

## ch4 : L'algorithme RSA

### I] Contexte asymétrique

### II] Description de l'algorithme RSA

① Clés

② Fonction RSA :

$$f: \begin{cases} \mathbb{Z}/n\mathbb{Z} \rightarrow \mathbb{Z}/n\mathbb{Z} \\ x \mapsto y = x^e \text{ mod } n \end{cases}$$

Th: Si  $n=pq$ , et si  $\text{pgcd}(e, \varphi(n))=1$

alors  $f$  est bijective

$$\text{et } f^{-1}: \begin{cases} \mathbb{Z}/n\mathbb{Z} \rightarrow \mathbb{Z}/n\mathbb{Z} \\ y \mapsto x = y^d \text{ mod } n \end{cases}$$

$$\text{où } d = e^{-1} \text{ mod } \varphi(n)$$

③ Sécurité de RSA

- trouver la clé secrète  $\rightarrow$  [pb de la factorisation]

- trouver  $f^{-1}(y)$  pour certains  $y$   $\rightarrow$  [pb de calcul  
de racines  
e-ièmes  
modulo  $n$ ]

### III] Implementation

① Exponentiation modulaire

$$y = \boxed{x^e \text{ mod } n}$$

$$x = \boxed{y^d \text{ mod } n}$$

$$d = e^{-1} \text{ mod } \overline{(p-1)(q-1)}$$

(algorithme d'Euclide étendu)

Implémentation naïve :  $\begin{cases} n \leftarrow 1024 \text{ bits} \\ y \leftarrow 1024 \text{ bits} \\ d \leftarrow 1024 \text{ bits} \end{cases}$

ex:  $d \approx 2^{1024}$

$$y^d \approx y^{2^{1024}} = \left( (y^2)^2 \dots \right)^2$$

$$|y| = 1024 \text{ bits}$$

$$|y^2| = 2048 \text{ bits}$$

$$|y^4| = 4096 \text{ bits}$$

$$|y^d| \approx \underbrace{(1024 \times 2^{1024})}_{\text{taille de } y^d} \text{ bits}$$

### Méthode "square and multiply"

. On écrit la décomposition de  $d$  en base 2

$$d = \underbrace{d_{k-1}}_{(k \text{ bits})} 2^{k-1} + \underbrace{d_{k-2}}_{\dots} 2^{k-2} + \dots + \underbrace{d_2}_{\dots} 2^2 + \underbrace{d_1}_{\dots} 2^1 + \underbrace{d_0}_{\dots}$$

- On applique l'algorithme "square and multiply" pour calculer  $x = y^d \bmod n$

$x := 1$   
 Pour  $i$  de  $(k-1)$  à  $\emptyset$  :  
 $| x := \boxed{x^2 \bmod n} \rightarrow \text{square}$   
 Si ( $d_i = 1$ ) alors  $x := \boxed{x \times y \bmod n} \downarrow \text{multiply}$

Exemple :  $d = 13 = \overline{1101}$

$$d = 8 + 4 + 1 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0$$

$$(k=4) \quad d_3 = 1 ; d_2 = 1 ; d_1 = 0 ; d_0 = 1$$

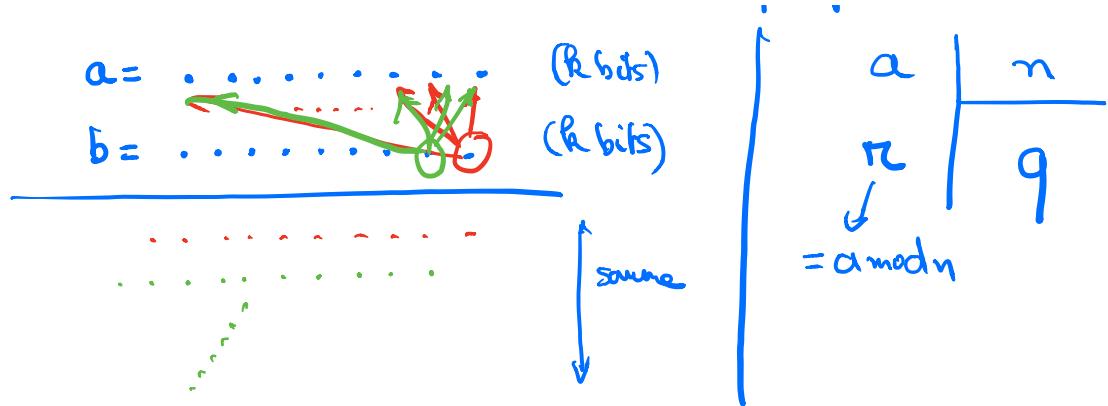
$i$	$x$
	1
3	1
	$1 \times y = y \bmod n$
2	$y^2 \bmod n$
	$y^2 \times y = y^3 \bmod n$
1	$y^3 \bmod n$
	$(y^3)^2 = y^{12} \bmod n$
0	$y^{12} \times y = \boxed{y^{13} \bmod n}$

