

# Cryptographie asymétrique

19 septembre 2023

## 1 Intro

L'asymétrie ne sert pas à chiffrer mais plutôt aux échanges de clés, etc..

**Ansi** recommande

- Des clés de 80 à 100 bits pour un niveau moyen de sécu (données ne durant pas dans le temps  $\sim$  minutes)
- $> 100$  bits : forts

## 2 Arithmétique entiers

La complexité est calc en fonction de :

- La taille des données.
- ex : un entier  $n$  en représentation binaire est en  $\log_2(n) = \log(n)$ .

A regarder : table de soustraction binaire lol.

### 2.1 multiplication

$$11101 = a$$

$$\times 1101 = b$$

multiplication naive :

- Taille(b) additions d'élts de taille a.
- Complexité : Taille(a)\*Taille(b)
- Memoire : Taille(a\*b)=Taille(a)+Taille(b)

Méthode de Karatsuba :  $a, b \in \mathbb{N}$  et  $k = \log(a) = \log(b)$ .  $a = \alpha 2^{k/2} + \beta$ ,  $b = \gamma 2^{k/2} + \delta$ . On écrit :

$$ab = \alpha\gamma 2^k + (\alpha\gamma + \beta\delta - (\alpha - \beta)(\gamma - \delta))2^{k/2} + \beta\delta$$

On remarque que ya 3 multiplication d'élts de taille  $k/2$  et 6 soustr/add de taille  $k/2$ .

- Complexité :  $T(k)$  est donnée par

$$\begin{aligned} 3T(k/2) + 6O(k/2) &= 3^T(k/4) + 6 * 3O(k/4) + 6O(k/2) \\ &= 3^{\log(k)} + 2ck \sum_{i=1}^{\log(k)} (3/2)^i \\ &= 3^{\log(k)} + 2Ck \frac{(3/2)^{\log(k)} - 1}{(3/2) - 1} \\ &= \dots \\ &= O(k^{\log(3)}) \end{aligned}$$

## 2.2 division

Division naive (euclidienne) :

- Taille(a)-Taille(b)+1 soustraction de taille Taille(b).
- Complexité :  $O((\text{taille(a)}\text{taille(b)}+1)\text{taille(b)})$ .
- Mémoire : Taille(a)-Taille(b)+1 + taille(b).

## 2.3 algorithme d'euclide normal/etendu

**Lemme 2.3.1.** Avec  $a = r_0$ ,  $b = r_1$ ,  $r_i = q_{i+2}r_{i+1} + r_{i+2}$ . On a  $r_{i+2} < r_i/2$ . Sauf pour les derniers  $i$ .

D'ou

- Au plus  $\log(a)$  divisions : i.e.  $\sum_{i=0}^{k-1} (\log(r_i) - \log(r_{i+1} + 1) \log(r_i)) \leq \log(a)(k + \log(a))$
- Complexité en  $\log(a)^2$

Euclide étendu :  $u_0 = 1, u_1 = 0$  et  $v_0 = 0, v_1 = 1$  et on écrit

$$\begin{aligned} u_{i+2} &= u_i - q_i u_{i+1} \\ v_{i+2} &= v_i - q_i v_{i+1} \end{aligned}$$

Pour calculer le pgcd :

- Complexité :  $O(\log^2(a))$ . (exo)

A montrer :

**Lemme 2.3.2.**  $n$  un entier, calcul de la racine carrée entière de  $n$  en

$$O(\log^3 n)$$

## 2.4 indicatrice d'euler/inversion

**Proposition 2.4.1.**  $a^{-1} \bmod n$  se calcule en

$$O(\log^2(n))$$

grâce à euclide

**Definition 2.4.2.**  $\phi : \mathbb{Z}/n\mathbb{Z} \rightarrow \#\{0 < i \leq n\}^*$

**Proposition 2.4.3.** On veut  $\phi(1) = 1$  pour la récursion.

**Proposition 2.4.4.**  $\sum_{d|n} \phi(d) = n$

Ca se prouve en posant  $\sum_{d|n} \phi(d) = f(n)$  alors :

$$f(mn) = \sum_{d|mn} \phi(d) = \sum_{d_1|n} \sum_{d_2|m} \phi(d_1 d_2) = f(m)f(n)$$

. On écrit ducoup  $f(n) = f(\prod p_i^{\alpha_i})$  et  $f(p^\alpha) = \sum_{k < \alpha} \phi(p^k) = \sum_k p^k - p^{k-1} = p^\alpha$

**Proposition 2.4.5.**  $p \neq q$  deux nombres premiers et  $n = pq$ . On retrouve  $p, q$  en  $O(\log^3(n))$  avec  $n, \phi(n)$ .

### 3 corps finis

$$q = p^d$$

**Proposition 3.0.1.** • Complexité de l'addition/soustraction dans  $\mathbb{F}_q$  :  $O(\log(q))$

• Complexité de la mult/l'inverse dans  $\mathbb{F}_q$  :  $O(\log^2(q))$

Pour la multiplication :  $2d - 2$  calculs des sommes  $\sum a_i b_{j-i}$  et  $d$  mults a chaque fois puis  $d$  additions. A la fin  $O(\log^2(q))$ .

