

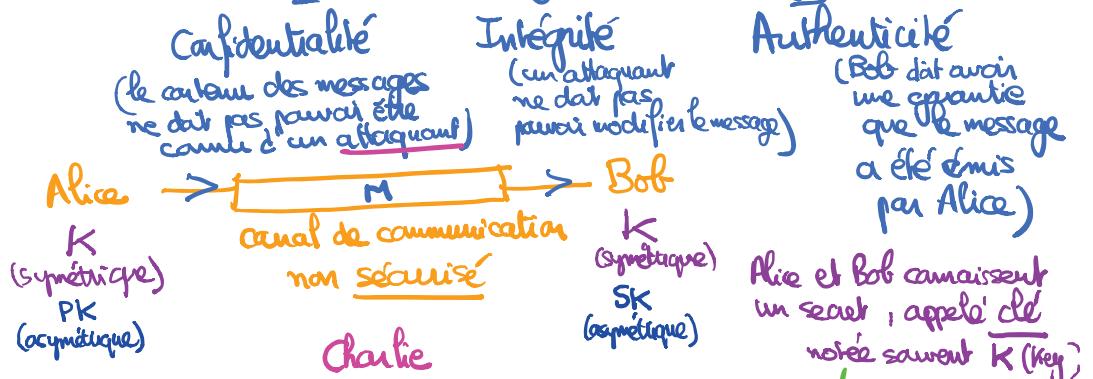
Complexité algorithmique et cryptographie

théorie de la complexité

= temps de calcul
des algorithmes
permettant de résoudre
un problème

+ mémoire

Mise en place d'algorithmes
pour résoudre les questions
liées à la sécurité informatique

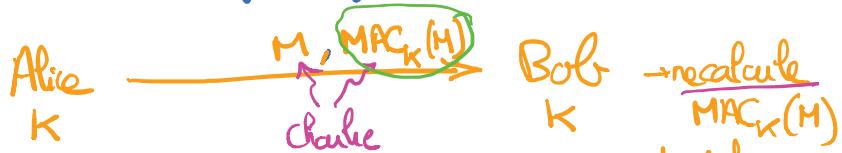


On suppose que le canal de communication n'est pas physiquement sécurisé.

→ la cryptographie est l'ensemble des moyens ~~pour~~ (de nature algorithmique) qui permettent de compenser le fait que le canal ne soit pas physiquement sécurisé.

Transformations appliquées aux messages

- Confidentialité → algorithmes de chiffrement
 - symétriques
 - asymétriques
- Intégrité
 - symétrique → MAC (Message Authentication Code)
 - asymétrique → signature
- Authenticité
 - symétrique : ? pas possible
 - asymétrique → signature



Question : ce mécanisme de MAC garantit-il l'authenticité ?

et vérifie que la valeur reçue est identique au résultat de ce calcul

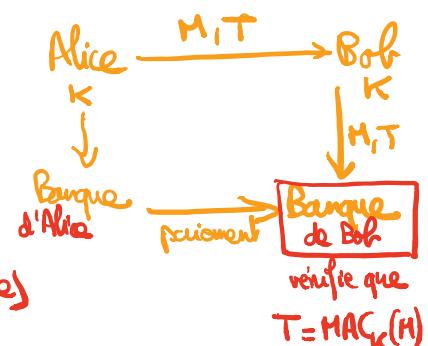
Réponse : 1) Bob obtient une garantie que l'émetteur du message connaît la clé K
 ↳ à priori l'émetteur est donc bien Alice

2) Mais Bob ne peut pas convaincre un tiers qu'Alice est l'émetteur du message

Ex: $M = \text{"Moï Alice je paie 10000€ à Bob"}$

$$T = \text{MAC}_K(M)$$

Authenticité → preuve de la connaissance d'un secret
 → non répudiation (pas possible en symétrique)



	Confidentialité	Intégrité	Authenticité
symétrique	chiffrement symétrique One-time pad DES, AES	MAC	Authenticité
asymétrique	chiffrement asymétrique ElGamal 1, RSA		signature (électronique) ElGamal 2, RSA, Zero-knowledge

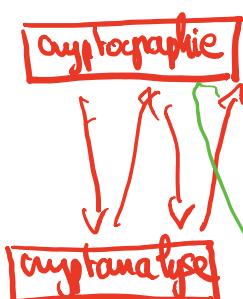
1976 : standardisation du DES
 finalisation de la crypto asymétrique
 (W. Diffie, M. Hellman : "New Directions in Cryptography")

En cryptographie particulièrement, utilisation d'objets algébriques :

- théorie des nombres → factorisation des entiers (arithmétique discrète)
- courbes elliptiques
- codes correcteurs
- systèmes polynomiaux (géométrie algébrique)
- réseaux euclidiens

Résumé : • On construit des algorithmes cryptographiques

- à partir d'objets de nature algébrique
- pour garantir des propriétés de sécurité (confidentialité, intégrité, authenticité) + anonymat, ...



• Etude des attaques : ok: cryptanalyse différentielle linéaire réduction de réseaux ...

• Preuve de sécurité

Dans certains cas, on peut montrer que, par exemple, l'algorithme de chiffrement E garantit effectivement la confidentialité

si et seulement si un certain problème algébrique est calculièrement difficile

“dépasse les capacités de calcul de l'attaquant”

Complexité algébrique et cryptographie

Ch 1 : Problèmes difficiles de théorie des nombres

I] Complexité et cryptographie

RSA \rightarrow factorisation

ElGamal \rightarrow logarithme discret

Theorie de la complexité

① Introduction

Idée: Mesure de la "difficulté" algorithmique d'un problème

Problème de décision: collection d'instances qui sont des ensembles de données qui admettent exactement une des 2 réponses ("oui" ou "non") à une certaine question.

