Algorithmique
Cours 5 : Cryptographie et cryptosystème RSA

ROB3 – année 2014-2015

Cryptographie et web

- Avant l'apparition du web, la cryptographie servait essentiellement à assurer la confidentialité des échanges d'informations entre un petit nombre d'acteurs s'étant mis d'accord sur des conventions secrètes.

- Avec **Internet** apparaît le besoin de communications entre un grand nombre d'intervenants qui ne se verront jamais, ou qui n'ont aucun moyen de mettre en place ces conventions secrètes, comme par exemple les vendeurs et les acheteurs sur le Web. Mettre en place ces conventions malgré tout serait problématique, car le nombre de clefs à gérer deviendrait rapidement astronomique il croît avec le carré du nombre d'utilisateurs.

- Il faut donc trouver des moyens de communiquer sans échange préalable d'informations sensibles, puisque cela doit se faire dans un environnement ouvert, susceptible d'espionnage de toute nature, interne ou externe.

- Les solutions à ces problèmes passent par l'introduction d'un nouveau paradigme, celui de la cryptographie à clefs publiques, concept inventé par W. Diffie et M. Hellman en 1976.

Cryptographie : cadre général

- **Alice** et **Bob** souhaitent communiquer en privé
- **Eve**, une oreille indiscrète, souhaite savoir ce qu'Alice et Bob disent

  Alice envoie un message spécifique x, écrit en binaire, à son ami Bob :

  1. Alice encode le message en e(x) et l'envoie
  2. Bob applique une fonction de décryptage d(.) pour le décoder : d(e(x)) = x
  3. Eve va intercepter e(x) : par exemple, elle peut être un « sniffer » sur le réseau

- Dans l'idéal, la fonction d'encryptage e est choisie de façon à ce que sans connaître d(.), Eve ne puisse rien tirer de l'information qu'elle a interceptée

  Autrement dit, la connaissance de e(x) ne dit rien sur ce que x pourrait être

- Une fontion d'encryptage e : <messages> → <messages encryptés> doit être inversible -afin de rendre le décryptage possible- et est donc une bijection. Son inverse est la fonction de décryptage d(.).

Algorithmes de chiffrement

Deux classes d'algorithmes de chiffrement :

- Cryptographie symétrique (ou à **clé privé**)

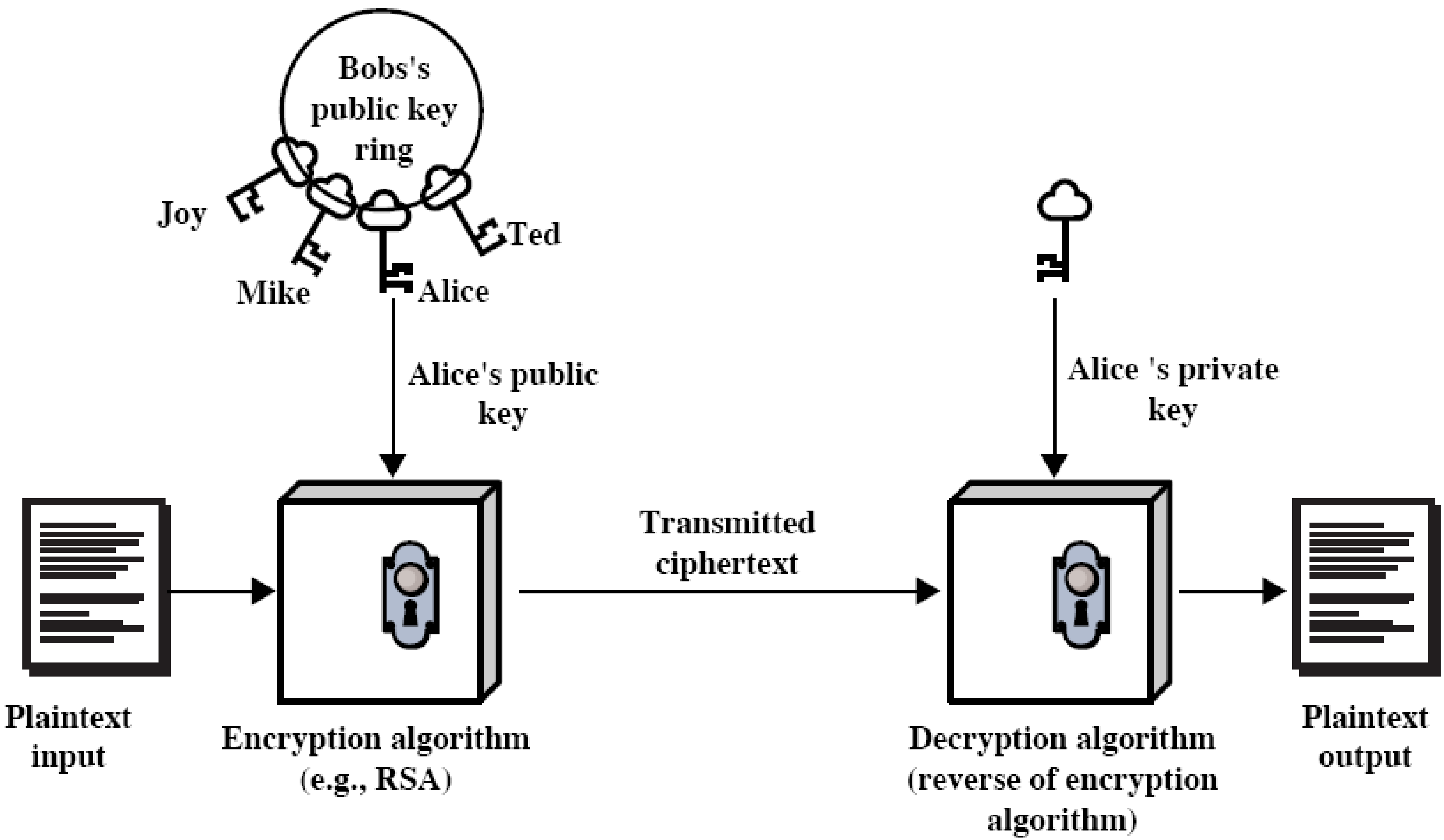
  Les clés de cryptage et de décryptage sont identiques : expéditeur et destinataire possèdent tous deux la clef du coffre qui contient le message (c'est une image !) et qui fait la navette entre les deux.