**Proposition 3.0.2.**  $d = \gcd(n, q - 1)$  racines n-emes de l'unité dans  $\mathbb{F}_q$ .  $\mathbb{F}_q$  admet une racine primitive ssi  $n \mid q - 1$ .

Pour le deuxieme truc  $(g^j)^n = 1$  ssi  $q - 1 \mid nj$  d'ou  $q - 1/d \mid j$  et on a  $d$  valeurs possibles pour  $j$ .

#### 3.1 résidus quadratiques

On prend  $p \neq 2$  :

**Proposition 3.1.1.**  $x \mapsto (x^{p-1/2})$  donne l'indice de  $\mathbb{F}_p^{*2}$  et deux non résidus sont des puissances impaires donc le produit est une puissance paire.

**Proposition 3.1.2.** C'est un morphisme de groupe.

Maintenant on remplace  $x \mapsto x^{p-1/2}$  par l'unique caractère abélien dans  $\{\pm\}$  (Jacobi).

#### 3.2 Calcul de racine carrée, algo de shanks tonelli

On réduit ca à un calcul de racine  $2^\alpha$ -eme de l'unité !

1. On écrit  $p - 1 = 2^\alpha * s$ ,  $s$  impair.
2.  $r = a^{(s+1)/2}$
3. on résoud  $x^2 a^{-1} \equiv 1 \mod p$
4. En gros :  $1 \equiv a^{(p-1)/2} \equiv a^{2^{\alpha-1}s} \equiv (r^2 a^{-1})^{2^{\alpha-1}} \mod p$
5. D'ou on cherche une racine de l'unité,  $z$ , alors  $z^2 \equiv y$  avec  $y = r^2 a^{-1}$ .

6.  $z^2y \equiv y^{2^{\alpha-1}} \mod p$  d'où  $(z^2y^{1-2^{\alpha-1}})^{2^{\alpha-1}} \equiv z^{2^{\alpha}}(y^{2^{\alpha-1}})^{1-2^{\alpha-1}} \equiv z^{2^{\alpha}} \equiv 1 \mod p$

7. D'où il faut trouver une racine  $2^{\alpha}$ -eme de l'unité.

Determination de la racine  $2^{\alpha}$ -eme de l'unité :

1. Pour  $\left(\frac{n}{p}\right) = -1$  on pose  $b = n^s$

2. Alors  $|b|^{2^{\alpha}}$ .

On cherche ensuite le  $b^j$  tel que  $b^{2^j}r^2a^{-1} \equiv 1 \mod p$ , on écrit  $j = j_0 + 2j_1 + \dots + 2^{\alpha-1}j_{\alpha-1}$  :

1.  $b^{2^j}r^2a^{-1} \equiv b^{2^{j_0}+\dots+2^{\alpha-1}j_{\alpha-1}} \equiv b^{2^{j_0}+\dots+2^{\alpha-1}j_{\alpha-2}} \mod p$

2. On regarde  $(b^{2^j}r^2a^{-1})^{2^{\alpha-2}} \equiv (b^{2^{\alpha-1}})^{j_0}a^{2^{\alpha-2}s} \mod p$

3. Comme  $b^{2^{\alpha-1}} \equiv n^{(p-1)/2} \equiv -1 \mod p$

4. Alors pour avoir  $(b^{2^j}r^2a^{-1})^{2^{\alpha-2}} \equiv 1$  il faut prendre  $j_0 = 0$  ssi  $(r^2a^{-1})^{2^{\alpha-1}}$

Maintenant pour les autres coeffs que  $j_0$ , on suppose qu'on connait les  $l < \alpha - 2$  premiers tq  $((b^{j_0+\dots+2^lj_l})r^2a^{-1})^{2^{\alpha-2-l}} \mod p$  on cherche  $j_{l+1}$  tq :

1.  $((b^{j_0+\dots+2^lj_l})r^2a^{-1})^{2^{\alpha-2-l}} \mod p$

2. On a  $(b^j)^{2^{\alpha-2-l}}(r^2a^{-1})^{2^{\alpha-2-l-1}} \equiv b^{2^{\alpha-2-l}(j_0+2j_1+\dots+2^lj_l)}b^{2^{\alpha-1}j_{l+1}}b^{2^{\alpha}(\dots)} \mod p$

3. A nouveau on a  $b^{2^{\alpha-1}j_{l+1}} \equiv (-1)^{j_{l+1}}$

4. Et donc on pose  $j_{l+1} = 0$  ssi  $((b^{j_0+\dots+2^lj_l})r^2a^{-1})^{2^{\alpha-2-l-1}} \equiv 1 \mod p$

## 4 Protocoles de cryptographie à clef publique

Basé sur le principe de Kerkhoff.

