Cryptographie à clé publique
Cours 2

Julien Lavauzelle

Université Paris 8

Master 1 mathématiques et applications – parcours ACC

03/02/2023

La semaine dernière

Questions ?

Rappel : évaluation

Modalités d'évaluation de l'UE :

1. **Première note : contrôle continu**
   - 4 ou 5 **devoirs à la maison**, à rendre dans les deux semaines.
   - Principalement des exercices de programmation.

2. **Seconde note : partiel sur table** en fin d'année

Note finale :	50 % note 1 + 50 % note 2

Exponentiation rapide en 1 slide

Supposons que dans un groupe $\mathbb{G}$, on sait évaluer rapidement :

- la multiplication de deux éléments $(g, h) \mapsto gh$
- le carré d'un élément $g \mapsto g^2$

Alors, pour évaluer l'opération $g \mapsto g^m$, il suffit de $\lceil \log(m) \rceil$ multiplications et $\lceil \log(m) \rceil$ élévations au carré.

Preuve : pour cela, on décompose m en binaire : $m = \sum_{i=0}^{d} m_i 2^i$, et on va calculer de manière itérative :

$$g^m = g^{m_0} \cdot (g^2)^{m_1} \cdot (g^{2^2})^{m_2} \cdots (g^{2^d})^{m_d}$$

Autrement dit, on exécute l'algorithme :

- $p \leftarrow g$
- $r \leftarrow 1_{\mathbb{G}}$
- **Pour** $i = 0, \ldots, d$, **faire :**
  - **Si** $m_i = 1$, **alors** $r \leftarrow p \times r$
  - $p \leftarrow p^2$
- **Retourner** r.

Remarque : le calcul des m_i (c'est-à-dire, la décomposition en binaire) peut se faire en cours de boucle.

Aujourd'hui

1. Fonctions à sens unique

2. Chiffrement asymétrique

3. RSA
   - Le système de chiffrement
   - Implantation rapide
   - Cryptanalyse

Plan

Fonctions à sens unique

La cryptographie asymétrique repose sur les concepts de **fonctions à sens unique** (*one-way function*) et de **trappe** (*trapdoor*).

Définition. Une **fonction à sens unique** est une fonction $f : \mathcal{X} \to \mathcal{Y}$ telle que :

1. il est « facile » de calculer $f(x)$ pour tout $x \in \mathcal{X}$,
2. pour presque tout $y \in \mathcal{Y}$ de la forme $y = f(x)$, il est « difficile » de calculer x à partir de y seulement.

La notion de « facilité » se réfèrent à une **capacité de calcul**.

- On suppose actuellement que des calculs nécessitant plus de 2^{80} opérations binaires sont très difficiles, et ceux plus de 2^{128} opérations binaires sont infaisables.

  Penser : 2^{30} op. = 1 seconde sur un processeur.

- On parlera souvent de complexité **polynomiale** ou **exponentielle** par rapport à un ou plusieurs paramètres du système.

Exemples de fonctions à sens unique (1)

Exemple 1. la fonction d'exponentiation dans un groupe cyclique $\mathbb{G}$ d'ordre N. On se fixe un générateur g de $\mathbb{G}$.

$$\begin{array}{rccc} f: & \{0,\dots,N-1\} & \to & \mathbb{G} \\ & x & \mapsto & g^x \end{array}$$

1. L'évaluation de f est **rapide** : calculer $f(x)$ demande $O(\log(N))$ opérations dans le groupe, grâce à l'algorithme d'exponentiation binaire.
2. Lorsque $\mathbb{G}$ et N sont bien choisis, inverser f est **coûteux**. C'est le problème du **logarithme discret**.
   $\to$ Dans un groupe $\mathbb{G}$ **générique**, Shoup a démontré que l'on ne peut pas avoir une complexité meilleure que $\Theta(\sqrt{p})$, où p est le plus grand facteur premier de N.

