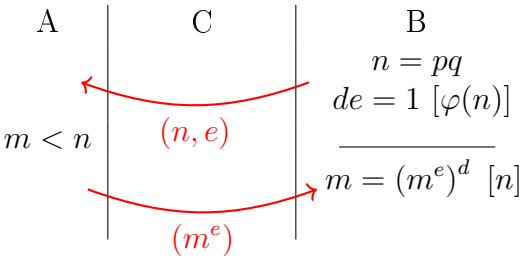
Réduction RSA

Alice (A) veut envoyer un message à Bob (B). Mais Charlie (C) peut intercepter leurs conversations. On cherche donc un moyen de coder les messages de sorte que seuls Alice et Bob puissent les comprendre bien que tous puissent les lire.

Définition 1 *Le cryptosystème RSA fonctionne comme suit. Bob choisit deux nombre premiers impairs distincts (positifs) p et q et fixe ainsi un nombre $n = pq$. Il choisit ensuite un entier $e \in \mathbb{N}$ (pour encode) inversible modulo $\varphi(n)$ où φ désigne l'indicatrice d'Euler. On notera par la suite $d \in \mathbb{N}$ (pour décode) tel que $de = 1\ [\varphi(n)]$. Le couple (n, e) est appelé clef publique et est partagé tandis que le couple (n, d) est appelé clef privée et est gardé secret. Alice veut envoyer un message $m \in \mathbb{Z}$. Quitte à le découper en morceaux et l'envoyer en plusieurs fois, on peut supposer que $m < n$. Alice partage alors m^e. Bob peut alors retrouver m par la relation $m = (m^e)^d \mod n$. La sécurité de ce système repose sur le fait qu'on ne sache pas calculer le logarithme discret efficacement.*



Propriété 2 *Pour tout $m < n$ on a $(m^e)^d \mod n = m$.*

Preuve 3 *On regarde modulo p et q.*
Si $m = 0\ [p]$ alors $m = 0 = (m^e)^d\ [p]$.
Sinon $de = 1 + k\varphi(n)$ pour un certain $k \in \mathbb{Z}$ donc $(m^e)^d = m^{de} = m^{1+k\varphi(n)} = m * (m^{p-1})^{q-1} = m\ [p]$ car $\varphi(n) = (p-1)(q-1)$.
Par symétrie, on a aussi $m = (m^e)^d\ [q]$.
On conclut par injectivité dans le lemme chinois et le fait qu'on a choisi $m < n$.

Remarque 4 *Quelques remarques sur la complexité des opérations dans ce protocole :*

- *trouver de grands nombres premiers est couteux, c'est usuellement fait avec l'algorithme de Miller-Rabin de façon probabiliste.*
- *de par sa forme, le calcul de $\varphi(n) = (p-1)(q-1)$ se fait en $O(1)$ multiplications.*
- *trouver l'inverse modulaire se fait à l'aide de l'algorithme d'Euclide étendu qui est en $O(\log(n)^2)$ opérations.*
- *les calculs de puissance se font avec l'algorithme d'exponentiation rapide en $O(\log(n))$ multiplications.*

Théorème 5 *Retrouver d à partir de (n, e) et m^e équivaut[1] à factoriser n à l'aide d'un algorithme probabiliste.*

[1] au sens de la reduction de complexité, c'est à dire qu'il existe un algorithme polynômial en la taille des données qui permet de déduire l'un à partir de l'autre et réciproquement.

Remarque 6 *Alors que j'écris ceci, le problème de si craquer RSA équivaut à la factorisation de n est encore ouvert, on se contente donc de montrer ce résultat plus faible. On peut aussi (facilement) montrer que retrouver $\varphi(n)$ équivaut à factoriser n ou d'autres propriétés garantissant malgré tout sa sécurité.*

$\boxed{\Leftarrow}$
Si l'on peut factoriser n à l'aide d'un oracle, on retrouve p et q. Cela nous donne $\varphi(n) = (p-1)(q-1)$ et donc il nous suffit de calculer $d = e^{-1} \mod \varphi(n)$ comme Bob l'a fait. Toute ces opérations se font en $O(\log(n)^2)$ par l'algorithme d'Euclide étendu.