## Crypto symétrique

+ rapide

1 clef partagée

×

Taille de clef petite

## Crypto asymétrique

+lent

2 clefs

Mise en reseau facile

Taille de clé grande

Probleme de la crypto sym : nombre quadratique de clé par rapport au nb de personnes face a linéaire pour l'asym. (+ faut pouvoir échanger les clés)

Cryptographie asymétrique :

1. Authentification
2. Echange de clefs
3. Signature

Etant donné une fct de chiffrement asym  $f$  :

- $f(m, k_{pub}) = c$
- $f^{-1}(c, k_{priv}) = m$

Authentification par challenge :

- $f(challenge, k_{pub}) \rightarrow c$  un challenge est donné et doit être déchiffré
- $challenge = m \leftarrow f^{-1}(c, k_{priv})$

Echange de clefs:

- $k$  la clef de session qu'on veut partager
- $f(k, k_{pub}) \rightarrow c$
- $k = f^{-1}(c, k_{priv})$

Signature d'un message :

- $f^{-1}(m, k_{priv}) = sign$

- $f(\text{sign}, k_{\text{pub}}) = m$

Propriétés d'une signature :

1. Non-répudiable (irrévocable, on peut pas dire qu'on l'a pas signé)
2. Le message est non-modifiable : inaltérable
3. Authentique
4. Non-réutilisable
5. Infalsifiable

## 4.1 RSA

Décrit [ici](#). On regarde des attaques sur RSA, les  $p, q$  doivent être tous achetés !

**Definition 4.1.1.** Attaque par module commun

Etant donné une communauté de  $k$  personnes ayant tous.les  $p * q = n$ . Chaque utilisateurs recoit  $(N, e_i(\text{publique}), d_i(\text{privee}))$ . Si on connait  $e_i, d_i$  alors on sait que  $e_i d_i \equiv 1 \text{ mod } \phi(n)$  d'où  $e_i d_i = 1 + k\phi(n)$ .

On pose  $m = e_i d_i - 1 = k\phi(n)$  d'où  $\forall a \in (\mathbb{Z}/N\mathbb{Z})^\times$ ,

$$a^m \equiv 1 \text{ mod } \phi(n)$$

Or  $4 \mid \phi(n)$  donc  $4 \mid m$ .

Donc/étant donné

$$a^m \equiv 1 \text{ mod } n$$

. On a  $a^{m/2}$  est une racine carrée de 1 mod  $n$  (y'en a 4). Si  $a^{m/2} \equiv \alpha \not\equiv \pm 1 \text{ mod } n$  alors  $(\alpha - 1)(\alpha + 1) \equiv 0 \text{ mod } N$  et  $\gcd(\alpha - 1, n) \neq 1$  et  $\gcd(\alpha + 1, n) = p$ . Soit  $a \in (\mathbb{Z}/n\mathbb{Z})^\times$ . On pose :  $m = 2^t s$

- On calc  $a^s \text{ mod } n$ , si  $= \pm 1 \text{ mod } n$  on change  $a$ .
- Sinon on calc successivement  $a^{2^i s} \text{ mod } n$ . Et on s'arrete des qu'on trouve 1.
- Si a l'étape d'avant on change  $a$ .
- sinon on a trouvé  $\alpha$ .

Autre attaque : Si on chiffre  $m$  pour deux destinataire :

- $c_1 \equiv r^{e_1} \pmod n$
- $c_2 \equiv r^{e_2} \pmod n$

Si  $\gcd(e_1, e_2) = 1$  alors  $\exists u, v \in \mathbb{Z}$  tq  $ue_1 + ve_2 = 1$ . Donc  $c_1^u * c_2^v = m \pmod n$ .

- Besoin d'une fonction de hachage pour la signature

Alice ne veut pas signer  $m$ , Marvin choisit  $r \in (\mathbb{Z}/n\mathbb{Z})^\times$  et calcule  $m' = m * r^e \pmod n$ . Alice signe  $m'$ , donc Marvin obtient  $\text{sign}(m') \equiv m'^d \equiv (mr^e)^d \equiv m^d r^e$ .

Nouvelle attaque

**Definition 4.1.2.** par exposant public petit :

On propose que tout le monde utilise le même  $e$  petit pour accélérer le chiffrement:  $m$  est chiffré par  $k$  utilisateurs différents:  $\begin{cases} c_1 \equiv m^e \pmod{n_1} \\ \vdots \\ c_k \equiv m^e \pmod{n_k} \end{cases}$  Soit les  $n_i$  sont premiers entre eux et on fait un lemme chinois, si  $e < k$ ,  $m^e < \prod_i n_i$ . Si pas premiers entre eux : gros pb.

**Definition 4.1.3.** Attaque par petit exposant privé, but : améliorer la vitesse de déchiffrement.

**Théorème 4.1.4.** Soit  $N = pq$  avec  $q < p < 2q$  et  $d = 1/3\sqrt[4]{n}$ . Etant donné le couple  $(n, e)$  avec  $ed \equiv 1 \pmod{\phi(n)}$ , on peut retrouver efficacement  $d$ .