Exemple 1: Pb SAT (Satisfaisabilité des fonctions booléennes)

Instance: Une fonction à variables booléennes

$$(x_1, \dots, x_n) \mapsto F(x_1, \dots, x_n)$$

où $F(x_1, \dots, x_n)$ est une expression construite avec les x_i et des connecteurs logiques

\vee, \wedge, \neg
 $\downarrow \quad \downarrow \quad \downarrow$
ou et non

ex: $F(x_1, x_2, x_3, x_4)$
 $= (\neg(\overline{x}_1 \wedge (\neg x_3)) \vee$
 $(x_4 \wedge x_2 \wedge (\neg x_1)))$

Question: Existe-t-il une valeur du n-uplet (x_1, \dots, x_n) rendant vraie $F(x_1, \dots, x_n)$?

Algorithme possible : faire une recherche exhaustive sur (x_1, \dots, x_n) (en espérant trouver un n-uplet tel que $F(x_1, \dots, x_n)$ soit vrai)

↪ complexité = $\boxed{\mathcal{O}(2^n)}$ (au pire)

Exemple 2 : FBQ (Formules Booléennes Quantifiées)

Instance : une famille booléenne avec quantifications du type

$$\forall x_i, \exists x_j, \dots, F(x_1, \dots, x_n)$$

(où F est une fonction à variables booléennes)

Question : cette famille est-elle vraie ?

ex : $\forall x_1, \exists x_2, \forall x_3, \forall x_4,$
 $(x_1 \wedge (x_2 \vee x_3)) \wedge (\neg x_4 \vee x_2)$

Algorithme possible : recherche exhaustive

↪ complexité = $\boxed{\mathcal{O}(2^n)}$

- Remarque :
- dans l'exemple 1 (SAT), si on a trouvé que la réponse est "oui", il est facile de transmettre à un tiers un "certificat" qui lui permet de vérifier "très rapidement" que la réponse est "oui"
 - dans l'exemple 2 (FBQ), il n'y a pas cette possibilité

Exemple 3 : Équations diophantiennes (10^{ème} problème de Hilbert)

Instance : Une équation polynomiale à plusieurs inconnues et à coefficients entiers

$$\text{ex: } x^2 + 3y^4 + 2z^5 = 7$$

Question : L'équation admet-elle une solution en nombres entiers ?

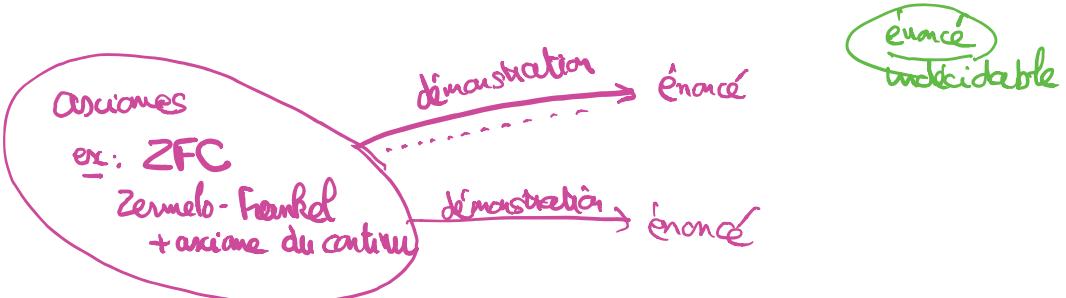
Remarque : Hilbert a énoncé (en 1900) 23 problèmes

En 1971, Matiasentch a démontré qu'il ne peut pas exister d'algorithme pour le 10^e pb de Hilbert

Idee de Matiasentch : / tout énoncé mathématique peut se réécrire comme un pb de décision sur les équations diophantiennes

Gödel (1936) : théorèmes d'incomplétude

↓
1er théorème : pour un système d'axiomes qui contient l'arithmétique de Peano, il existe des énoncés qui ne peuvent pas être démontrés, ni l'énoncé contenant (énoncé indécidable dans ce système d'axiomes)



② Calculabilité au sens de Turing

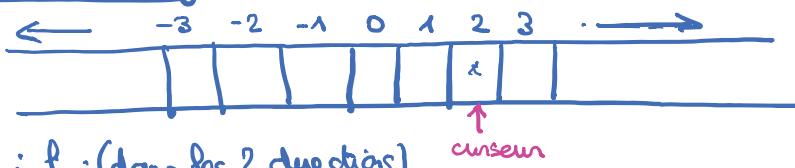
Enigma (algo allemand pendant la 2^{nde} guerre mondiale)

→ cryptanalyse

→ conception de machines dédiées à la cryptanalyse
(Imitation Game)

→ **Colossus** (Bletchley Park)

Machine de Turing



Ruban infini (dans les 2 directions) cursor

Chaque case contient un symbole (\in alphabet fini Σ , que l'on peut supposer égal à {0,1}, ou b (= blanc))

Le numan est lu, case par case, par un cuseur

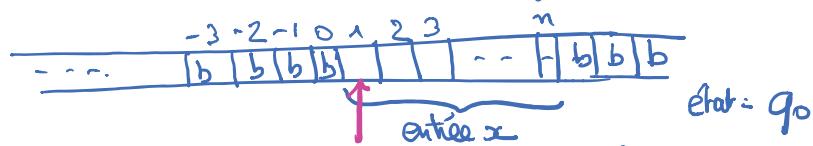
La machine est, à chaque instant, dans un état q_i
($\in Q$ = ensemble fini des états de la machine)

Def : une opération élémentaire est entièrement déterminée par le symbole lu et par l'état q_i dans lequel se trouve la machine :

- 1) le curseur remplace le symbole lu par un élément de $\Sigma \cup \{b\}$ (éventuellement le même)
- 2) le curseur se déplace d'au plus une case vers la gauche ou vers la droite.
- 3) la machine passe de l'état q_i à l'état $q_j \in Q$

$$M: (\sum \cup \{b\}) \times Q \rightarrow (\sum \cup \{b\} \times \{-1, 0, 1\} \times Q)$$

Calcul déterministe: suite d'opérations où la machine commence par être dans un état initial q_0 , son curseur étant sur la case 1, une entrée x étant placée dans les cases 1, ..., n (x = suite de n éléments de Σ) les autres cases contenant le symbole b



le calcul se termine lorsque la machine aboutit à l'état terminal $q_f \in Q$

La sous, si il y en a une, est le contenu du ruban dans l'état terminal

Terminologie: l'ensemble des suites finies d'éléments de l'alphabet Σ est noté Σ^*

Un élément de Σ^* = mot

Une partie de Σ^* = langage

Une fonction $f: \Sigma^* \rightarrow \Sigma^*$ est Turing-calculable s'il existe une machine de Turing M qui sur toute entrée x du domaine de définition $\text{Dom}(f)$ de f termine son calcul avec $f(x)$ comme sortie.