En pratique, pas de « groupe générique ». On choisit par exemple

– le groupe multiplicatif $\mathbb{F}_q^\times$: si q est bien choisi, la meilleure complexité est en

$$O(\exp(c(\log q)^{1/3}(\log\log q)^{2/3})) \quad \text{où } c>0 \text{ est une constante;}$$

– le groupe de points d'une courbe elliptique $\mathcal{E}$ sur $\mathbb{F}_q$. Si la courbe $\mathcal{E}$ et q sont bien choisis, la meilleure complexité est en

$$O(\sqrt{q}).$$

$\to$ à la base du protocole de Diffie–Hellman, du chiffrement ElGamal, de la signature DSA.

Exemples de fonctions à sens unique (2)

Exemple 2. La fonction de multiplication de deux grands nombres premiers. On note $\mathcal{P} = \{(p,q) \mid p,q \text{ nombres premiers distincts} \leq d\}$. On définit

$$\begin{array}{rccc} f: & \mathcal{P} & \to & \mathbb{Z} \\ & (p,q) & \mapsto & N = pq \end{array}$$

1. L'évaluation de f est **rapide** : calculer un produit d'éléments plus petits que d est polynomial en $\log(d)$ (méthode naïve : $\log^2(d)$).
2. Pour d assez grand, inverser f est **coûteux** en pratique. C'est le problème de la **factorisation d'entiers**.

En pratique, il existe des algorithmes qui factorisent en temps

$$O(\exp(c(\log N)^{1/3}(\log\log N)^{2/3})) \quad \text{où } c > 0 \text{ est une constante.}$$

$\to$ à la base du chiffrement RSA.

Exemples de fonctions à sens unique (3)

On peut « dériver » des fonctions à sens unique à partir d'autres.

Exemple 3. Soit $N = pq$ difficile à factoriser. Alors, la fonction carré modulaire est à sens unique. On pose $\mathcal{X} = \mathcal{Y} = \mathbb{Z}/N\mathbb{Z}$ et

$$\begin{array}{rccc} f : & \mathbb{Z}/N\mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & x^2 \mod N \end{array}$$

1. L'évaluation de f est **rapide** : calculer un carré modulo N a un coût en $O(\log^2 N)$.
2. Inverser f correspond à **extraire une racine carrée** modulo N. On peut démontrer que c'est équivalent à factoriser N.

$\to$ à la base du chiffrement de Rabin.

Fonctions à trappe

Définition. Une fonction à sens unique $f : \mathcal{X} \to \mathcal{Y}$ admet une **trappe** T si la connaissance de T permet de calculer facilement tout x à partir de $y = f(x) \in \mathcal{Y}$.

On reprend l'**exemple 3.** Soit $N = pq$ difficile à factoriser. Alors, la fonction carré modulaire est à sens unique. On pose $\mathcal{X} = \mathcal{Y} = \mathbb{Z}/N\mathbb{Z}$ et

$$\begin{array}{rccc} f : & \mathbb{Z}/N\mathbb{Z} & \to & \mathbb{Z}/N\mathbb{Z} \\ & x & \mapsto & x^2 \mod N \end{array}$$

Ici, une **trappe** consiste en la connaissance d'un exposant d tel que $x = x^{2d} \mod N$.

- On peut calculer $f(x)^d \mod N = (x^2 \mod N)^d \mod N = x \mod N$.
- d vaut $2^{-1} \mod \phi(N)$ où $\phi(N)$ est la fonction indicatrice d'Euler. Ainsi, $\phi(N)$ est également une trappe.

Plan

Définitions

Un schéma de **chiffrement asymétrique** (ou **à clé publique**) engage deux entités (Alice et Bob) et se décompose en trois étapes.

1. **Génération de clés.** Alice engendre une paire de clés $(\mathtt{pk}, \mathtt{sk})$ où :
   - $\mathtt{pk}$ est la **clé publique** d'Alice (distribuée à tout le monde, dont Bob)
   - $\mathtt{sk}$ est la **clé privée** d'Alice (gardée secrètement)

