Cryptologie : Cryptographie à clé publique

Matthieu Amiguet

2005 – 2006

haute école arc ingénierie
neuchâtel berne jura — saint-imier le locle porrentruy

Généralités 2

- Concept apparu en 1976 (Diffie, Hellman)
- Permet de résoudre le problème de la distribution des clés
- Repose généralement sur des résultats de
  - théorie des nombres
  - algorithmique
- RSA
  - Rivest, Shamir & Adelman, 1977
  - Devenu un standard *de facto*.

Modulos 3

- Vous connaissez l'*opérateur* de modulo en C, Java, ...

```
a = b % c
// a contient maintenant le reste
// de la division entière de b par c
```

- Dans ce chapitre, il sera plus utile de voir les modulos comme une *relation d'équivalence*
  - On peut le voir comme une sorte de surcharge de l'opérateur `==` pour les entiers...

```
if (a == b (mod c))
// Pseudocode !!
// Condition vraie après
// l'affectation ci-dessus !
```


Modulos – définition 4

$a == b \pmod n$

se lit	"a est égal à b modulo n"
en maths	$a - b = kn, \quad k \in \mathbb{Z}$
en français	a et b ont le même reste dans une division par n

Remarque Dans l'expression "a est égal à b modulo n", le "modulo n" porte sur toute l'égalité, pas sur le b !

Re-remarque La notation mathématique standard est $a \equiv b \pmod n$ est se lit "a est congru à b modulo n"

Modulos – exemples 5

- Le modulo n, c'est un peu comme si on prenait les entiers et qu'on les "enroulait" pour recoller 0, n, $2n$, ...
- Par exemple, chacune des égalités suivantes est correcte
  - $1 == 8 \,(mod\, 7)$
  - $8 == 1 \,(mod\, 7)$
  - $15 == 8 \,(mod\, 7)$
  - $1 == 15 \,(mod\, 7)$
  - $-6 == 1 \,(mod\, 7)$
  - ...

Arithmétique modulo 6

- Si l'on choisit un modulo, cela définit une nouvelle "structure arithmétique" sur les entiers
- Par exemple, modulo 7 :
  - $2 * 3 == -1$
  - $9 * 3 == -1$ (9 et 2 sont au fait *le même chiffre* modulo 7 !)
  - $2 * 4 == 1$
- Ce dernier exemple est particulièrement intéressant : dans le monde modulo 7, 4 joue le rôle de l'*inverse* de 2
  - En quelque sorte, $4 == \frac{1}{2}$...
  - On dit que 4 est l'*inverse modulaire* de 2 (modulo 7)

Le théorème chinois 7

Théorème chinois

Soient p, q avec $pgdc(p,q) = 1$. Alors

$$\begin{array}{rrcll} & x & == & y & (mod\, p) \\ & x & == & y & (mod\, q) \\ \Rightarrow & x & == & y & (mod\, pq) \end{array}$$

Le théorème chinois – exemples 8

- Exemple :

$$\begin{array}{rrcll} & 5820 & == & 15 & (mod\, 27) \\ & 5820 & == & 15 & (mod\, 43) \\ \Rightarrow & 5820 & == & 15 & (mod\, 1161) \end{array}$$

- Contre-exemple :

$$\begin{array}{rrcll} & 19 & == & 1 & (mod\, 9) \\ & 19 & == & 1 & (mod\, 6) \\ \text{mais} & 19 & \neq\neq & 1 & (mod\, 54) \end{array}$$

Le petit théorème de Fermat 9

Théorème de Fermat

Soient p premier et a avec $pgdc(a,p) = 1$. Alors

$$a^{p-1} \equiv 1 \pmod{p}$$

- On a donc aussi $a^p \equiv a \pmod{p}$
- Exemple : $5^{43} \equiv 5 \,(mod\, 43)$
- Contre-exemple : $5^{45} \equiv 35 \,(mod\, 45)$.

Complexité 10

- But : définir une mesure du “temps de résolution” d’un problème par un algorithme
  - indépendamment du matériel utilisé
  - en fonction de la “taille” des données à traiter (mesurée en bits, nombre éléments, . . .)
  - mesuré en “nombre d’opérations” (cycle d’horloge, instruction, opération arithmétique, . . .)
- On peut faire une mesure
  - au pire des cas
  - en moyenne.

Temps polynomial 11

Notation asymptotique

- f est de complexité $O(g(n))$ ssi il existe des constantes $c > 0$, $n_0 > 0$ avec $0 \leq f(n) \leq cg(n)$ pour tout $n \geq n_0$

- Un algorithme *en temps polynomial* est un algorithme dont le temps d’exécution au pire est $O(n^k)$
- Dans le cas contraire, on parle d’algorithme *en temps exponentiel*.

Classes de complexité 12

- La classe de complexité **P** est l’ensemble de tous les problèmes solubles en temps polynomial
- La classe **NP** est l’ensemble des problèmes pour lesquels la vérification d’une solution donnée peut se faire en temps polynomial
- Un problème est dit **NP**-complet si sa résolution en temps polynomial entraîne la résolution en temps polynomial de tous les problèmes de **NP**
- On suppose que **P** $\neq$ **NP**, mais ça reste une supposition ! ! !

Factorisation 13

Problème de la factorisation

- Soit n un entier
- Trouver sa décomposition en facteurs premiers
- Problème **NP** ! ! !

Génération de clés RSA

- Générer deux nombres premiers p et q
  - distincts
  - grands (p.ex. 512 bits)
  - de même ordre de grandeur
- Calculer $n = pq$ et $\Phi = (p-1)(q-1)$
- Choisir un entier aléatoire e tel que $pgcd(e, \Phi) = 1$
- Calculer l'unique entier d tel que
  - $ed == 1 \pmod{\Phi}$
  - $1 < d < \Phi$
- Clé publique : (n, e)
- Clé privée : d.