**Preuve :** On pose  $ed - k\phi(n) = 1$ . D'où  $\frac{e}{\phi(n)} - \frac{k}{d} = \frac{1}{d\phi(n)}$ . On approche  $\phi(n)$  par  $n$  et en utilisant le fait  $d < 1/3\sqrt[4]{n}$  on a :

$$\left| \frac{e}{n} - \frac{k}{d} \right| < \frac{1}{2d^2}$$

En passant par un dev en fractions continues à la bonne précision on retrouve  $k/d$ .  $\square$   
RSA est pas indistinguable. (exponentiation binaire est rapide)



## 4.2 Probleme de log discret

Securité dépend du groupe dans lequel on travaille : Si on prend  $G = (\mathbb{Z}/p\mathbb{Z}, +)$  et  $h = gx \bmod p$  alors  $x = hg^{-1}$ , une étape.

**Definition 4.2.1.** Problème de Diffie-Hellman(DHP): Etant donnés  $g, g^a, g^b$  peut-on trouver  $g^{ab}$ .

**Definition 4.2.2.** Signature d'El Gamal :  $k$  doit être secret et d'usage unique.

- $k$  doit être secret : a faire
- $k$  doit être d'usage unique : pareil

## **5 IGC(infrastucture de gestion de clefs)**

Gère les distributions de certificats.

### **5.1 Création d'un certificat num**

2 modes:

- Décentralisé: l'utilisateur crée son bi-clef
- Centralisé: L'autorité de confiance crée le bi-clef

### **5.2 Révocation d'un certificat num**

- Fin de limite de validité
- Révoque avant la date limite(par exemple si oubli du mot de passe)

### **5.3 Types de certificats**

- Signature
- Chiffrement et signature

### **5.4 Reste**

L'IGC est composée de 4 entités obligatoires:

- Autorité de confiance: signe les certifs
- Autorité d'enregistrement:s'assure de l'identité du demandeur de certifs.
- Autorité de dépôt: stocke les certifs et liste de révocations
- Entité finale: celle qui demande le certif

Optionnellement: Autorité de sequestre, conserve les clefs privées

## 6 Attaque sur log discret

Dans un groupe cyclique  $G = \langle g \rangle$ . On cherche étant donnés  $h := g^x$  à trouver  $x$ .

**Théorème 6.0.1** (Shoup, 1997). *Dans un groupe générique d'ordre  $p$  premier, le calcul d'un log discret est au minimum en  $\mathcal{O}(\sqrt{p})$ .*

### 6.1 Baby step, Giant step!

**Idée:** Déterminer  $s$  et  $i$  tq  $x = st + i$  où  $t$  est choisi et  $0 \leq i < t$ , i.e.

$$h * (g^{-t})^s = g^i$$

**Pré-calcul:**( $\mathcal{O}(t)$  étapes) Calcul de tous les  $g^i$  pour  $0 \leq i \leq t - 1$  et  $g^{-t}$ . (plus table de hashage des  $g^i$ )

**Calcul:**( $\mathcal{O}(p/t)$  étapes) Pour  $s$  allant de 0 à  $n/t$ , on calcule  $h * (g^{-t})^s$  et on teste si  $\exists i$  tq  $h * (g^{-t})^s = g^i$ .  $\square$

On prend  $t = \sqrt{p}$  et on a la borne de Shoup.

### 6.2 Algorithme Rho-Pollard

**Idée:** Balade aléatoire dans  $G$  en attendant une collision (d'ou la lettre  $\rho$ ).

Il nous faut une fonction pseudo-aléatoire tq si  $\alpha_i$  et  $\beta_i$  sont tq

$$g_i = g^{\alpha_i} h^{\beta_i}$$

On puisse déterminer  $\alpha_{i+1}, \beta_{i+1}$  tq  $F(g_i) = g_{i+1} = g^{\alpha_{i+1}} h^{\beta_{i+1}}$ . Si on a une collision i.e.  $g_i = g_j$  alors

$$g^{\alpha_i + x\beta_i} = g^{\alpha_j + x\beta_j}$$

D'ou  $\alpha_i + x\beta_i \equiv \alpha_j + x\beta_j \pmod{n = |g|}$ . Donc si  $\text{pgcd}((\beta_i - \beta_j), n) = 1$  alors

$$x \equiv (\alpha_j - \alpha_i)(\beta_i - \beta_j)^{-1} \pmod{n}$$

**Algorithme:** On calcule la suite  $(\alpha_i, \beta_i)$  avec  $F$  et a chaque étape on regarde si y'a une collision.

**Complexité:** Petite digression sur le

Paradoxe des Anniversaires.