   L'algorithme de génération de clés, KeyGen, prend en entrée un **paramètre de sécurité** désirée, ainsi que les paramètres du système.
2. **Chiffrement.** Bob souhaite envoyer un message m à Alice. Pour cela, Bob utilise la clé publique $\mathtt{pk}$ d'Alice :
$$c = \mathsf{Enc}(m, \mathtt{pk})$$
3. **Déchiffrement.** Alice souhaite déchiffrer un chiffré c qui lui est adressé. Pour cela, Alice utilise sa clé privée $\mathtt{sk}$:
$$m' = \mathsf{Dec}(c, \mathtt{sk})$$

Le schéma de chiffrement est **valide** si pour tout message m, on a

$$\mathsf{Dec}(\mathsf{Enc}(m, \mathtt{pk}), \mathtt{sk}) = m\,.$$

Sécurité : types d'attaques

On distingue **différents types d'attaques** (= résultat de l'attaque) sur un chiffrement asymétrique.

Par ordre d'importance décroissant :

1. Le **cassage total**. Dans cette attaque, l'adversaire récupère un moyen (algorithme efficace, données) de déchiffrer **tous** les messages chiffrés à destination d'Alice.
   $\rightarrow$ une manière d'opérer un cassage total est de retrouver la clé privée d'Alice.
2. Le **cassage partiel**. Dans ce type d'attaque, l'adversaire retrouve
   - une information partielle sur la clé, ou les messages de Bob
   - ou bien, le message de Bob avec bonne probabilité
3. La **distinction de l'aléa**. Ce type d'attaque correspond à la capacité à distinguer un texte aléatoire (dans l'espace des chiffrés) d'un chiffré particulier.

Sécurité : modes d'attaques

On distingue **différents modes d'attaques** (= moyens de l'attaquant) sur un chiffrement asymétrique.

Par ordre de pouvoir de l'attaquant :

1. **Attaque à chiffré seul.** On donne à l'attaquant une série de chiffrés $c_1, \ldots, c_k$. Étant donné un nouveau chiffré c_{k+1}, il faut attaquer le système.
2. **Attaque à clair connu.** On donne à l'attaquant une série de couples clairs/chiffrés $(m_1, c_1), \ldots, (m_k, c_k)$. Étant donné un nouveau chiffré c_{k+1}, il faut attaquer le système.
3. **Attaque à clair choisi.** On donne à l'attaquant une série de couples clairs/chiffrés $(m_1, c_1), \ldots, (m_k, c_k)$ pour lesquels il peut choisir la valeur du clair. Étant donné un nouveau chiffré c_{k+1}, il faut attaquer le système.
4. **Attaque à chiffré choisi.** Dans une première phase, l'attaquant peut appeler la procédure de chiffrement et de déchiffrement sur n'importe quels clairs et n'importe quels chiffrés. Puis, on lui fournit un autre chiffré et il doit attaquer le système.

   On distingue souvent si l'attaquant est adaptatif (CCA2) ou non (CCA1).

Remarque. Lorsque le type d'attaque est la distinction, on parle de sécurité IND-CPA pour l'attaque à clair choisi, et IND-CCA1/IND-CCA2 pour l'attaque à chiffré choisi.

Plan

Plan

Un peu d'histoire

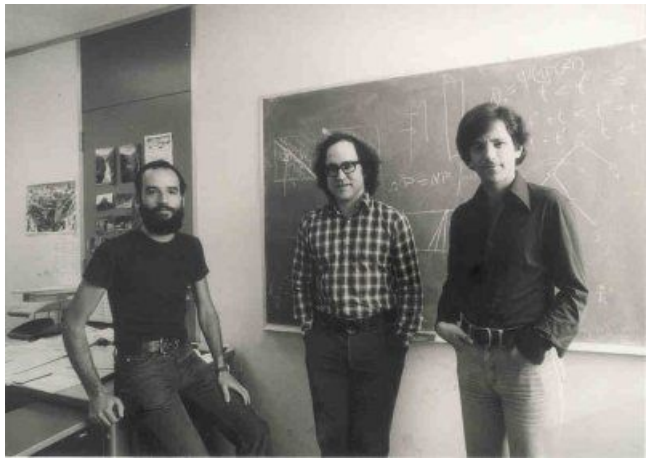
Historique.

- En 1976, Diffie et Hellman publient leur article fondateur de la cryptographie à clé publique.