- Cryptographie asymétrique (ou à **clé publique**)
  Les clés de cryptage et de décryptage sont distinctes : l'expéditeur a une clef pour fermer le coffre, et le destinataire une clef distincte, qu'il est le seul à posséder et qui permet d'ouvrir ce coffre. La clef de fermeture ne permet d'ouvrir le coffre. Elle est publique.

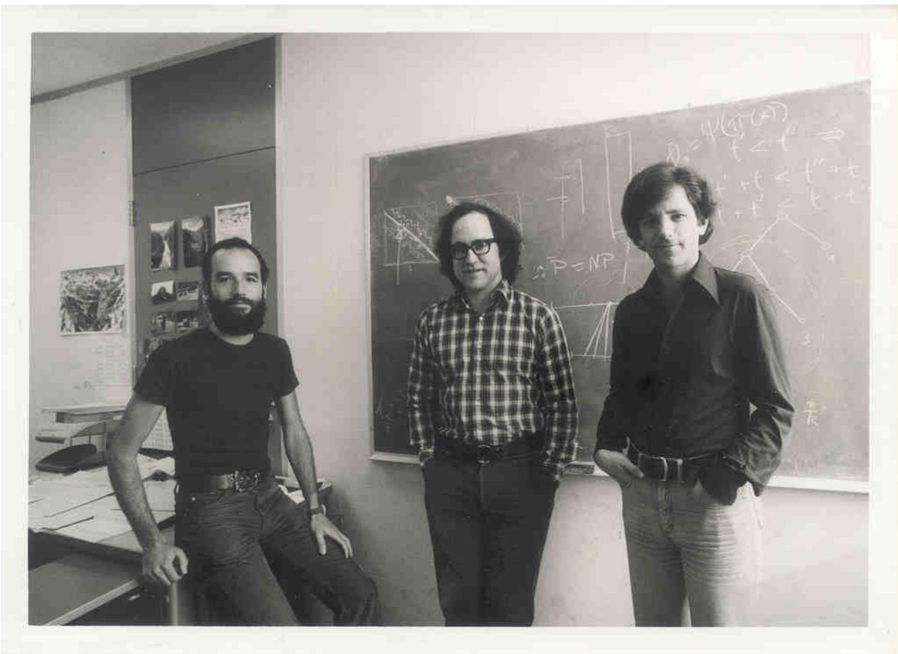
Cryptographie à clé publique

- Aussi connue sous le nom de cryptographie asymétrique.
- Chaque utilisateur dispose d'une paire de clé : une clé publique et une clé privée.
- La clé publique est utilisée pour le cryptage.
  - La clé est connue du public.
- La clé privée est utilisée pour le décryptage.
  - La clé est connue du seul propriétaire.

