ et $q = 13$, le module $N = p \cdot q = 143$ et $m = (p-1)(q-1) = 120$
- prenons $e = 7$, on en déduit que $d = 103$ car $e \cdot d \equiv 1 \mod 120$

Soit le message $M = 42$:

- chiffrement avec la clef publique : $C = 42^7 \mod 143 = 81$
- déchiffrement avec la clef privée : $M = 81^{103} \mod 143 = 42$

Les algorithmes en jeu

Algo 1 : génération d'un nombre premier aléatoire de n bits
Algo 2 : multiplication
Algo 3 : division entière et son reste
Algo 4 : exponentielle modulaire
Algo 5 : algorithme d'Euclide (pgcd)
Algo 6 : algorithme d'Euclide étendu
Algo 7 : inverse multiplicatif modulaire

Remarque

- toutes les opérations précédentes concernent des grands entiers
- on fixe à n leur taille en nombre de bits dans ce cours

Algo 1 : Générer un grand nombre premier

Entrée : un entier naturel n et k un nombre de tests à effectuer

Sortie : un grand nombre premier aléatoire N sur n bits

```
genPrime (entier n, entier k) {// algorithme probabiliste !
  tant que (Vrai) {
    // tirage aléatoire d'un candidat à tester
    N ← entier_aléa(n bits)
    // test de primalité
    ok ← Vrai ; i ← 1
    tant que (ok et i≤k){
      A ← entier_aléa(1,N)
      si ((A^(N-1) mod N) ≠ 1) {
        ok ← Faux  // N est composé (c'est sûr)
      }
      i ← i+1
    }
    si (ok) {
      retourner N  // N est probablement premier
} } }
```

Complexité test primalité seul : $O(k.n^3)$

Remarque

En effet, on montrera ultérieurement que la complexité de l'exponentielle modulaire est $O(n^3)$.

Algo 2 : Multiplication (rappel)

Entrée : deux grands entiers naturels x et y sur n bits

Sortie : leur produit $x \times y$

```
multiplication (grand entier x, grand entier y){
  // rappel cours 3
  res ← 0
  tant que (x > 0){
    si (ODD(x)){
      res ← res + y
    }
    y ← MULT2(y)
    x ← DIV2(x)
  }
  retourner res
}
```

Complexité : $O(n^2)$

Remarque

Avec l'algorithme de Karatsuba, on obtiendrait même $O(n^{1,59})$!

Algo 3 : Division entière

Entrée : deux grands entiers naturels x et y sur n bits avec y non nul

Sortie : le couple quotient et reste (q, r) de la division entière de x par y tel que : $x = q \cdot y + r$ avec $r < y$

```
divisionEntière (grand entier x, grand entier y){// y non nul
  si (x<y) {
    retourner (0,x)
  q,r ← divisionEntière(DIV2(x),y)
  q ← MULT2(q)
  r ← MULT2(r)
  si (ODD(x)){
      r ← r + 1
  }
  si (r ≥ y){
    r ← r − y
    q ← q + 1
  }
  retourner (q,r)
}
```

Complexité : (*cf.* TD 11)

Algo 4 : Exponentielle modulaire

Entrée :

- deux grands entiers naturels x et y sur n bits
- le module N, entier naturel sur n bits

Sortie :

- la valeur de : $x^y \mod N$

```
expoMod(grand entier x, grand entier y, grand entier N){

  //  voir TD 11

}
```

Complexité : $O(n^3)$

Algo 5 : Algorithme d'Euclide

Entrée : deux grands entiers naturels a et b sur n bits

Sortie : leur plus grand commun diviseur : pgcd(a, b)

```
Euclide(grand entier a, grand entier b){// rappel du TD 3
  si (b = 0){
    retourner a
  }
  retourner Euclide (b,a mod b)
}
```

Complexité : $O(n^3)$

Remarque

Il existe une version dichotomique plus efficace (*cf.* TD 11).

Algorithme d'Euclide étendu

Coefficients de Bézout

- il existe une infinité de couples d'entiers relatifs (x, y) tels que :

$$\text{pgcd}(a, b) = a.x + b.y$$

- x et y sont appelés les *coefficients de Bézout*
- l'*algorithme d'Euclide étendu* permet de trouver un tel couple (x, y)

Inverse modulaire

- si $pgcd(e, m) = 1$, l'inverse modulaire de e modulo m existe et c'est l'entier d tel que :

$$e.d \equiv 1 \mod m$$

- dans ce cas, comment calculer l'inverse de e modulo m ?

Application de l'algorithme d'Euclide étendu

Exemple

- on cherche l'inverse de $e = 7$ modulo $m = 120$
- on a bien pgcd(e, m) = pgcd$(7, 120) = 1$
- l'appel à Euclide_étendu(7,120) retourne (-17 , 1 ,1)
- on peut vérifier que :

$$\text{pgcd}(7, 120) = 1 = (\ -17\ ) * 7 + (\ 1\ ) * 120$$

- on déduit l'inverse modulaire $d = -17 \mod 120 = 103$
- on peut vérifier que :

$$e.d \equiv 1 \mod m$$

en effet :

- $103 \times 7 = 721$
- $721 \mod 120 = 1$

Algo 6 : Algorithme d'Euclide étendu

Entrée : deux grands entiers naturels a et b sur n bits

Sortie : le triplet (x, y, d) vérifiant :

$$\text{pgcd}(a, b) = d = a.x + b.y$$

```
entier Euclide_étendu (entier a, entier b) {
  si (b = 0) {
    retourner (1,0,a)
  }
  sinon {
    x,y,d ← Euclide_étendu(b,a mod b)
    retourner (y, x−(a div b)∗y, d)
  }
}
```

Complexité : $O(n^3)$

Algo 7 : Algorithme pour l'inverse modulaire

Entrée : deux grands entiers naturels a et m sur n bits

Sortie : l'inverse modulaire de a modulo m s'il existe

```
inverse_modulaire (grand entier a, grand entier m) {
  x,y,d ← Euclide_étendu(a,m)
  si (d ≠ 1) {
    retourner 'Pas d'inverse modulaire'
  }
  sinon {
    retourner x mod m
  }
}
```

Complexité : $O(n^3)$

Remarque

C'est une application directe de l'algorithme d'Euclide étendu.