  New directions in cryptography. W. Diffie, M. Hellman. IEEE Trans. Inf. Theory. **1976**.

- L'année suivante naît le schéma de chiffrement RSA (Rivest, Shamir, Adleman). Publié dans :

  A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Rivest, Shamir, Adleman. Comm. ACM. **1978**. `[lien]`



- Breveté par le MIT en 1983. Brevet expiré en 2000.
- En 1973, le britannique Cocks décrit un algorithme similaire, « trop coûteux » pour l'époque.

- Actuellement utilisé dans beaucoup de protocoles et spécifications : SSL/TLS, IPSec, S/MIME (email)
- Certifié et standardisé (PKCS # 1, ANSI X9)

RSA : génération de clés

Génération de la paire de clés RSA

Effectué par Alice (algorithme KeyGen) :

1. Choisir aléatoirement deux grands nombres premiers p et q.
2. Calculer $n = pq$ et $\phi(n) = (p-1)(q-1)$.
3. Choisir un élément $e \in \{2, \ldots, \phi(n)-1\}$ premier avec $\phi(n)$.
4. Calculer d tel que $de \equiv 1 \mod \phi(n)$.
5. Retourner les clés suivantes :
   - clé publique $\texttt{pk} = (n, e)$
   - clé privée $\texttt{sk} = d$

Remarques.

- Danc ce cadre, pour Alice il n'est pas nécessaire de garder p et q.
- Il n'est pas nécessaire que e soit choisi aléatoirement.
- Il y a des contraintes supplémentaires sur la génération de p, q, e et d afin que le cryptosystème soit sûr.

RSA : chiffrement et déchiffrement

L'espace des clairs et celui des chiffrés sont $\mathbb{Z}/n\mathbb{Z}$.

Pour chiffrer $m \in \mathbb{Z}/n\mathbb{Z}$, Bob utilise la clé publique $\texttt{pk} = (n, e)$ d'Alice :

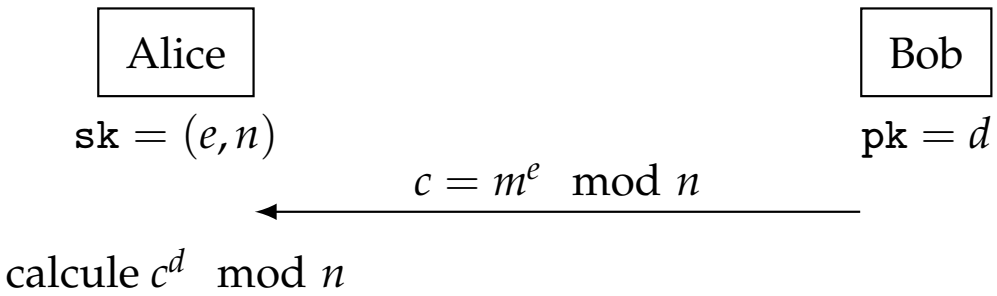
CHIFFREMENT RSA : $\mathsf{Enc}(\cdot, (n, e))$

1. Calculer $c = m^e \mod n$
2. **Retourner** $\mathsf{Enc}(m, (n, e)) = c$

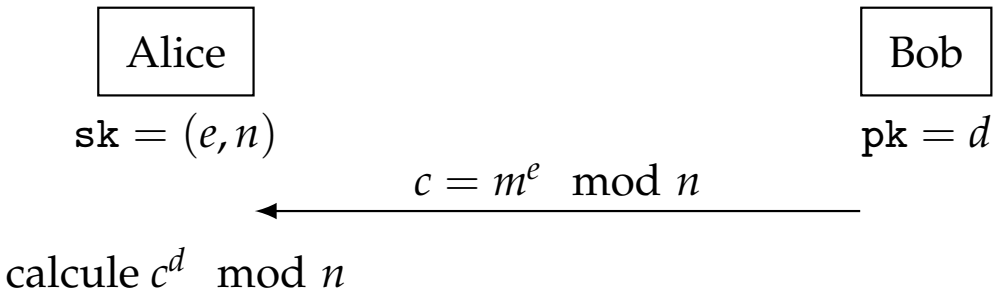
Pour déchiffrer $c \in \mathbb{Z}/n\mathbb{Z}$, Alice utilise sa clé secrète $\texttt{sk} = d$:

DÉCHIFFREMENT RSA : $\mathsf{Dec}(\cdot, d)$

1. Calculer $m' = c^d \mod n$
2. **Retourner** $\mathsf{Dec}(c, d) = m'$



RSA est valide



Vérification de la validité de RSA. On a

$$m' \equiv c^d \equiv m^{ed} \mod n\,.$$

Pour $m = 0$, la validité est évidente. Sinon, on distingue deux cas :

- Si m est premier avec n, alors $m \in (\mathbb{Z}/n\mathbb{Z})^\times$, qui a pour ordre $\phi(n)$. Donc, par le théorème d'Euler, on a $x^{\phi(n)} \equiv 1 \mod n$. Puis, comme $ed \equiv 1 \mod \phi(n)$, on obtient

$$m' \equiv m^{ed} \mod \phi(n) \equiv m \mod n\,.$$

- Sinon : ou bien p, ou bien q divise m. Prenons par exemple p. Alors :
  - m est premier avec q donc $m^{q-1} \equiv 1 \mod q$;
  - comme $\phi(n) = (p-1)(q-1)$, on a $m^{ed} \equiv m \mod q$ car $ed \equiv 1 \mod \phi(n)$;
  - pour finir, p et q divisent $m^{ed} - m$, ce qui implique que $m^{ed} \equiv m \mod n$.

Exemple-jouet pour RSA

Alice choisit $n = 23 \times 41 = 943$ et $e = 3$.

Elle publie $\texttt{pk} = (e, n)$ et garde secrètement $\texttt{sk} = d = e^{-1} \bmod \phi(n) = 587$.

Pour chiffrer $m = 111$, Bob calcule $111^3 \bmod 943$ qui vaut $c = 281$.

Pour déchiffrer $c = 281$, Alice calcule $281^{587} \bmod 943$, qui vaut bien $111 = m$.

Plan

Implantation rapide de RSA : génération de clés

Pour la **génération des clés**, on doit :

1. engendrer deux grands nombres premiers p et q
   - on tire aléatoirement p et q et on teste leur primalité
   - historiquement : algorithme de Solovay–Strassen ($\simeq$ Euler–Jacobi)
   - puis remplacé par Miller–Rabin, plus efficace
     $\rightarrow$ complexité en $O(k \log^3 n)$ pour s'assurer la primalité de p, q avec probabilité $\geq 1 - O(2^{-k})$
2. calculer le produit de p et q $\quad \Longrightarrow$ complexité binaire $O(\log^2 n)$
3. calculer un inverse modulaire $\quad \Longrightarrow$ complexité binaire $O(\log^2 n)$

Complexité totale en $O(k \log^3 n)$, où k est une constante non négligeable.

Implantation rapide de RSA : chiffrement

Pour le **chiffrement**, il faut calculer $x^e \mod n$.

On utilise l'**exponentiation rapide**.

- Alice a intérêt à choisir e petit pour que Bob ait peu de multiplications / réductions modulaires à effectuer
- On verra que $e = 3$ (en général, e petit) pose problème (attaque de Håstad)
- En pratique, on essaie de prendre $e = 2^{16} + 1 = 65537$, le quatrième nombre de Fermat.
  - $\rightarrow$ sauf si e divise $\phi(n)$ (c'est très rare)
  - $\rightarrow$ $\log(e)$ est une petite constante $\implies$ la complexité du chiffrement est en $O(\log^2 n)$

Implantation rapide de RSA : déchiffrement

Pour le **déchiffrement**, il faut calculer $x^d \mod n$.

- Avec l'**exponentiation rapide**, on a une complexité en $O(\log d \log^2 n)$.
- **Idée d'accélération :** Alice peut précalculer

$$d_p := d \mod (p-1) \qquad \text{et} \qquad d_q = d \mod (q-1)$$

Grâce au théorème des restes chinois, Alice reconstruit x^d à partir des valeurs

$$x^{d_p} \mod p \quad \text{et} \quad x^{d_q} \mod q\,.$$

Pour cela, elle calcule u et v tels que $up + vq = 1$. Puis :

$$x^d = (x^{d_p} \mod p)vq + (x^{d_q} \mod q)up \mod n\,.$$

$\Longrightarrow$ Division par $\simeq 4$ du temps de calcul (mais surcoût de stockage).