Cryptographie à clé publique



Le cryptosystème RSA



- 1977 : Rivest, Shamir et Adleman (RSA...) cherchent à établir que tout système à clé publique présente des failles.
- Ils découvrent au contraire un nouveau système à clé publique qui marche très bien et supplante les autres.

Sécurité du système : il est beaucoup plus facile de faire le produit de deux nombres premiers que de factoriser un nombre en le produit de deux nombres premiers.

Rappels d'arithmétique modulaire

- x mod n = y $\leftrightarrow$ x = y (mod n) $\leftrightarrow$ n divise (x-y)
- **Règle de substitution**
  Si x = x' (mod n) et y = y' (mod n) alors :
  x + y = x' + y' (mod n) et xy = x'y' (mod n)
- Associativité : x + (y + z) = (x + y) + z (mod n)
- Commutativité : xy = yx (mod n)
- Distributivité : x * (y + z) = xy + yz (mod n)
- Cela implique qu'il est légal de réduire les résultats intermédiaires par mod n lors d'une séquence d'opérations arithmétique
  Exemple : $2^{345} = (2^5)^{69} = 32^{69} = 1^{69} = 1 \pmod{31}$

Le protocole RSA

- Génération des clés
  - Générer deux grands nombres premiers p et q
  - Soit $n = pq$
  - Soit $m = (p-1)(q-1)$
  - Choisir un nombre e premier avec m (choix fréquent : $e = 3$)
  - Trouver d tel que $de \bmod m = 1$
- Clés obtenues
  - **Clé publique :** (e,n)
  - **Clé privée :** (d,n)
- Cryptage et décryptage
  - **Cryptage :** $y = x^e \bmod n$
  - **Décryptage :** $x = y^d \bmod n$

Exemple « à la main »

RSA

- n = pq
- m = (p-1)(q-1)
- e et m premiers entre eux
- de mod m = 1
- **Clé publique :** (e,n)
- **Clé privée :** (d,n)
- **Cryptage :** $y = x^e \bmod n$
- **Décryptage :** $x = y^d \bmod n$

- p = 7 et q = 19
- n = 7 * 19 = 133
- m = (p-1) * (q-1) = 6 * 18 = 108
- Choix de e premier avec m
  PGCD(2,108) = 2 ; PGCD(3,108) = 3 ;
  PGCD(4,108) = 4 ; PGCD(5,108) = 1 → e = 5
- Détermination de d tel que de mod m = 1
  Autrement dit, il existe k tel que d = (1+km) / e
  k = 0 → d = 1/5 ; k = 1 → d = 109/5 ; k = 2 → d = 217/5 ;
  k = 3 → d = 325/5 = 65
- **Clé publique :** (n = 133 ; e = 5)
- **Clé privée :** (n = 133 ; d = 65)

Exemple « à la main »

- $n = 133$; $e = 5$; $d = 65$
- Supposons qu'on cherche à transmettre $x = 6$
- Cryptage
  $y = x^e \bmod n = 6^5 \bmod 133 = 7776 \bmod 133 = 62$
- Décryptage
  $x = y^d \bmod n$
  $= 62^{65} \bmod 133$
  $= 62 * 62^{64} \bmod 133$
  $= 62 * (62^2)^{32} \bmod 133$
  $= 62 * (3884)^{32} \bmod 133$
  $= 62 * (3884 \bmod 133)^{32} \bmod 133$
  $= 62 * 120^{32} \bmod 133$
  $= 62 * 36^{16} \bmod 133 = 62 * 99^8 \bmod 133 = 62 * 92^4 \bmod 133$
  $= 62 * 85^2 \bmod 133 = 62 * 43 \bmod 133 = 2666 \bmod 133 = 6$

RSA

- $n = pq$
- $m = (p-1)(q-1)$
- e et m premiers entre eux
- $de \bmod m = 1$
- **Clé publique :** (e,n)
- **Clé privée :** (d,n)
- **Cryptage :** $y = x^e \bmod n$
- **Décryptage :** $x = y^d \bmod n$