On regarde un tirage aléatoire uniforme sur  $n$  boules avec remise. On cherche  $P(n, k)$  la probabilité qu'après  $k$ -tirages on ait tiré au moins deux fois une même boule: On regarde

$$1 - P(n, k) = \frac{n(n-1) \dots (n-(k-1))}{n^k} = \prod_{i=1}^{k-1} 1 - i/n$$

Or pour tout  $x$  réel,  $1 + x \leq e^x$ . D'où

$$1 - P(n, k) \leq \prod e^{-i/n} = e^{\sum(-i/n)} = e^{k(k-1)/2n}$$

On cherche quand  $1 - P(n, k) \geq 1/2$ :

$$e^{-(k(k-1))/2n} \leq 1/2$$

Donne  $k(k-1)/2n \geq \ln(2)$ , d'où  $k \geq \sqrt{n} \sqrt{2 \ln(2)}$ . Donc si on prend ce  $k$  on a

$$P(n, k) \geq 1/2$$

(wow!).

**Retour à la complexité:**  $\mathcal{O}(\sqrt{n})$  en temps mais  $\mathcal{O}(\sqrt{n})$ -mémoire.. Mais il existe une version sans mémoire de même complexité!

## 6.3 Réduction de Pohlig-Hellman

**Idée:** Au lieu de calculer un unique log discret dans un grand groupe on calcule plusieurs logs discrets dans des groupes plus petits. On suppose que  $n = \prod p_i^{\alpha_i}$  est composé  $\rightarrow$  lemme chinois. Plusieurs paramètres:

On pose  $n_i = \frac{n}{p_i^{\alpha_i}}$ ,  $g_i = g^{n_i}$  alors:

$$|g_i| = p_i^{\alpha_i}$$

$$\text{Puis } h_i = h^{n_i} = (g^x)^{n_i} = g_i^x$$

Et  $x$  est déterminé mod  $p_i^{\alpha_i}$ .

**Algorithme mod  $p^\alpha$ :** On pose  $x = \sum_{i=0}^{\alpha-1} x_i p^i$  et on le fait de proche en proche,  $h^{p^{\alpha-1}} = (g^x)^{p^{\alpha-1}}$ . Alors

Déterminer  $x_0$  c'est déterminer le log discret de  $h^{p^{\alpha-1}} \pmod p$ .

Maintenant étant donnés  $x_0, \dots, x_{e-1}$  on calcule  $x_e$ , on fait comme pour shanks tonelli, i.e.

$$h_{e-1}^{p^{\alpha-(e+1)}} = (g^{p^{\alpha-1}})^{x_e}$$

**Complexité totale (selon Shoup):**  $\mathcal{O}(r \log)$

## 7 Tests de primalité

Plusieurs types:

1. test de composition
2. test de primalité
3. (Cribles)

On considère tjr un entier impair (obvious). Et le test naïf ou on teste tout les entiers plus petits:  $\mathcal{O}(\sqrt{n} \log^2(n))$

### 7.1 Tests probabilistes

**Test de Fermat:** On test si  $a^{n-1} \equiv 1 \pmod n$  pour pleins d' $a$ .

**Definition 7.1.1.** On prend  $b \in (\mathbb{Z}/n\mathbb{Z})^\times$ . Si  $b^{n-1} \equiv 1 \pmod n$  alors  $n$  est dit pseudo-premier en base  $b$ .

**Proposition 7.1.2.** eq

- (i)  $n$  est p.p en base  $b$  ssi  $|b| \mid n-1$
- (ii) Si  $n$  est p.p en base  $b_1$  et  $b_2$  alors  $n$  est pp en base  $b_1 b_2$
- (iii) Si  $\exists c \in (\mathbb{Z}/n\mathbb{Z})^\times$  tq  $c^{n-1} \not\equiv 1 \pmod n$ , alors il ya au moins autant de bases pour lesquelles  $n$  n'est pas pp que de bases pour lesquelles il l'es.

**Démo:** (iii) Etant donné  $B = \{b \mid b^{n-1} \equiv 1 \pmod n\}$ . Supposons qu'on a ce  $c$ , on pose  $C = cB$ . On a  $\#C = \#B$ . Mtn si  $cb \in B$  alors par (ii)  $c \in B$ . Pas possible.  $\square$

**Algo:** On itère  $k$  fois

- on tire aléatoirement  $b$ .
- On teste si  $b$  est premier à  $n$ .
- Si oui on teste si  $b^{n-1} \equiv 1 \pmod{n}$ .

Non  $n$  est composé.

Oui On change de  $b$ .

Si après  $k$  tirages, l'algo n'indique que  $n$  est pas composé alors il est premier avec une proba  $\geq 1 - \frac{1}{2^k}$  à la condition qu'il existe un  $c$  vérifiant (iii).