Remarque. Pour éviter les **attaques par canaux auxiliaires** (attaques matérielles, analyse de temps de calcul), il faut implémenter le déchiffrement en temps constant (c'est-à-dire que le temps de déchiffrement doit être indépendant de d).

Plan

Factorisation de n

La sécurité de RSA repose sur le problème de la **factorisation**.

C'est un problème encore supposé difficile (mais pas NP-difficile).

Algorithmes de résolution **exponentiels** :

- division successives en $O(\sqrt{n})$
- méthode ρ de Pollard en $O(\sqrt{\min\{p,q\}}) = O(n^{1/4})$

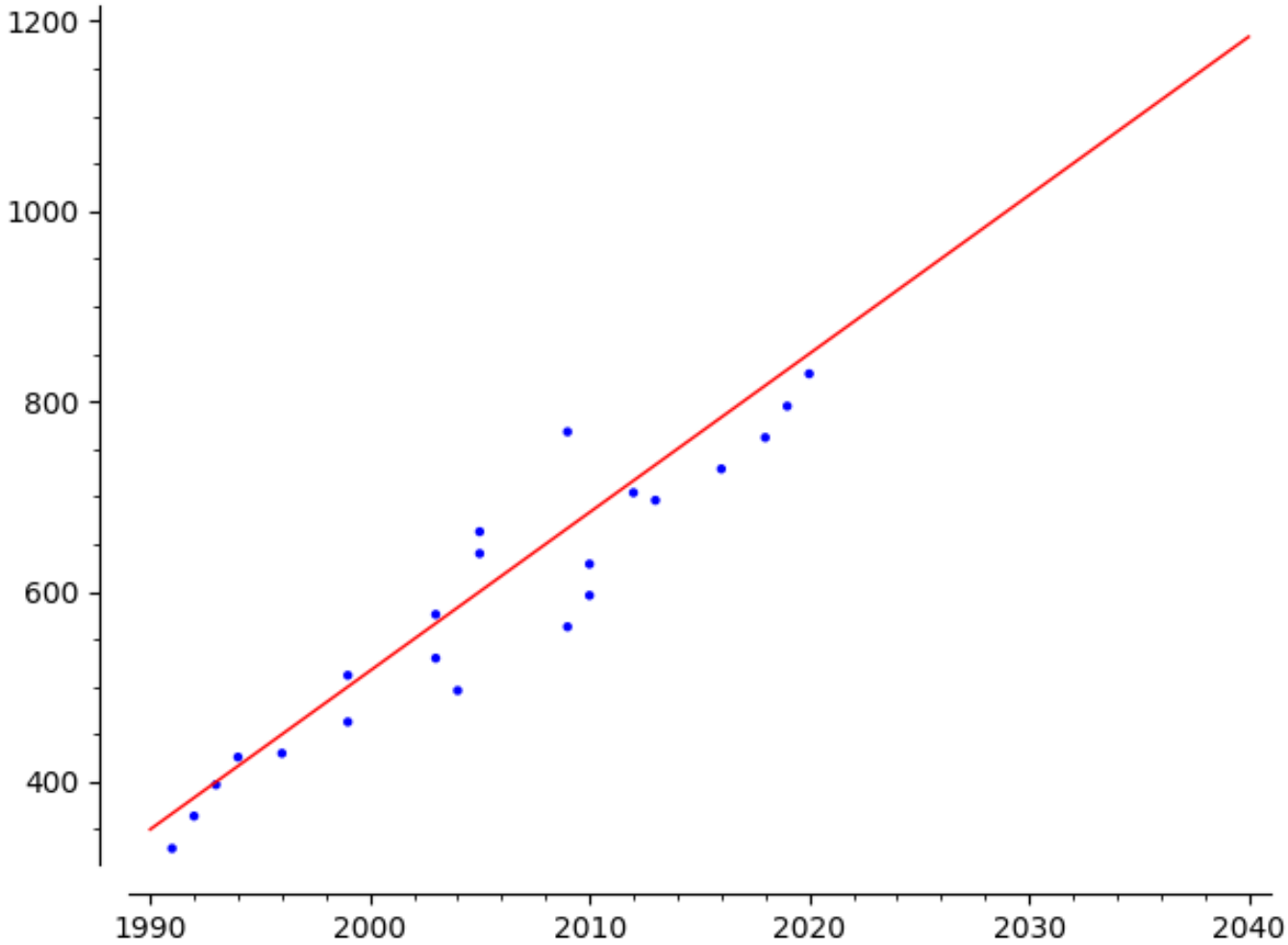
Algorithmes de résolution **sous-exponentiels** :