$\boxed{\Rightarrow}$
Supposons maintenant que l'on connaisse d, montrons que l'on peut factoriser n avec un algorithme probabiliste en temps polynômial en $\log(n)$.

Lemme 7 *Soit $c \in \mathbb{Z}$ tel que $c^2 = 1\ [n]$ mais $c \neq \pm 1\ [n]$, alors $pgcd(n, c+1)$ est un facteur non trivial de n (c'est à dire p ou q).*

Preuve 8 *On a $n \mid c^2 - 1$ mais $n \nmid c-1$ et $n \nmid c+1$. Donc soit $\begin{cases} p \mid c+1 \\ q \mid c-1 \end{cases}$ soit $\begin{cases} p \mid c-1 \\ q \mid c+1 \end{cases}$. Par symétrie, si l'on est dans le premier cas, on a $p \mid c+1$, $q \nmid c+1$ (car q est impair) donc $pgcd(n, c+1) = p$ est une facteur non trivial.*

On va maintenant présenter un algorithme probabiliste qui renvoie un facteur non trivial avec une probabilité d'au moins $\frac{1}{2}$.

Entrée : Un entiers $n \in \mathbb{N}$ produit de deux nmobre premiers et $d, e \in \mathbb{N}$ tels que $de = 1\ [\varphi(n)]$.
Sorie : un facteur non trivial de n ou une erreur.

1: Choisir au hasard $1 < a < n$.
2: Si $pgcd(n, a) \neq 1$ on a trouvé un facteur non trivial. $O(\log(n)^2)$
3: Sinon on pose $de - 1 = 2^s t$ avec t impair et $s \geq 1$. $O(\log(n))$
4: Si $a^t = \pm 1$ on ne peut rien dire, renvoyer erreur. $O(\log(n)^3)$
5: Posons $l := \min\{k \in [\![2, n]\!] \mid a^{2^k t} = 1\}$. $O(\log(n)^3)$
6: Si $a^{2^{l-1}t} = \pm 1$, on ne peut rien dire, renvoyer erreur.
7: Sinon renvoyer $pgcd(n, a^{2^{l-1}t})$. $O(\log(n)^2)$

Correction : l'algorithme ne renvoie un résultat qu'à la ligne 2 et 7. Le cas de la ligne 2 est trivial. La ligne 7 renvoie un facteur non trivial par le lemme, en effet $a^{2^{l-1}t}$ est alors une racine carré de l'unité non triviale par minimalité de l et le test de la ligne 4.
On remarquera de plus que $de = 1 + k\varphi(n)$ est impair car $\varphi(n)$ est pair donc $s \geq 1$ et $a^{2^s t} = 1$ (cf prop 2) d'où l'ensemble ligne 5 est non vide et le min est bien défini.

Complexité : le calcul des pgcd se fait en $O(\log(n)^2)$ opérations avec l'algorithme d'Euclide. Pour obtenir la décomposition 2-adique il suffit de diviser par 2 tant qu'on le peut, cela se fait en $O(\log(n))$ car la division par 2 est une opération élémentaire en binaire. Le calcul de a^t se fait par exponentiation rapide en $O(\log(n))$ multiplications c'est-à-dire en $O(\log(n)^3)$ opérations machine (cf complexité de la multiplication d'entiers naïve). Pour obtenir l on élève au carré jusqu'à ce que l'on vérifie la condition demandée. Il y a au plus s étapes dans la boucle, soit $O(\log(n))$ passages au carré (ie multiplications). On a donc besoin de $O(\log(n)^3)$ opérations pour trouver l. Il suffit de garder en mémoire le résultat précédent pour éviter d'avoir à recalculer $a^{2^{l-1}t}$ et les comparaisons sont en $O(1)$. Finalement, l'algorithme s'exécute en $O(\log(n)^3)$ qui est bien polynômial en la taille de l'entrée.

Probabilité d'erreur : on va montrer que la probabilité de tomber sur un cas où on ne peut rien dire est inférieur à $\frac{1}{2}$.