**Complexité:**  $\mathcal{O}(k \log^3 n)$ .

**Definition 7.1.3.** Les entiers de Carmichael sont ceux qui vérifient Fermat en étant composé impairs.

**Proposition 7.1.4.** (eq)

- (i) Si  $n$  contient un facteur carré alors  $n$  n'est pas de Carmichael.
- (ii) Si  $n$  est sans facteurs carré,  $n$  est de Carmichael ssi  $\forall p \mid n$

$$(p-1) \mid (n-1)$$

**Proof:** (i) Avec  $\langle g \rangle = (\mathbb{Z}/p^2\mathbb{Z})^\times$  et

- $b \equiv g \pmod{p^r}$
- $b \equiv 1 \pmod{n/p^r}$

D'où  $p(p-1) \mid n-1$  et comme  $n-1 \equiv 1 \pmod{p}$ . □

**Théorème 7.1.5** (Alford, Pomerance et Granville). *Il y'a une infinité de nombres de Carmichael. (par exemple  $561 = 3 * 11 * 17$ )*

**Test de Solovay-Strassen:**

**Proposition 7.1.6.** Si  $\forall b \in (\mathbb{Z}/n\mathbb{Z})^\times$ , on a

$$\left(\frac{b}{n}\right) = b^{\frac{n-1}{2}}$$

Alors  $n$  est premier.

**Lemme 7.1.7.** Si  $n$  est composé impair, alors pour au moins la moitié des  $b \in (\mathbb{Z}/n\mathbb{Z})^\times$ ,

$$\left(\frac{b}{n}\right) \neq b^{\frac{n-1}{2}}$$

**Démo:** Même argument que pour Fermat. Y faut juste avoir un  $c$  comme dans l'énoncé. On le cherche

Cas ou  $p^2 \mid n$ : On pose  $c = 1 + \frac{n}{p}$ . On a  $\left(\frac{c}{n}\right) = \left(\frac{c}{n}\right)^2 \times \left(\frac{\frac{c}{n}}{p^2}\right) = 1.1 = 1$ . Puis  $c^j = (1 + \frac{n}{p})^j = 1 + j \cdot \frac{n}{p} \pmod{n}$  d'où  $c^j = 1 \pmod{n}$  ssi  $j \equiv 0 \pmod{p}$ . Maintenant  $\frac{n-1}{2} \equiv 0 \pmod{p}$  ssi  $n-1 \equiv 0 \pmod{p}$  qui est pas possible.

Cas ou  $n = \prod p_i$ : On prend  $c_0$  un résidu non quadratique pour  $p_0$  et  $c_1 \equiv 1 \pmod{n/p_0}$ . On prend  $c$  tel que  $c \equiv c_0 \pmod{p_0}$  et  $c \equiv c_1 \pmod{\frac{n}{p_0}}$ . Maintenant en utilisant  $\left(\frac{c}{n}\right) = \left(\frac{c_0}{p_0}\right) \times \left(\frac{c_1}{n/p_0}\right)$  on prouve que le  $c$  est bien comme on veut.

**Algo de Solovay-Strassen:** On itère  $k$  fois.

- On tire aléatoirement  $b \in [2, n-1]$
- On teste si  $b$  et  $n$  sont premiers entre eux

Non  $n$  est composé

Oui On teste si  $\left(\frac{b}{n}\right) = b^{\frac{n-1}{2}} \pmod{n}$

Non  $n$  est composé.

Oui On change de  $b$ .

Après  $k$  passages avec succès,  $n$  est premier avec une proba  $\geq 1 - \frac{1}{2^k}$ .

**Complexité:**  $\mathcal{O}(k \log^3 n)$ .

**Test de Miller-Rabin:** l'idée est que si  $n \neq p^\alpha$  est composé alors  $n$  a plus de 2 racines carrées.

**Definition 7.1.8.** On pose  $s = v_2(n - 1)$  et  $t = n/2^s$ . Soit  $b \in (\mathbb{Z}/n\mathbb{Z})^\times$ .

Si  $b^t \equiv 1 \pmod n$  ou  $\exists 0 \leq r < n - 1$  tq  $(b^t)^{2^r} \equiv -1 \pmod n$  alors on dit que  $n$  est pp fort en base  $b$ .

**Lemme 7.1.9** ( $n$  un entier impair composé). *Pour au plus  $1/4$  des élt de  $(\mathbb{Z}/n\mathbb{Z})^\times$ ,  $n$  est pp fort.*

**Algo de Miller-Rabin:** On itère  $k$  fois,

- On tire aléatoirement  $n \in [2, n - 2]$ .
- On test si  $b$  est premier à  $n$ .

Non  $n$  est composé.

Oui On regarde si  $b^t \equiv \pm 1 \pmod n$ .

Si oui: On change de  $b$

Sinon: On calcule les  $(b^t)^{2^r} \pmod n$

Un  $-1$ : On change de  $n$

Pas de  $-1$ :  $n$  est composé

Après  $k$  tirages avec succès,  $n$  est premier avec une proba  $\geq 1 - \frac{1}{4^k}$ .