Sécurité du système

- Pour « casser » le code, il faut trouver d (exposant privé) à partir de n et e (clé publique)
- Eve sait que $de \bmod m = 1$
- Pour résoudre cette équation, il faut connaître m...
- ...Autrement dit, déterminer les nombres premiers p et q tels que $pq = n$ (puisque $m = (p-1)(q-1)$)
- Donc trouver la factorisation de n en deux nombres premiers p et q
- La factorisation d'un entier (de très grande taille) en facteurs premiers est extrêmement difficile, cette opération nécessitant une capacité de calcul très importante.
- Pour exemple : en 2010, l'INRIA et ses partenaires ont réussi à factoriser un entier de 768 bits. Il leur a fallu deux ans et demi de recherche, et plus de 10^{20} calculs. C'est à ce jour le meilleur résultat connu de factorisation.
- Il est régulièrement recommandé d'utiliser des tailles de clés de plus en plus grandes (actuellement de 2048 bits).

Un exemple (un peu) plus réaliste

- Supposons que les facteurs premiers p et q sont (en base 16) :

  p = 33 d4 84 45 c8 59 e5 23 40 de 70 4b cd da 06 5f bb 40 58 d7 40 bd 1d 67 d2 9e 9c 14 6c 11 cf 61

  q = 33 5e 84 08 86 6b 0f d3 8d c7 00 2d 3f 97 2c 67 38 9a 65 d5 d8 30 65 66 d5 c4 f2 a5 aa 52 62 8b

- n = pq est alors l'entier suivant à 508 bits (écrit ici en base 16) :

  n = 0a 66 79 1d c6 98 81 68 de 7a b7 74 19 bb 7f b0 c0 01 c6 27 10 27 00 75 14 29 42 e1 9a 8d 8c 51 d0 53 b3 e3 78 2a 1d e5 dc 5a f4 eb e9 94 68 17 01 14 a1 df e6 7c dc 9a 9a f5 5d 65 56 20 bb ab

- L'exposant public e est 65537 en base 10, soit en base 16 :

  e = 01 00 01

- L'exposant privé d s'écrit comme suit :

  d = 01 23 c5 b6 1b a3 6e db 1d 36 79 90 41 99 a8 9e a8 0c 09 b9 12 2e 14 00 c0 9a dc f7 78 46 76 d0 1d 23 35 6a 7d 44 d6 bd 8b d5 0e 94 bf c7 23 fa 87 d8 86 2b 75 17 76 91 c1 1d 75 76 92 df 88 81

Où sont les algorithmes ?

Des algorithmes dédiés sont requis pour :

- Déterminer deux nombres premiers p et q de grande taille
- Résoudre l'équation $de \bmod m = 1$
- Faire une exponentiation rapide lors du :
  - cryptage $y = x^e \bmod n$
  - décryptage $x = y^d \bmod n$

RSA

- $n = pq$
- $m = (p-1)(q-1)$
- e et m premiers entre eux
- $de \bmod m = 1$
- **Clé publique :** (e,n)
- **Clé privée :** (d,n)
- **Cryptage :** $y = x^e \bmod n$
- **Décryptage :** $x = y^d \bmod n$

Résoudre l'équation $de \bmod m = 1$

- Remarque : pour e et m fixés, cette équation n'admet pas toujours de solution !
  Exemple : $2d \bmod 6 \neq 1$ pour tout d
  Plus généralement, dès lors que $\mathrm{PGCD}(e,m) \neq 1$, l'équation n'admet pas de solution.
- L'approche brutale de résolution que nous avons employée pour l'exemple n'est bien sûr pas adaptée dès lors que e et m sont des grands nombres
- Pour e et m fixés, résoudre l'équation $de \bmod m = 1$ revient à trouver des entiers d et y tels que :

$$de + my = 1$$

- Pour ce faire, on fait appel à l'algorithme d'Euclide étendu

Algorithme d'Euclide

- L'algorithme d'Euclide permet de calculer le Plus Grand Commun Diviseur (PGCD) de deux entiers a et b.
- Il est fondé sur la propriété suivante :

Propriété. Si a = bq + r alors PGCD(a,b) = PGCD(r,b).

Preuve. Il suffit de montrer que PGCD(a,b) = PGCD(a-b,b), dont on déduit le résultat en soustrayant q fois b à a.