les variables utilisées ne contiennent jamais de valeurs de plus de 2048 bits

Complexité de l'algorithme square and multiply

$k \times k^2$  → nb d'opérations élémentaires  
 ↑  
 nb de termes dans la boucle

| nb d'opérations élémentaires pour multiplier 2 entiers de  $k$  bits ou pour faire la réduction mod  $n$



$\Rightarrow$  la complexité de square and multiply est

$$\boxed{\mathcal{O}(k^3)}$$

## ② Génération des clés

Objectif: générer deux nombres premiers  $p$  et  $q$   
 (typiquement de 512 bits chacun)  
 $(\Rightarrow n = p \times q$  fera 1024 bits)

Méthode usuelle : 1) générer un nombre entier  $p$  de 512 bits  
 2) utiliser un test de primalité sur  $p$   
 (ex: test de Miller-Rabin)  
 { - si le test répond "pas premier"  
 on revient à l'étape 1  
 - sinon, on garde ce nombre  $p$

Remarque : Le théorème des nombres premiers garantit que la densité des nombres premiers n'est pas trop petite

Th (Hadamard, de la Vallée-Poussin, 1896)

$$\text{Soit } \pi(x) = \text{Card} \{ p \leq x ; p \text{ premier} \}$$

$$\text{Atlas } \pi(x) \sim \frac{x}{\ln x} \quad (x \rightarrow +\infty)$$

Corollaire: Pour  $x \approx 512$  bits, la densité des nombres premiers au voisinage de  $x$  est  $\approx \frac{1}{\ln 2^{512}} \approx \frac{1}{300}$

## IV] Utilisation de RSA en chiffrement

### ① Méthode "naïve"

$$y = x^e \pmod{n}$$

message chiffré      message clair

$\Rightarrow x = y^d \pmod{n}$

déschiffrement

chiffrement

pb 1:  $x$  est un entier tel que  $0 \leq x < n$

$$x \in [0, n[ \iff y \in [0, n[$$

$\Rightarrow$  solution: utiliser un mode d'opération

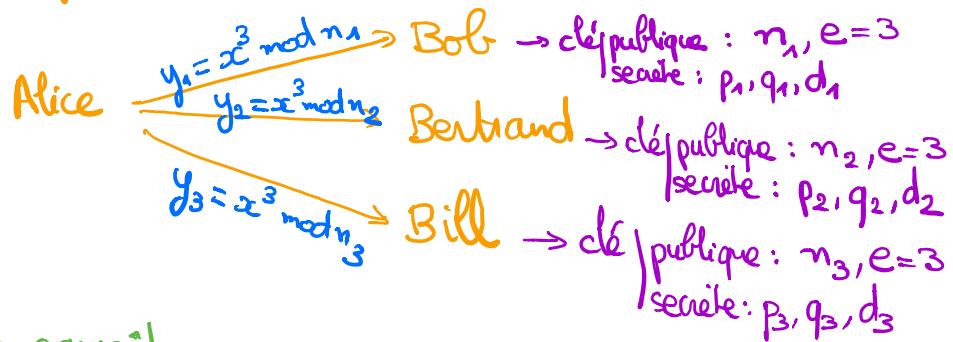
pb 2: si  $x < \sqrt[e]{n}$  (ex:  $e=3$ ,  $x < \sqrt[3]{n}$ )

$$\Rightarrow y = \underbrace{x^e}_{<n} \pmod{n}$$

$$\Rightarrow \boxed{x = \sqrt[e]{y}}$$

$$(\text{ex: } e=3, x = \sqrt[3]{y})$$

### pb 3 : Attaque de Hastad



L'attaquant connaît

$$\begin{aligned} y_1 &= x^3 \pmod{n_1} \\ y_2 &= x^3 \pmod{n_2} \end{aligned}$$

Th des  
restes  
chinois

$$x^3 \pmod{(n_1 \times n_2)}$$

[à condition que  
 $\text{pgcd}(n_1, n_2) = 1$ ]

$$\begin{aligned} x^3 \pmod{(n_1 \times n_2)} \\ y_3 = x^3 \pmod{n_3} \end{aligned}$$