**Complexité:**  $\mathcal{O}(k \log^3 n)$ .

## 7.2 Test de primalité assurée

**Test  $n - 1$  de Pocklington-Lehman:** On considère  $n$  impair.

**Proposition 7.2.1.** Soit  $p \mid n - 1$ . Si on a un  $a \in \mathbb{Z}$  tq:

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

et  $(a^{\frac{n-1}{p}} - 1)$  est premier à  $n - 1$ . Alors pour tout diviseur  $d$  de  $n$ ,

$$d \equiv 1 \pmod{p^\alpha}$$

où  $p^\alpha \mid n - 1$ .



**Démo:** Soit  $d \mid n$  premier et  $a$  comme dans l'énoncé. On a  $a^{n-1} \equiv 1 \pmod{f}$ . Et  $a^{\frac{n-1}{p}} \not\equiv 1 \pmod{d}$ . On note  $e_d = |a| \pmod{d}$ . Donc  $e_d \mid n-1$  et  $e_d \nmid (n-1)/p$ . Si  $p^\alpha \mid n-1$  alors  $p^{\alpha} \mid e_d$ . En plus  $a^{d-1} \equiv 1 \pmod{d}$  d'où  $e_d \mid d-1$ . Puis  $p^\alpha \mid e_d$  et  $d \equiv 1 \pmod{p^\alpha}$ . qed

**Corollaire 7.2.2.** *Soit  $n$  un entier tq  $n-1 = f.u$  avec  $f \geq \sqrt{n}$ ,  $f$  complètement factorisé et  $f$  premier à  $u$ . Supposons que  $\forall p \mid f, \exists a$  comme dans la prop précédente. Alors  $n$  est premier et inversement.*

**Démo:** Supposons que  $\forall p \mid f$ , il existe un  $a$  comme dans l'énoncé. D'après la prop,  $\forall p \mid f, \forall d \mid n$  on a  $d \equiv 1 \pmod{p^\alpha}$ . D'où par le lemme chinois  $d \equiv 1 \pmod{f}$ . Supposons que  $d = 1 + kf \neq 1$ . Alors  $n$  n'a pas de diviseur  $\neq 1 \leq \sqrt{n}$  donc  $n$  est premier. Inversement, si  $n$  est premier, on prend un générateur  $a$  de  $\mathbb{F}_n^*$ .

### 7.3 L'algorithme AKS

**AKS pour Aggraval, Kayal et Sapena. Idée:**  $n$  est premier ssi  $(X+a)^n = X^n + a \pmod{n}$ . (On va regarder modulo un polynome)

**Algo AKS:** On évince le cas ou  $n = m^b$  et  $b \neq 1$ .

- On pose  $r$  minimal tel que  $\mathcal{O}_r(n) \geq 4 \log^2 n$  avec  $(\mathcal{O}_r(n)$  l'ordre de  $n$  dans  $(\mathbb{Z}/r\mathbb{Z})^\times$ )
- Si  $\exists 1 \leq a \leq r$  tq  $1 < a \wedge n < n$ . Alors  $n$  est composé.
- Si  $n \leq r$ ,  $n$  est premier.
- Pour  $1 \leq a \leq [2\sqrt{\phi(r)} \log n]$ , on teste si  $(X+a)^n \neq X^n + a \pmod{(n, X^r - 1)} \implies n$  est composé. Sinon  $n$  est premier.

**Complexité:**  $\mathcal{O}(\log^{10,5}(n))$ .

## 8 factorisation d'entiers

La méthode naive est en  $\mathcal{O}(\sqrt{n} \log^2 n)$  comme d'hab.

## 8.1 Algo de Rho-Pollard

**Idée:** On fait une balade aléatoire dans  $\mathbb{Z}/n\mathbb{Z}$  (on calcule une suite  $(x_i)$ ) et pour tout  $x_j$  calcule ou teste si pour  $i < n$   $(x_j - x_i) \wedge n \neq 1, n$ . On obtient un  $r$  tel que  $x_j \equiv x_i \pmod{r}$ . (On prend  $x_j - x_i$  Et pas  $x_j$  pour avoir un paradoxe des anniversaires (bcp plus de poss en ajoutant  $x_i$ )).

**Proposition 8.1.1** (Rappel:). Une suite  $f(x_i) = x_{i+1}$  de  $S \rightarrow S$  ( $S$  fini), contient une collision en  $l$  étapes ou  $l = 1 + \lceil \sqrt{2\lambda r} \rceil$ . avec proba  $\leq e^\lambda$ .