Bob veut envoyer un message à Alice crypté par RSA 15

RSA – cryptage

- Bob se procure la clé publique (n, e) d'Alice
- Il représente le message comme un entier $0 \leq m \leq n-1$
- Il calcule $c = m^e \pmod{n}$
- Il envoie c à Alice

RSA – décryptage

- Alice calcule $m = c^d \pmod{n}$.

RSA – pourquoi ça marche 16

- $ed == 1 \pmod{\Phi} \Rightarrow$
  $ed = 1 + k\Phi = 1 + k(p-1)(q-1), \quad k \in \mathbb{Z}$
- Si $pgcd(m, p) = p$, $m^{ed} == m \pmod{p}$
  - car p divise m, donc l'égalité revient à $0 == 0 \ldots$
- Si $pgcd(m, p) = 1$

$$\begin{array}{rcl|l} m^{p-1} & == & 1 \pmod{p} & \hat{}k(q-1) \\ m^{k(p-1)(q-1)} & == & 1 \pmod{p} & *m \\ m^{1+k(p-1)(q-1)} & == & m \pmod{p} & \\ m^{ed} & == & m \pmod{p} & \end{array}$$

- Par un raisonnement similaire, $m^{ed} == m \pmod{q}$
- Théorème chinois : $m^{ed} == (m^e)^d == c^d == m \pmod{pq}$.

RSA – propriétés 17

- On *suppose* que trouver d en fonction de (n,e) est aussi difficile que de factoriser n
- Pour augmenter l'efficacité du cryptage on préfère généralement une petite valeur de e (p.ex. 3)
  - Attention ! pas sûr si on envoie le même message à différentes personnes avec différents n
- Si l'espace des messages est petit, possibilité de recherche exhaustive (on peut "saler" le message pour éviter cela, ie ajouter une chaîne de bits aléatoires)

RSA – propriétés (suite) 18

- Il ne faut pas que plusieurs personnes partagent le même n avec des (e,d) distincts, car la connaissance de (e,d) permet de factoriser n
- Nombre de points fixes de RSA :
  - $[1+pgdc(e-1,p-1)]\cdot[1+pgcd(e-1,q-1)]$
  - Toujours au moins 9
  - Proportion généralement négligeable.

L'élévation à la puissance et les modulos 19

- Le coeur du cryptage RSA consiste en une élévation à la puissance *modulo n*.
- Comment calculer efficacement $b^e \pmod m$?
- La première idée est de calculer $(b*b*\ldots*b)\%m$
  - Pour e grand, ceci posera des problèmes de capacité...
- On peut alors essayer $(\ldots((b*b)\%m)*b)\%m)\ldots)*b)\%m$
  - Les nombres restent "petits"
  - Il y a tout de même e multiplications et e modulos à calculer...

Vers une solution... 20

- Pour trouver un meilleur algorithme, représentons e en binaire : $e=\sum e_i 2^i$
  - donc les e_i sont les bits de la représentation binaire de e
- Alors $b^e = b^{\sum e_i 2^i}$
- Comme $x^{y+z} = x^y x^z$, on a $b^e = \prod b^{e_i 2^i} = \prod (b^{2^i})^{e_i}$
- NB : les e^i sont soit 0, soit 1... donc l'élévation à la puissance e^i est simple à calculer !
- On a donc passé d'une complexité en e à une complexité en $\log_2 e$.

Exponentiation modulaire rapide, l'algorithme 21

```
def modpow(base, exp, m):
    result = 1
    while (exp > 0) :
        if (exp & 1 > 0):
            result = (result * base) % m
        exp = exp >> 1
        base = (base * base) % m
        print base, result
    return result
```


Le calcul du pgdc 22

- Algorithme d'Euclide

```
def pgdc (a ,b) :
    while b != 0 :
        a, b = b, a % b
    return a
```

- Une extension de cet algorithme permet de trouver l'inverse modulaire.

Le calcul de l'inverse modulaire 23

```
def modinv(n,m):
  " Retourne l'inverse modulaire de n modulo m "
  i2, i1 = 0,1
  while True:
    q, r = m/n, m % n
    if r == 0: break
    i = i2 - q*i1
    m,n = n, r
    i2, i1 = i1, i
  if n != 1:
    raise Exception, "n et m non premiers entre eux"
  return i1
```


Comment trouver p et q ? 24

- La génération directe est impraticable
- Les tests de primalité ne sont pas assez efficaces
- Solution :
  - Tests probabilistes de primalité

⇒On génère des entiers qui ont une *très forte probabilité* d'être premiers.

Comparaison 25

	Clé privée	Clé publique
Débit	Haut	Bas
Clé	“Courte” (~100 bits)	“Longue” (~1000 bits)
“Talon d’Achille”	Distribution des clés	Authentification
Changement de clé	Souvent (Chaque session)	Rarement (→ plusieurs années)
Histoire	Longue (toujours/~1970)	Courte (~1700/~1980)
Sécurité prouvée	Peu praticable (flux) à inexistante (blocs)	Inexistante (Repose sur des conjectures)

Alors... que faire ? 26

- Une solution couramment utilisée est la suivante :
  - Utiliser un algorithme à clé publique pour échanger une clé privée (“clé de session”)
  - Utiliser cette clé privée pour crypter le contenu
- Cette solution permet de
  - résoudre le problème de la distribution de la clé privée
  - résoudre le problème de la durée de vie de la clé privée
  - bénéficier du haut débit des algorithme de clé privée
- Par contre,
  - le problème de l’authentification n’est pas résolu
  - il y a deux fois plus de vulnérabilités potentielles !