- méthode ECM de Lenstra, crible quadratique en $O(\exp(c\sqrt{\log n \log\log n}))$ où $c > 0$,
- crible algébrique en $O(\exp(c'(\log n)^{1/3}(\log\log n)^{2/3}))$ où $c' \simeq 1{,}92 > 0$.

$\rightarrow$ cours de M2 sur les algorithmes arithmétiques

Factorisation de n : records historiques

TAILLE EN BITS DES RECORDS DE FACTORISATION DE MODULES RSA EN FONCTION DE L'ANNÉE (EN BLEU), ET EXTRAPOLATION (EN ROUGE).



Factorisation de n : record

Le **record** de factorisation de modules RSA est RSA-250, effectuée en février 2020 :

$$
\begin{aligned}
& 214032465024074496126442307283933356300861471514475501779775492088141802344714013664 \\
& 334551909580467961099285187247091458768739626192155736304745477052080511905649310668 \\
& 769159001975940569345745223058932597669747168173806936489469987157849495937497937 \\
= \; & 641352894770715802787901901705773890848250147429434472081168596320245323446302386235 \\
& 98752668347708737661925585694639798853367 \\
\times \; & 333720275949781565562260106053551142279407603447675546667845209870238417292100370802 \\
& 57448673296881877565718986258036932062711
\end{aligned}
$$

Temps de factorisation : équivalent 2700 années (oui !) de calcul sur 1 cœur.

Source :

`https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html`

Factorisation de n : algorithme de Fermat

Si p et q sont très proches, il y a également possibilité de factoriser $n = pq$.

Remarque fondamentale. Si $n = pq$, alors

$$n = (t+s)(t-s) = t^2 - s^2 \quad \text{où} \quad t = \frac{p+q}{2} \text{ et } s = \frac{p-q}{2}.$$

Idée : En partant de $t = \lceil \sqrt{n} \rceil$, on teste si $c := t^2 - n$ s'écrit comme un carré s^2.
Si c'est le cas, on a trouvé $p = t + s$ et $q = t - s$.

ALGORITHME DE FERMAT POUR LA FACTORISATION D'ENTIERS

1. Calculer $t = \lceil \sqrt{n} \rceil$ et $c = t^2 - n$.
2. **Tant que** c n'est pas un carré :
   - incrémenter $t \leftarrow t + 1$
   - calculer $c \leftarrow c + 2t + 1$
3. Calculer une racine carrée s de c.
4. **Retourner** $t + s$ et $t - s$.

Complexité en $O((p - \sqrt{n})^2/2p)$ où $p > q$.

Si $|p - q| \leq an^{1/4}$, alors $p \simeq \sqrt{n} + O(an^{1/4})$ et on a une attaque en $O(a^2)$.

Algorithme de Fermat : exemples

Exemple 1 : $n = 2183$.

On a $\lceil \sqrt{n} \rceil = 47$.

t	$t^2 - n$	carré ?
47	26	$\times$
48	121	✓

Donc $n = (48 - 11)(48 + 11) = 59 \times 37$

Exemple 2. $n = 2581$.

On a $\lceil \sqrt{n} \rceil = 51$.

t	$t^2 - n$	carré ?
51	20	$\times$
52	123	$\times$
53	228	$\times$
54	335	$\times$
55	444	$\times$
56	555	$\times$
57	668	$\times$
58	783	$\times$
59	900	✓

Donc $n = (59 - 30)(59 + 30) = 29 \times 89$.