Remarque: Début dans les années 1930

- Gödel → nécessité au sens de Herbrand-Gödel
 - Turing → définition algorithmique
 - Church → λ -démonstrabilité
- équivalence entre les 3 définitions prouvée par Church et Kleene en 1936

Théorie de Church-Turing : Tout ce qui est calculable physiquement l'est aussi au sens de Gödel/Turing / Church.

③ Complexité en temps : classes P et NP

Définition : La longueur d'un calcul, sur une entrée $x \in \Sigma^*$, par une machine de Turing M , est le nombre $t_M(x)$ d'opérations élémentaires qui composent le calcul.

Complexité en temps : $T_M : \{ \text{IN} \rightarrow \text{IN} \}$

$$\begin{cases} n \mapsto \max_{\substack{x \in \Sigma^* \\ |x|=n}} t_M(x) \end{cases}$$

Def : Un algorithme déterministe en temps polynomial est (ou plus simplement : "algorithme polynomial") pour f est une machine M qui calcule f et telle qu'il existe un polynôme p tel que :

$$\forall n \in \mathbb{N}, T_M(n) \leq p(n)$$

Classe P : Ensemble des problèmes de décision admettant un algorithme déterministe polynomial

- ex :
 - test de primalité :
 - Fermat : $a^{p-1} \equiv 1 \pmod{p}$
 - (contre-exemple : les nombres de Carmichael sont des entiers n composés tels que $\forall a, a^{n-1} \equiv 1 \pmod{n}$)
 - Sorenson, Shassen
 - Miller-Rabin
 - AKS (2001) : Agrawal-Kayal-Saxena
 - "Prime is in P"

- est-ce que a et b sont premiers entre eux?
 \hookrightarrow le calcul du pgcd est en temps polynomial
 algorithme d'Euclide étendu
- est-ce que $ab = 1 \pmod{n}$?
 \rightarrow polynomial
- est-ce que A est une matrice inversible?
 $\hookrightarrow \det A = \sum_{\sigma \in S_n} (-1)^{\text{sign}(\sigma)} \prod_{i=1}^n a_{i,\sigma(i)}$
 $A = (a_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$ \downarrow
 $n!$ permutations

On peut utiliser le pivot de Gauss pour calculer $\det A$
 complexité : $\boxed{O(n^3)}$ = temps polynomial

- Connectivité d'un graphe
 - Multiplication d'entiers : $\overset{n \text{ bits}}{a} \times \overset{n \text{ bits}}{b}$
- Méthode scolaire
- $\boxed{O(n^2)}$
-

Méthode avec FFT (Transformée de Fourier rapide)
 $\hookrightarrow O(n \ln n \ln \ln n)$

VanderHoeven et al., 2018 $\rightarrow \boxed{O(n \ln n)}$

- Résolution d'un système de n équations à n inconnues
 Gauss $\rightarrow O(n^3) \underset{\text{linaire}}{\approx} 2,7$

\rightarrow Strassen $\rightarrow O(n^{\log_2 7})$
 Meilleure méthode connue $\rightarrow O(n^{2,37\dots})$
 Complexité conjecturée $\rightarrow \boxed{O(n^{2+\varepsilon})}$ ($\forall \varepsilon > 0$)

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} = \begin{pmatrix} a & a'+bc' \\ c & \cdot \end{pmatrix} \xrightarrow[8 \text{ multiplications}]{\quad} a'a' + bc'$$

Strassen a trouvé une méthode en 7 multiplications (1967)

Réalise un système binaire se ramène à des multiplications de matrices

$$(+)(+) \rightarrow 7 \text{ multiplications (par blocs)}$$

$$\text{Gauss : } O(n^3) \quad 3 = \log_2 8$$

$$\text{Strassen } O(n^{2,7}) \quad 2,7 \approx \log_2 7$$

ch 1: Problèmes difficiles de théorie des nombres

I] Complexité et cryptographie

① Introduction

② Calculabilité au sens de Turing

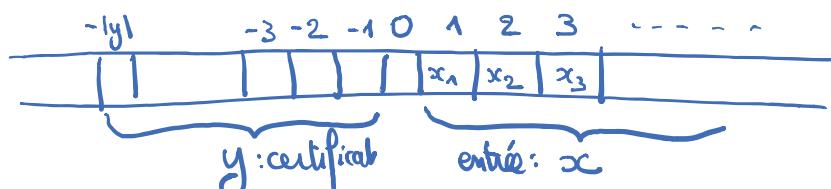
③ Complexité en temps - Classes P et NP

Classe P : $T_M(n) \leq p(n)$
(p : polynôme)

Calcul non déterministe

Def: On dit qu'un pb de décision est calculable par un algorithme non déterministe polynomial s'il existe une machine de Turing M et un polynôme p tels que

- La réponse est oui pour l'entrée x ssi $\exists y \in \Sigma^*$ (certificat) tel que M calcule 1 lorsqu'on met x dans les cases 1 à n , et y dans les cases $-1, \dots, -|y|$
- pour tout x donnant la réponse 1, M calcule 1 en un temps $\leq p(n)$



Exemple : pour le pb SAT

$$\begin{cases} x = \text{famille booléenne} \\ y = \text{valeurs booléennes qui rendent la famille vraie} \\ \text{certificat} \end{cases}$$

Classe NP : Ensemble des problèmes de décision admettant un algo non déterministe polynomial