Th des  
restes  
chinois

$$x^3 \pmod{(n_1 \times n_2 \times n_3)}$$

[à condition que  
 $\text{pgcd}(n_1 \times n_2, n_3) = 1$ ]

avec  $0 \leq x < n_1$   
 $0 \leq x < n_2$   
 $0 \leq x < n_3$

⇒

$$0 \leq x^3 < n_1 n_2 n_3$$

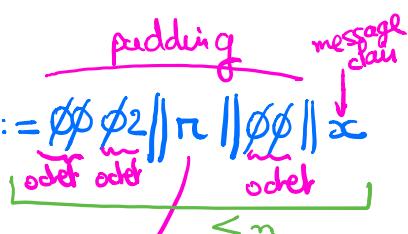
Donc  $x^3 \pmod{x_1 n_2 n_3} = x^3$

L'attaquant connaît  $x^3$  et donc retrouve  $x$

### ② Exemple de "bonne" méthode

Standard PKCS #1 v 1.5

Différence:  $y = (\mu(x))^e \pmod{n}$  avec  $\mu(x) = \frac{\phi_1 \phi_2}{n} \parallel \tau \parallel \frac{\phi_1 \phi_2}{n} \parallel x$



valeur aléatoire constituée  
d'au moins 8 octets  
qui sont tous non nuls

Réponse: autre avantage: si on chiffre 2 fois le même message, on obtient 2 chiffres complètement différents.

## IV] RSA en signature

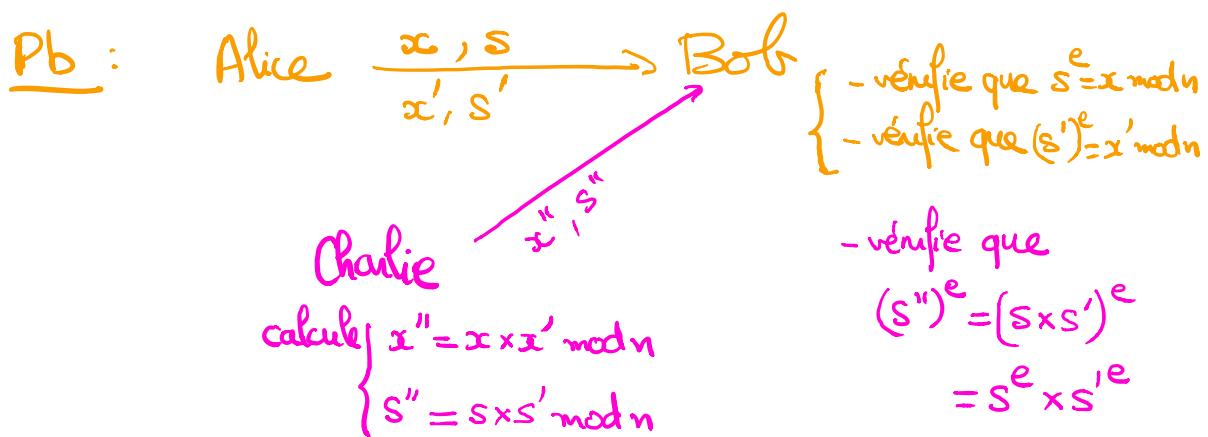
### ① Méthode "naïve"

$$\text{Signature: } s = x^d \pmod{n}$$

Signature      message

$\xrightarrow{d \text{ sécrète}}$

Vérification: on vérifie que  $s^e \stackrel{?}{=} x \pmod{n}$



Réponse: L'attaque fonctionne en utilisant la multiplicativité de la fonction RSA

$$\text{RSA : } f(s \times s') = f(s) \times f(s')$$

## ② Une "bonne" méthode de signature

Standard PKCS #1 v1.5

Public key  
cryptographic Standard

Signature:  $s = (\mu(x))^d \text{ mod } n$

avec  $\mu(x) = \underbrace{\phi\phi}_{\text{odd}} \underbrace{\phi 1}_{\text{odd}} \underbrace{\text{FF}}_{\text{odd}} \dots \underbrace{\text{FF}}_{\text{odd}} \underbrace{\phi\phi}_{\text{odd}} \parallel c_n \parallel h(x)$

↓  
no de  
la fonction  
de hachage

Vérification:  $s^e \text{ mod } n = \overbrace{\phi\phi \phi 1 \text{ FF} \dots \text{ FF} \phi\phi c_n}^{\text{h}(x)}$