Tout entier qui divise à la fois a et b doit aussi diviser a-b, et donc PGCD(a,b) ≤ PGCD(a-b,b). De même, tout entier qui divise à la fois a-b et b doit aussi diviser a et b, et donc PGCD(a,b) ≥ PGCD(a-b,b).

```
Fonction PGCD(a,b)
Entrée : deux entiers a,b avec a ≥ b ≥ 0
Sortie : plus grand commun diviseur de a et b
Si b = 0 alors retourner a
         sinon retourner PGCD(b,a mod b)
```

Complexité de l'algorithme d'Euclide

```
Fonction PGCD(a,b)
Entrée : deux entiers a,b avec a ≥ b ≥ 0
Sortie : plus grand commun diviseur de a et b
Si b = 0 alors retourner a
         sinon retourner PGCD(b,a mod b)
```

L'analyse de complexité fait appel au résultat suivant :
Si $a \geq b$ alors $a \bmod b < a/2$.

En effet, on a soit $b \leq a/2$, soit $b > a/2$:
- si $b \leq a/2$, alors $a \bmod b < b \leq a/2$;
- si $b > a/2$, alors $a \bmod b = a-b < a/2$.

Par conséquent, à chaque appel récursif, l'un des deux arguments (a ou b) est divisé par au moins 2. Si on avait deux nombres de n bits au départ, on arrive donc au cas de base en au plus 2n itérations.
A chaque itération, l'opération de modulo est en $O(n)$, et par conséquent la complexité totale est $O(n^2)$.

Algorithme d'Euclide étendu

- Supposons qu'on veuille résoudre l'équation 15d mod 26 = 1, autrement dit trouver d et y tels que : $\boxed{15d + 26y = 1}$
- Un légère modification de l'algorithme d'Euclide le permet !
- Observons le déroulement du calcul de PGCD(15,26)...

PGCD(15,26) = ?

26 = 15 + 11

15 = 11 + 4

11 = 2 * 4 + 3

4 = 1 * 3 + 1

3 = 3 * 1 + 0

Remarque : 15 et 26 dont premiers entre eux, d'où l'existence de d et PGCD(15,26) = 1.

$$\begin{aligned} \text{PGCD}(15,26) = 1 &= 4 - 3 = 4 - (11 - 2 * 4) = -11 - 3 * 4 \\ &= -11 + 3 * (15 - 11) = 3 * 15 - 4 * 11 \\ &= 3 * 15 - 4 * (26 - 15) = \boxed{15 * 7 + 26 * -4 = 1} \end{aligned}$$

$$\boxed{d = 7}$$

Algorithme d'Euclide étendu

Ce principe par « substitution arrière » peut se formaliser par un algorithme récursif très proche de la fonction PGCD :

```
Fonction PGCDmodif(a,b)
Entrée : deux entiers a,b avec a ≥ b ≥ 0
Sortie : x,y,d tels que d=PGCD(a,b) et ax + by = d
Si b = 0 alors retourner (1,0,a)
(x',y',d) = PGCDmodif(b,a mod b)
retourner (y',x'-(a/b)y',d)
```

La complexité est bien évidemment la même que celle de l'algorithme d'Euclide (la version étendue de l'algorithme d'Euclide se contente de faire remonter quelques information supplémentaires à chaque itération), soit $O(n^2)$.

Exponentiation modulaire

```
Fonction Modexp(x,e,n)
Entrée : entiers x,e,n
Sortie : xᵉ mod n
Si e = 0 alors retourner 1
f = Modexp(x,e/2,n)
Si e est pair alors
   retourner f² mod n
       sinon
       retourner x * f² mod n
```

RSA

- n = pq
- m = (p-1)(q-1)
- e et m premiers entre eux
- de mod m = 1
- **Clé publique :** (e,n)
- **Clé privée :** (d,n)
- **Cryptage :** $y = x^e \bmod n$
- **Décryptage :** $x = y^d \bmod n$

Autrement dit :

$$x^e = \begin{cases} \left(x^{\lfloor e/2 \rfloor}\right)^2 & \text{si } e \text{ est pair} \\ x \cdot \left(x^{\lfloor e/2 \rfloor}\right)^2 & \text{si } e \text{ est impair} \end{cases}$$

Soit N le nombre de bits du plus grand entier parmi x,e,n. L'algorithme s'arrête après au plus N appels récursifs, et lors de chaque appel il multiplie des entiers d'au plus N bits (faire le calcul modulo n nous sauve ici), pour une complexité globale de `Modexp` en $O(N^3)$.