Remarque : $P \subseteq NP$

④ Problèmes NP-complets

Def. : On dit que le pb de décision P_1 se réduit polynomiallement à P_2 s'il existe $\varphi : \Sigma^* \rightarrow \Sigma^*$ calculable en temps polynomial tel que

$$\begin{bmatrix} \text{La réponse à } P_1 \text{ est oui pour l'entrée } x \\ \Leftrightarrow \text{La réponse à } P_2 \text{ est oui pour l'entrée } \varphi(x) \end{bmatrix}$$

Notation : $P_1 \leq P_2$

Remarque : $(P_1 \leq P_2 \text{ et } P_2 \in P) \Rightarrow P_1 \in P$

Définition : Un problème Π est NP-complet si

$$\forall P \in NP, P \leq \Pi$$

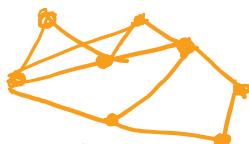
Théorème [Cook, 1971] : SAT est NP-complet

Remarque : Si un pb P vérifie $SAT \leq P$
 alors P est aussi NP-complet
 → méthode pour obtenir de nombreux pb NP-complets

Ex : [Garey, Johnson]

• SAT

• Cycle hamiltonien



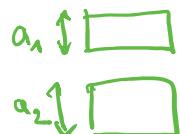
: est-ce pour un graphe, existe-t-il un circuit passant par chaque sommet une fois et une seule

• Pb de la 3-colorabilité d'un graphe

• Sac à dos :

Instance : { Famille d'entiers $(a_i)_{1 \leq i \leq n}$
 Entier C }

Question : existe-t-il (x_1, \dots, x_n) avec
 $\forall i, x_i \in \{0, 1\} / \sum x_i a_i = C$



a _{1,2}
a _{1,2}
a _{1,1}
a ₅
a ₂

• Système d'équations quadratiques sur $K = \mathbb{F}_2$

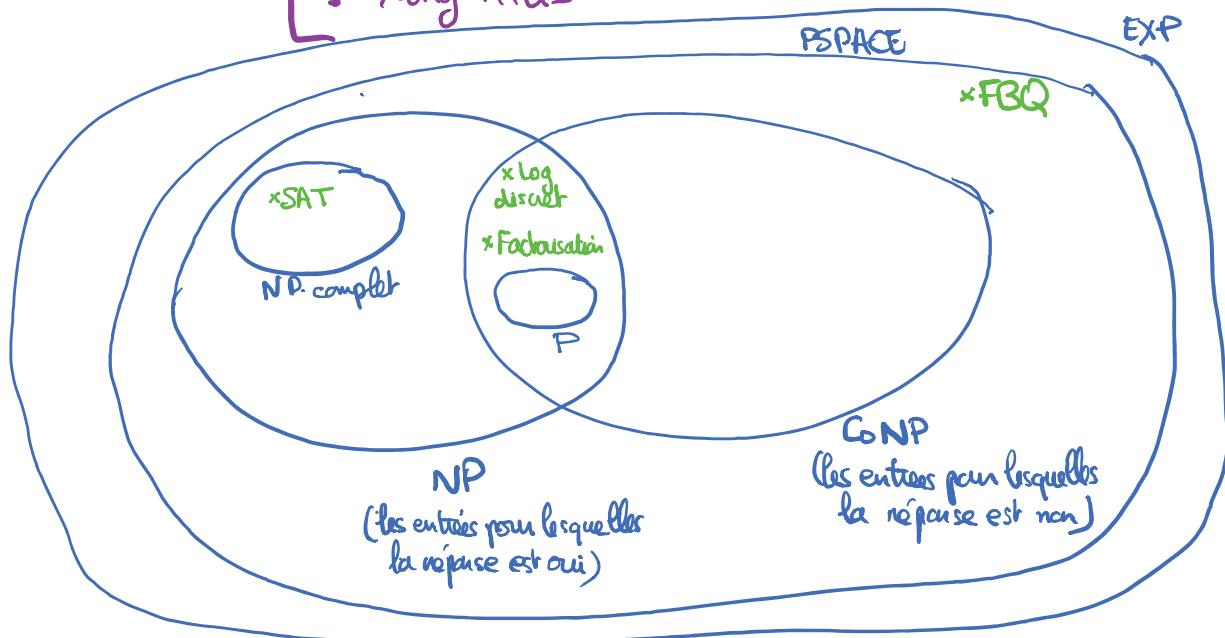
$$\begin{cases} \sum_{1 \leq i, j \leq n} a_{ij} x_i x_j = b_i \\ \sum_{1 \leq i \leq j \leq n} a_{ijn} x_i x_j = b_n \end{cases}$$

→ NP-complet

Conjecture : $P \neq NP$

7 problèmes des millénaires en 2000 (Clay Institute)

- $P \neq NP$
- Conjecture de Poincaré : résolue par Perelman
- Hypothèse de Riemann
- Conjecture de Birch Swinnerton-Dyer
- Conjecture de Hodge
- Équations de Navier-Stokes
- Yang-Mills



III] Factorisation

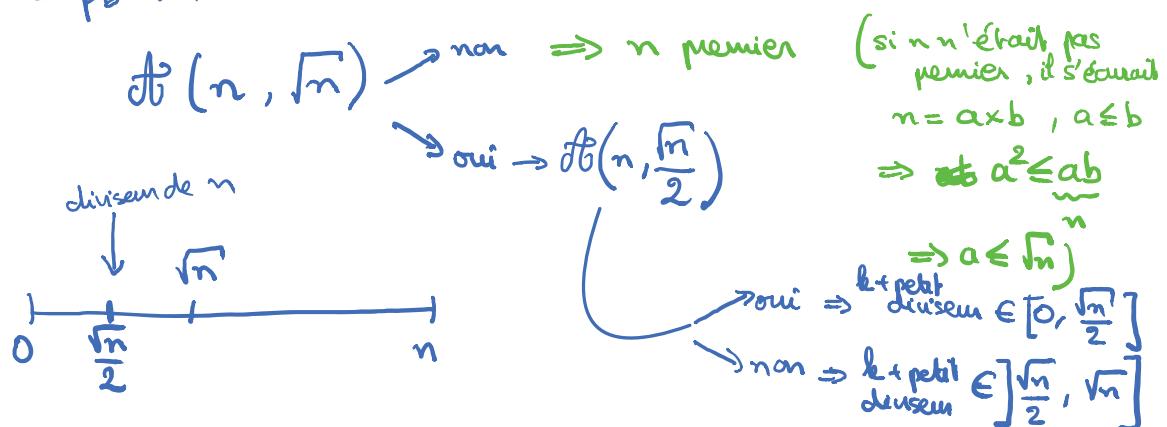
① Complexité

Pb décision : FACTM (pb des facteurs majorés)

Instance : Un entier n , et un nombre $M \leq n$

Question : Existe-t-il un diviseur de n qui est $\leq M$?