On se concentre sur les lignes 3 à 7, on se fixe (s,t) comme dans l'algorithme et on cherche parmi les a premiers avec n lesquels conduisent à un échec.
Posons $I = \{k \in [\![1,s]\!] \mid \forall a \in (\mathbb{Z}/n\mathbb{Z})^\times \ a^k = 1\}$. On a

- $0 \notin I$ car t est impair donc $(-1)^t = -1 \neq_1 [n]$ (n>2).
- $s \in I$ (cf prop 2)
- $\forall k \in I,\ k+1 \in I$. En effet on se contente d'élever 1 au carré.

On note donc $I = \{l'+1, \dots, s\}$ avec $l' \in [\![0, s-1]\!]$.
On pose maintenant $G = \{a \in (\mathbb{Z}/n\mathbb{Z})^\times \mid a^{2^{l'}t} = \pm 1\}$. On a alors :

- G est un sous-groupe de $(\mathbb{Z}/n\mathbb{Z})^\times$.
- G contient les cas d'échecs (lignes 4 et 6). En effet si $a \in (\mathbb{Z}/n\mathbb{Z})^\times$ et $a^t = \pm 1$ alors, vu que $l' \geq 0$, $a^{2^{l'}t} = (a^t)^{2^{l'}} = \pm 1$. Sinon, pour l fixé comme dans l'algorithme, si $a^{2^{l-1}t} = \pm 1$ alors $l-1 \leq l'$ par construction de l'. Ainsi, en passant au carré suffisamment de fois, on a $a^{2^{l'}t} = \pm 1$.
- G est un sous-groupe stricte. En effet il existe, par minimalité de l', un $a \in (\mathbb{Z}/n\mathbb{Z})^\times$ tel que $a^{2^{l'-1}t} \neq 1$. Si $a^{2^{l'-1}t} \neq -1$ alors $a \notin G$. Sinon par le lemme chinois il existe $b \in \mathbb{Z}/n\mathbb{Z}$ tel que $b \equiv 1\ [p]$ et $b \equiv a\ [q]$. On a alors que $b \in (\mathbb{Z}/n\mathbb{Z})^\times$, $b^{2^{l'-1}t} = 1\ [p]$ et $b^{2^{l'-1}t} = a^{2^{l'-1}t} = -1\ [q]$. Ainsi par le lemme chinois $b \in (\mathbb{Z}/n\mathbb{Z})^\times \backslash G$.

On a donc que $\#G \leq \frac{\#(\mathbb{Z}/n\mathbb{Z})}{2} \leq \frac{n}{2}$ par le théorème de Lagrange.
On a donc au plus la moitié des $a \in \mathbb{Z}/n\mathbb{Z}$ qui peuvent poser problème. On a donc en moyenne besoin d'appeler que 2 fois l'algorithme pour obtenir un facteur non trivial.

Remarques :

- Par manque de temps, je conseille de juste faire le schéma d'explication puis de passer directement au théorème 5 en évoquant éventuellement a l'oral les points écrits.
- Il est intéressant (important) d'avoir des bases de théorie de la complexité. Au moins savoir ce qu'est une réduction ce qui est le cœur de la preuve.
- Le meilleur algorithme connu à ces jours pour factoriser un entier (le crible algébrique) est de complexité sous-exponentielle, on reste loin du polynômial.
- Un CPU 1 cœur fait environ 10^9 opérations par secondes, soit $10^9 * 60 * 60 * 24 * 365 \approx 10^{17}$ opérations par an. L'algorithme naïf de recherche de facteurs premiers d'un nombre n à 1000 chiffres (jusqu'à $\sqrt{n}$) demandera environ 10^{500} opérations. Il faudra donc 10^{500-17} années pour factoriser n avec ces méthodes, on est en sécurité !
- Un calcul technique permet de calculer le nombre exacte de cas d'erreur (au lieu de notre majoration au karcher mais suffisante). Vous pourrez la trouver dans "Carnet de voyage en Algébrie" de Philipe Caldero et Marie Peronnier.