**Sans mémoire:** Etant donné  $(i_0, j_0)$  deux premiers indices tq  $(x_{j_0} - x_{i_0}) \wedge n = r \neq 1, n$ . Soit  $j \in \mathbb{N}$ , on pose  $h = \lceil \log j \rceil$  (ie  $2^h \leq j < 2^{h+1}$ ). On pose  $i = 2^n - 1$

On calcule  $(x_k)$  tq  $f(x_k) = x_{k+1}$  et on teste si  $(x_j - x_i) \wedge n \neq 1, n$  avec  $i$  déf comme précédemment. Si  $x_{j_0} \equiv x_{i_0} \pmod{r}$  alors

$$x_{j_0+m} \equiv f^m(x_{j_0}) \equiv f^m(x_{i_0}) \equiv x_{i_0+m} \pmod{r}$$

Donc si  $j - i = j_0 - i_0$  on a  $x_j \equiv x_i \pmod{r}$ . On pose  $h_0 = \lceil \log j_0 \rceil$  et  $i = 2^{h_0+1} - 1$  et  $j = i + j_0 - i_0$ . Par déf de  $j$ ,  $x_j \equiv x_i \pmod{r}$ :

$$j = 2^{n_0+1} - 1 + j_0 - i_0$$

$$j_0 - i_0 \geq 1 \text{ donc } j \geq 2^{k_0+1}$$

$$j_0 - i_0 \leq j_0 < 2^{k_0+1}$$

Donc  $j < 2^{k_0+1} - 1 + 2^{k_0+1} = 2^{k_0+2} - 1$  et  $2^{k_0+1} \leq j < 2^{k_0+2}$  ( $\lceil \log j \rceil = h_0 + 1$ )

La fonction  $f$  est arbitraire faut bien la choisir

## 8.2 Méthode de Fermat et améliorations

**Proposition 8.2.1.** Il existe une bijection entre les couples  $(a, b)$  tq  $x = ab$  et les couples  $(t, s)$  tq  $x = t^2 - s^2$ .

**Idée:** On pose  $t = \lceil \sqrt{n} \rceil + 1$  et on teste si  $t^2 - n$  est un carré. Sinon on passe à  $t + 1, t + 2, \dots$

**Méthode de Dixon**

**Idée:** Au lieu de travailler dans  $\mathbb{Z}$ , on travaille dans  $\mathbb{Z}/n\mathbb{Z}$ . On cherche  $t^2 \equiv s^2 \pmod{n}$ . Si on a un couple, on calcule  $(t - s) \wedge n$ . Comme c'est dur de trouver ces couples on les construit.

**Méthode:**

On décide d'une base de facteurs  $B \subseteq \mathbb{Z}$  pour laquelle on va chercher des  $t$  tq  $t^2 \bmod n$  soit  $B$ -friables i.e.  $t^2 = \prod_{b_i \in B} b_i^{\alpha_i} \bmod n$ .

- On choisit  $A \in \mathbb{Z}$  ou on va tester si  $t \in A$  est tq  $t^2 \bmod n$  est  $B$ -friable.
- On teste si  $\forall t \in A$ ,  $t^2 \bmod n$  est  $B$ -friable. On obtient ainsi  $A' \subseteq A$  où pour tout  $t \in A'$ ,  $t^2 \bmod n$  est friable.
- Pour tout  $t_j \in A'$ , on associe

$$\epsilon_j = (\alpha_{1j} \bmod 2, \alpha_{2j} \bmod 2, \dots, \alpha_{bj} \bmod 2)$$

- A partir de ces  $\epsilon_j$  on cherche  $\sum_j \epsilon_j = 0$

$$\prod_j t_j^2 = \prod_{b_i \in B} \prod b_i^{\alpha_{i,j}} = \prod_{b_i \in B} b_i^{\sum \alpha_{i,j'}} \bmod n$$

**Crible quadratique:**

- $B = \{p | p < P \text{ et } \left(\frac{n}{p}\right) = 1\} \cup \{2\}$ , on pose  $\#B = h$ .
- $A = [\sqrt{n} + 1, \sqrt{n} + K]$  où  $P < K < P^2$ .

Dans ce crible on va tester si  $t^2 - n$  est  $B$ -friable avec  $t \in A$ . Si  $t^2 - n \bmod [n] = p_i^{\alpha_i} \prod_{i \neq 1} p_i^{\alpha_i}$  avec  $\alpha_i \geq 1$ . Alors  $t^2 - n \equiv 0 \bmod [p_i^{\alpha_i}]$  donc  $t^2 \equiv n \bmod [p_i^{\alpha_i}]$ . Si  $t$  est tq  $t^2 \equiv n \bmod [p_i]$  alors  $(t + kp_i)^2 \equiv n \bmod [p_i]$ .

**suite:** Soit  $2 \neq p \in B$ . Par def  $\left(\frac{n}{p}\right) = 1$ . On calcule  $x_0$  tq  $x_0^2 \equiv n \bmod [p]$ . On cherche  $x_1$  tq  $(x_0 + x_1 p)^2 \equiv n \bmod [p^2]$  puis

$$x_0^2 + 2x_0x_1p + x_1^2p^2 \equiv n \bmod [p^2]$$

$$p(k_0 + 2x_0x_1) \equiv 0 \bmod [p^2]$$

donc  $k_0 + 2x_0x_1 \equiv 0 \bmod [p]$ . On relève la racine tant qu'il existe  $t \in A$  tq  $t^2 \equiv n \bmod [p^\alpha]$

**Complexité:**  $\mathcal{O}(e^{(1+\epsilon)\sqrt{\log n \log \log n}})$

**8.3**

TEUP

$\beta$