- Si on a un algorithme en temps polynomial pour factoriser les entiers, alors on peut résoudre FACTM en temps polynomial.
- Supposons qu'on ait un algorithme \mathcal{B} polynomial pour le pb FACTM. Soit $n \in \mathbb{N}$



Par dichotomie, on trouve ce plus petit diviseur en $O(\ln n)$
appelé à l'algorithme \mathcal{B}
i.e. un nb polynomial de fois

On a trouvé le + petit facteur premier, p_1 , de n ← étape 1

$$\text{On pose } n' = \frac{n}{p_1} \quad \text{étape 2}$$

et on recommence avec n' → $O(\ln n)$ appels à \mathcal{B}
pour trouver le + petit facteur
puis avec $n'' = \frac{n'}{p_2}$ ← étape 3

$$\text{avec } n = \prod_{i=1}^k p_i^{\alpha_i} \geq \prod_{i=1}^k 2^{\alpha_i} = 2^{\sum_{i=1}^k \alpha_i}$$

$$\Rightarrow \ln n \geq \left(\sum_{i=1}^k \alpha_i \right) \ln 2$$

nombre d'étapes dans l'algo

$$\Rightarrow \sum_{i=1}^k \alpha_i \leq \left\lceil \frac{\ln n}{\ln 2} \right\rceil$$

⇒ Complexité de l'algo
 $\rightarrow O((\ln n)^2)$ appels à \mathcal{B}

Remarque sur P=NP : Astuce attribuée à Levin

pour résoudre SAT, on peut considérer toutes les machines de Turing prenant en entrée une instance de SAT

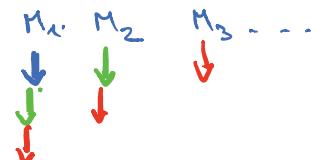
M_1, M_2, M_3, \dots

Pour une instance donnée,

- on exécute la 1^{ère} opération sur M_1

- puis on exécute la 2^e opération sur M_1 et la 1^{ère} opération sur M_2

...



② Idée de Fermat

. Ideas : essayer de diviser par tous les entiers $\leq n$

$\rightarrow O(n)$ avec $n = e^{\ln n}$

algorithme exponentiel

taille de l'entrée

. diviser par les entiers $\leq \sqrt{n}$

$O(\sqrt{n})$ avec $\sqrt{n} = e^{\frac{1}{2} \ln n}$

. diviser par les nombres premiers $\leq \sqrt{n}$
il y en a $\approx \frac{1}{\ln \sqrt{n}} = e^{-\frac{1}{2} \ln n - \ln \ln \sqrt{n}}$

Théorème des Nombres Premiers (Hadamard, de la Vallée Poussin, 1896)

Soit $\pi(x) = \text{Card} \{ p / p \text{ premier}, 1 \leq p \leq x \}$

On a $\pi(x) \underset{x \rightarrow +\infty}{\sim} \frac{x}{\ln x}$

Exemple : Pour les entiers de 1024 bits, la densité des nombres

$$\text{premiers est } \approx \frac{1}{\ln 2^{1024}} \approx \frac{1}{300}$$

- Idee (Fermat): pour factoriser n , il suffit de trouver x et y tels que $x^2 = y^2 \pmod{n}$ avec $x \neq \pm y \pmod{n}$
- $\boxed{\operatorname{pgcd}(n, x-y)}$
- n divise $(x-y)(x+y)$ mais ni $(x-y)$ ni $(x+y)$

Fermat prend comme valeurs de x des entiers juste supérieurs à \sqrt{n}

$$x = \begin{cases} \lceil \sqrt{n} \rceil \\ \lceil \sqrt{n} \rceil + 1 \\ \lceil \sqrt{n} \rceil + 2 \\ \dots \end{cases}$$

$\Rightarrow x^2$ est "légèrement supérieur" à n
et on espère que $x^2 - n$ est un caré parfait y^2
(d'où $x^2 - y^2 = n$)

ex: $n = 9167$ $\sqrt{n} \approx 95,7$

$$96^2 = 49 \pmod{9167}$$

$$\Rightarrow 96^2 = 7^2 \pmod{9167}$$

$$\left. \begin{array}{l} \operatorname{pgcd}(9167, 96+7) = 103 \\ \operatorname{pgcd}(9167, 96-7) = 89 \end{array} \right) \quad 103 \times 89 = 9167$$

Remarque: Dans les entiers de 1 à n , il y en a \sqrt{n} qui sont des carés parfaits
 \rightarrow complexité $\approx \sqrt{n}$

Raffinement: $n = 849239$
 $\sqrt{n} \approx 921,5$

$$\rightarrow 922^2 = 845 = 5 \times 13^2 \pmod{n}$$

$$923^2 = \dots$$

$$\rightarrow 933^2 = 2 \times 5^4 \times 17 \pmod{n}$$

$$\rightarrow 937^2 = 2 \times 5 \times 13^2 \times 17 \pmod{n}$$

$$922^2 \times 933^2 \times 937^2 = 2^2 \times 5^6 \times 13^4 \times 17^2 \pmod{n}$$

$$\Rightarrow x^2 = y^2 \pmod{n}$$

$$\text{avec } x = 922 \times 933 \times 937$$

$$y = 2 \times 5^3 \times 13^2 \times 17$$

$$\Rightarrow \begin{cases} \text{pgcd}(x+y, n) = 1229 \\ \text{pgcd}(x-y, n) = 691 \end{cases} \quad \boxed{\begin{aligned} 1229 \times 691 \\ = 849239 \end{aligned}}$$