Attaque de Wiener

En 1990, Wiener donne une attaque en temps polynomial lorsque d a été choisi trop petit par rapport à n.

Théorème (Wiener). Soit $n = pq$ un module RSA tel que $q < p < 2q$ et $de \equiv 1 \mod \phi(n)$. Si $d \leq \frac{1}{3} n^{1/4}$, alors il existe un algorithme de complexité $O(\log^3(n))$ qui calcule d à partir de n et e.

L'idée est de calculer les **réduites** de la **fraction continue** associée à e/n, et de noter que l'une d'elle sera de la forme k/d où $\text{pgcd}(k, d) = 1$.

Attaque de Coppersmith

Théorème (Coppersmith). Soit $P \in \mathbb{Z}[X]$ de degré D et $n \geq 2$. Il existe un algorithme de complexité polynomiale en $\log(n)$, qui retrouve l'ensemble des racines x de P modulo n qui sont telles que $|x| \leq n^{1/D}$.

Soit maintenant $((n,e),d)$ une paire de clés RSA et m un message chiffré en $c = m^e \mod n$.

Conséquence du théorème. Si on connaît une approximation de m à $n^{1/e}$ près, alors on peut retrouver m en temps polynomial en $\log(n)$.

En effet, si l'on sait que $m = m_0 + x$ avec m_0 connu et $|x| \leq n^{1/e}$, alors on pose

$$P(X) = (X + m_0)^e - c\,.$$

On a alors $P(x) = 0$, et on peut retrouver x, donc m, grâce au théorème précédent.

Cette attaque s'étend au contexte suivant :

Attaque de Coppersmith. Soient deux chiffrés $c_1 = m_1^e$ et $c_2 = m_2^e$ pour la même paire de clés RSA $((n,e),d)$. Alors, si $|m_1 - m_2| \leq n^{1/e^2}$, on peut déterminer m_1, m_2 en temps polynomial, avec bonne probabilité.

Remarque. Ce sont des attaques spécifiques à certains messages (avec connaissance partielle), qui ne remettent pas en question la sécurité de la fonction à sens unique.

Autres attaques

D'**autres attaques** peuvent être montées dans des contextes particuliers.

On peut démontrer que :

1. Si l'on connaît $\phi(n)$, alors on peut aisément factoriser n (cf exercice)
2. Si l'on connaît d, alors on peut factoriser n en temps polynomial avec un algorithme probabiliste. Si d est révélé, Alice doit donc également changer n.
3. Si un même message est envoyé à $\geq e$ utilisateurs ayant des clés différentes, alors un attaquant peut retrouver le message (attaque de Håstad, cf exercice).

Remarque. Les attaques de Håstad et de Coppersmith ne s'appliquent plus lorsqu'on utilise une conversion sémantiquement sûre pour RSA (voir prochain cours).

Résumé des bons/mauvais choix de paramètres

1. **Choix de p et q :**
   - p et q doivent être de taille suffisante pour **éviter une factorisation** de n
     $\rightarrow$ En pratique, n doit être de 2048 bits minimum.
     $\rightarrow$ Pour de la sécurité à longue durée (2030 et postérieur), l'ANSSI encourage à prendre n de taille ≥ 3072 bits.
   - p et q ne doivent pas être trop proches : $|p - q| \gg n^{1/4}$
     $\rightarrow$ Remarque : ce n'est pas très contraignant en pratique

2. **Choix de e et d :**
   - On doit éviter e trop petit, typiquement $e = 3$ est à proscrire (Coppersmith, Håstad, ...)
     $\rightarrow$ En pratique $e = 2^{16} + 1 = 65537$, si possible
   - On doit éviter d trop petit.
     $\rightarrow$ Pour $d < \frac{1}{3} n^{1/4}$ on a l'attaque de Wiener (1990).
     $\rightarrow$ Boneh et Durfee (1999) ont donné une attaque lorsque $d < n^{0292}$.

Fin de la partie cours

Questions ?