Algorithm de cube quadratique (Pomerance)

On veut factoriser n

- On se fixe une base de factorisation $B = \{1, p_1, \dots, p_h\}$
- On dit qu'un entier est fiable (ou lisse / smooth)
s'il n'a que des petits facteurs premiers
En particulier qu'il est B -fiable si tous ses facteurs sont dans B
- On dit que b est B -adapté si le représentant de $b^2 \pmod{n}$ se trouvant dans $[-\frac{n}{2}, \frac{n}{2}]$ est B -fiable

1ère phase : Obtenir et stocker des entiers b_i B -adapté

$$\text{i.e.: } b_i^2 = (-1)^{\epsilon_i} p_1^{\alpha_{i,1}} \dots p_h^{\alpha_{i,h}} \pmod{n} \quad (*)$$

Puis, lorsqu'on a suffisamment de b_i :

2^e phase : À chaque relation (*) on associe un vecteur binaire
 $\vec{u}_i = (u_{i,0}, u_{i,1}, \dots, u_{i,h}) \in \mathbb{F}_2^{h+1}$
 où $\begin{cases} u_{i,0} = \epsilon_i \bmod 2 \\ u_{i,j} = \alpha_{i,j} \bmod 2 \quad (1 \leq j \leq h) \end{cases}$

algorithme linéaire (On cherche une combinaison linéaire non nulle des \vec{u}_i qui sort nulle (dans \mathbb{F}_2^{h+1}))

En revenant aux entiers mod n :

$$\sum_{i=1}^I \beta_i \vec{u}_i = \vec{0} \text{ dans } \mathbb{F}_2^{h+1} \text{ avec les } \beta_i \text{ non tous nuls}$$

$$\prod_{i=1}^I (b_i)^{\beta_i} = \prod_{i=1}^I (-1)^{\sum_{j=1}^h p_j \alpha_{i,j}} \quad \beta_i$$

$$= (-1)^{\sum_{i=1}^I \beta_i \epsilon_i} \quad = 0 \bmod 2$$

$$= 1 \quad = 0 \bmod 2$$

$$x^2 = y^2 \bmod n$$

$$\text{avec } x = \prod_{i=1}^I b_i^{\beta_i} \quad \text{et} \quad y = \prod_{j=1}^h p_j^{\frac{1}{2} \sum_{i=1}^I \beta_i \alpha_{i,j}}$$

Complexité de l'algorithme.

2^e phase : va donner le résultat à condition d'avoir $I \geq h+2$

nb de relations du type (*) trouvées lors de la phase 1

nb d'autres premiers dans la base de faisabilité B

1^{ère} phase :

Th: On pose $\Psi(x, T) = \text{Card} \{ n \text{ entier}, 1 \leq n \leq x, \text{ayant tous ses facteurs premiers} \leq T \}$

Pour $1 \leq T \leq x$, posons $v = \frac{\ln x}{\ln T}$

alors $\frac{\Psi(x, T)}{x} = v^{-v+o(1)}$

On prend $B = \{-1, p_1, \dots, p_h\}$ avec p_1, \dots, p_h = les entiers premiers $\leq T = \exp\left(\frac{1}{2}\sqrt{\ln x \ln x}\right)$

$$\text{d'où } v = \frac{\ln \sqrt{x}}{\ln T} = \frac{\frac{1}{2} \ln x}{\frac{1}{2} \sqrt{\ln x \ln x}} = \sqrt{\frac{\ln x}{\ln x \ln x}}$$

$$\text{et } \ln v \approx \frac{1}{2} \ln \ln x$$

\Rightarrow le nombre de valeurs à essayer (dans la 1^{ère} phase) pour obtenir $h+2$ relations du type (*) est :

$$\boxed{v^v \times (h+2)}$$

avec $v^v = e^{v \ln v} = e^{\frac{1}{2} \sqrt{\ln x \ln x}} = T$

$\rightarrow = T \times \underbrace{(h+2)}_{\substack{\text{nb d'entiers premiers} \\ \leq T}} \approx T \times \underbrace{\frac{T}{\ln T}}_{\substack{\text{d'après la th des} \\ \text{entiers premiers}}}$

Complexité de l'algorithme complet

$$\underbrace{\frac{T^2}{\ln T}}_{\substack{1^{\text{ère}} \text{ phase}}} + (h+2)^3 \underbrace{\approx \frac{T}{\ln T}}_{\substack{\text{Gauss}}} \rightarrow \left(\frac{T}{\ln T}\right)^3 = \boxed{e^{\frac{3}{2} \sqrt{\ln x \ln x}}}$$

- plus que polynomial
- moins qu'exponentiel
 $= \text{algorithme sous-exponentiel}$

Notation : $L_{\alpha, c}(z) := e^{c(\ln z)^\alpha (\ln \ln z)^{1-\alpha}}$

Remarque : $\alpha = 0 \rightarrow L_{0, c}(z) = e^{c(\ln \ln z)^0} = (\ln z)^c$

$\alpha = 1 \rightarrow L_{1, c}(z) = e^{c(\ln z)^1} = e^{c \ln z}$

Algorithm de crible quadratique de Pomerance

→ complexité $L_{\frac{1}{2}, \frac{3}{2}}(n)$

Mieux algorithme : GNFS (General Number Field Sieve)

Crible algébrique général
(H. Lenstra, A. Lenstra, Manasse, Pollard 1990)
(Crible du corps de nombres)

Complexité : $L_{\frac{1}{3}, c}(n) = \tilde{\mathcal{O}}(e^{c(\ln n)^{\frac{1}{3}} (\ln \ln n)^{\frac{2}{3}}})$
avec $c \approx 1,92$

ex : pour RSA , $L_{\frac{1}{3}, c}(n) \geq 2^{80}$

$\Leftrightarrow \underline{|n| \geq 1024 \text{ bits}}$

ch 1 : Problèmes difficiles de théorie des nombres

I) Complexité et cryptographie

II) Factorisation

① Complexité

② Idée de Fermat

$$\begin{cases} x^2 = y^2 \pmod{n} \\ \text{avec } x = \pm y \pmod{n} \end{cases}$$

↳ algorithme de cube quadratique (Pomerance)

Complexité : $\frac{T^2}{\ln T} + (h+2)^3 \xrightarrow{\text{2e phase}} \left(\frac{T}{\ln T}\right)^3 = e^{\frac{3}{2}}$ flun lun

pre phase (cube) $\approx \frac{T}{\ln T}$ flun lun

$\approx \frac{T^2}{\ln T} \approx e^{\frac{1}{2}}$ flun lun

Remarque : L'exposant 3 vient du pivot de Gauss

Mais ici on a un système qui est creux

$$\begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix} \xrightarrow{h} \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \end{pmatrix}$$

$$b_i^2 = \prod_{j=1}^h p_j^{d_{ij}} \pmod{n}$$

le nombre de facteurs premiers qui interviennent dans la décomposition est $O(\ln n)$

→ algorithme de Block-Lanczos

$$O(dh^2)$$

où h est la dimension du système et d le nombre maximal d'éléments non nuls dans chaque ligne.

$$\text{Ici } d = o(h) \Rightarrow O(h^{2+\epsilon}) \text{ avec } h \approx \frac{T}{\ln T}$$

$$T = e^{\frac{1}{2} \sqrt{\ln \ln n}}$$

$$\Rightarrow O(e^{(1+\epsilon) \sqrt{\ln \ln n}})$$

complexité pour le critère quadratique de Pomerance

III] Problème du logarithme discret

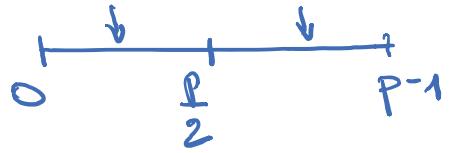
① Complexité

pb du log discret : p premier, g générateur de $(\mathbb{Z}/p\mathbb{Z})^*$
 à partir de $y = g^x \pmod{p}$
 peut-on retrouver x ?

Pb de décision :

<u>Instance</u> : P, g, y, t	<u>Question</u> : le logarithme discret de y par rapport à g est-il $\leq t$?
--------------------------------	---

- Si on a un algorithme en temps polynomial pour résoudre le pb du log discret, alors on a un algo en temps polynomial pour résoudre ce pb de décision
- Réciproquement : soit \mathcal{A} un algo en temps polynomial pour le pb de décision, alors on peut résoudre le pb du log discret en faisant un nombre polynomial d'appels à \mathcal{A}
 (utiliser une méthode de dichotomie, avec $t = \frac{f}{2}$)



→ Nombre d'appels (étapes de la dichotomie)
 $O(\ln p)$, on trouve le logarithme discret

Algorithme naïf : $y = g^x \bmod p$: on cherche à trouver x

- recherche exhaustive sur x → $O(p)$
 exponentiel
 $= O(e^{\frac{\ln p}{\ln g}})$
 taille de l'entité

- Algorithme baby step - giant step

$$y = g^x \bmod p \quad a = \lfloor \sqrt{p} \rfloor$$

$$\begin{array}{c} x \\ \parallel \\ n \end{array} \quad \begin{array}{l} \rightarrow x = aq + r \\ y = g^x \bmod p \\ \Leftrightarrow y = g^{aq+r} \bmod p \end{array}$$

$$\Leftrightarrow y g^{-r} \equiv g^{aq} \bmod p$$

r	$y g^{-r} \bmod p$	$g^q \bmod p$
0	—	—
1	—	—
2	—	—
...	—	—
$\approx \sqrt{p}$	—	—

$$x = aq + r \text{ avec } 0 \leq r < a < \sqrt{p} \quad \begin{matrix} x/a \\ r/q \end{matrix}$$

$$q = \frac{x-r}{a} < \frac{p-1}{\sqrt{p}-1} = \sqrt{p}+1 \quad (\text{car } a > \sqrt{p}-1)$$

pour trouver r et q , on trie les deux tables

$$\begin{matrix} O(N \ln N) \\ \downarrow \\ \text{taille de la table} \end{matrix} = O(\sqrt{p} \ln p)$$

$$\rightarrow \text{complexité: } O(\sqrt{p} \ln p)$$

Remarque: avec l'algorithme rho de Pollard
on peut atteindre $O(\sqrt{p})$
exponentiel
 $O(e^{\frac{1}{2} \ln p})$

② Méthode de calcul d'indices (pour trouver x tel que $g^x \equiv y \pmod{p}$)

1ère phase: On choisit $B = \{p_0, p_1, \dots, p_h\}$

Idée: on choisit c_i aléatoirement
on calcule $g^{c_i} \pmod{p} \rightarrow$ représentant dans $[-\frac{p}{2}, \frac{p}{2}]$

et on espère que

$$g^{c_i} \pmod{p} = \prod_{j=0}^h p_j^{\alpha_{ij}} \quad (*)$$

$$\Rightarrow c_i = \sum_{j=0}^h \alpha_{ij} \log_p p_j \pmod{p-1} \quad (**)$$

2^e phase \Rightarrow lorsque l'on a obtenu $(h+1)$ équations du type (**)
linéairement indépendantes

on trouve tous les $\log_g p_j$ par pivot de Gauss
(ou Block-Lanczos)

3^e phase : On veut trouver x tel que $[y = g^x \bmod p]$

\rightarrow on tire aléatoirement un exposant e
jusqu'à ce que $y g^e \bmod p$ se décompose
dans la base B

$$y g^e = \prod_{j=0}^h p_j^{\beta_j} \bmod p$$

$$\Rightarrow \underbrace{\log_g y}_{} + e = \sum_{j=0}^h \beta_j \log_g p_j \pmod{p-1}$$

$$\Rightarrow \log_g y = \left[\sum_{j=0}^h \beta_j \underbrace{\log_g p_j}_{} - e \right] \bmod(p-1)$$

trouvé lors de
la 2^e phase

Complexité : $O\left(e^{(1+o(1))\sqrt{\ln p + \ln \ln p}}\right)$

c'est à-dire : $L_{\frac{1}{2}, 1+o(1)}(p)$

Remarque : le meilleur algorithme connu est en

$$O\left(e^{c(\ln p)^{1/3} (\ln \ln p)^{2/3}}\right)$$

i.e. $\underbrace{L_{\frac{1}{3}, c}(p)}_{\text{sous-exponentiel}} \Rightarrow$ pour une sécurité en 2^{80}
il suffit de prendre $|p| \geq 1024$ bits

Rémarque: le record de facteurisation actuel est 820 bits (2019)

Ch 2: L'algorithme RSA en pratique

I) Rappels sur RSA

① Définition

- 1976 : W. Diffie, M. Hellman, New Directions in Cryptography
↳ cryptographie à clé publique asymétrique
- 1976 : puzzles de Merkle (1er exemple de chiffrement asymétrique)
- 1977 : algorithme RSA (Rivest, Shamir, Adleman)
pure Turing

Paramètres

- Choisir e exposant impair $e \neq 1$
- Choisir p et q , premiers, $\boxed{p \neq q}$
tels que $\text{pgcd}(p-1, e) = \text{pgcd}(q-1, e) = 1$
- Poser $n = p \times q$
- Calculer $d = e^{-1} \bmod \varphi(n)$ grâce à l'algorithme d'Euclide étendu
 $= (p-1)(q-1)$

Théorème: Soit $f: \begin{cases} \mathbb{Z}/n\mathbb{Z} \rightarrow \mathbb{Z}/n\mathbb{Z} \\ x \mapsto x^e \bmod n \end{cases}$

Alors f est une bijection et

$f^{-1}: \begin{cases} \mathbb{Z}/n\mathbb{Z} \rightarrow \mathbb{Z}/n\mathbb{Z} \\ y \mapsto y^d \bmod n \end{cases}$

Démonstration : $\forall x \in \mathbb{Z}/n\mathbb{Z}, (x^e)^d = x \pmod{n}$

$$\Leftrightarrow x^{ed} = x \pmod{n}$$

avec $ed = 1 \pmod{\varphi(n)}$

cela vient du fait que $x^{\varphi(n)} = 1 \pmod{n}$
pour x tel que $\text{pgcd}(x, n) = 1$

Remarque : si $\text{pgcd}(x, n) > 1$

alors x divisible par p
ou par q

$$\Rightarrow x^p = x \pmod{p}$$

ou $x^q = x \pmod{q}$

Dans tous les cas $\begin{cases} x^p = x \pmod{p} \\ \text{et } x^q = x \pmod{q} \end{cases}$
 $\Rightarrow x^{ed} = x \pmod{n}$

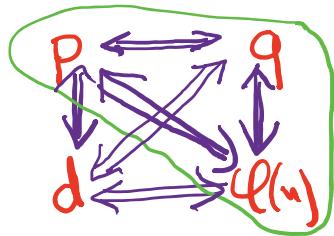
② Sécurité

Objectif d'un attaquant : $\begin{bmatrix} 1) \text{ Retrouver la clé secrète} \\ 2) \text{ Pour des valeurs } y \text{ calculer } f^{-1}(y) \end{bmatrix}$

1) Retrouver la clé secrète : p, q, d, $\varphi(n)$

à partir de la clé publique : n, e

Si on retrouve un des quatre éléments $p, q, d, \varphi(n)$
alors on obtient facilement les 3 autres



$$q = n/p$$

$$d = e^{-1} \bmod (p-1)(q-1)$$

$$\varphi(n) = (p-1)(q-1)$$

$$d = e^{-1} \bmod \varphi(n)$$

Si on a $\varphi(n)$ peut-on retrouver p et q ?

oui: $\varphi(n) = (p-1)(q-1) = \underbrace{pq - (p+q) + 1}_{=n}$

$$\begin{cases} p+q = n - \varphi(n) + 1 \\ pq = n \end{cases}$$

→ p et q sont racines de $X^2 - (n - \varphi(n) + 1)X + n$

$$\{p, q\} = \frac{n - \varphi(n) + 1 \pm \sqrt{(n - \varphi(n) + 1)^2 - 4n}}{2}$$

• Si on connaît d , peut-on retrouver p et q ?

$$ed = 1 \bmod \varphi(n)$$

Prenons x aléatoire, et posons:

$$y = x \frac{ed-1}{2} \bmod n$$

ensuite car $ed-1$ est divisible par $\varphi(n)$
($\varphi(n)$ pair
 $(p-1)(q-1)$)

$$\text{On a: } y^2 = x^{ed-1} = 1 \bmod n$$

(sauf si $\text{pgcd}(x, n) > 1$, auquel cas on aurait
directement la factorisation
de n)

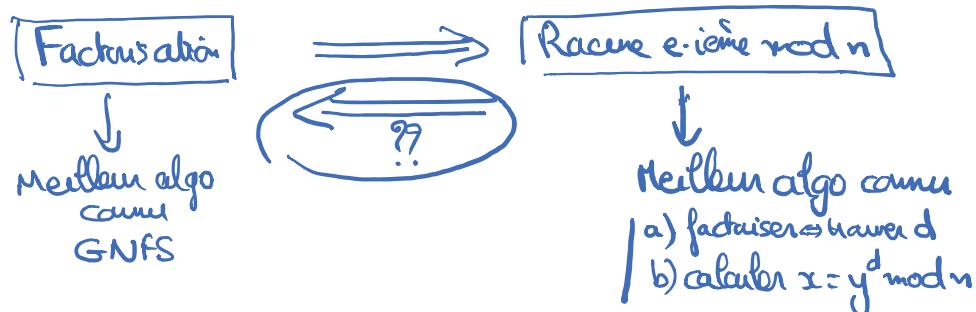
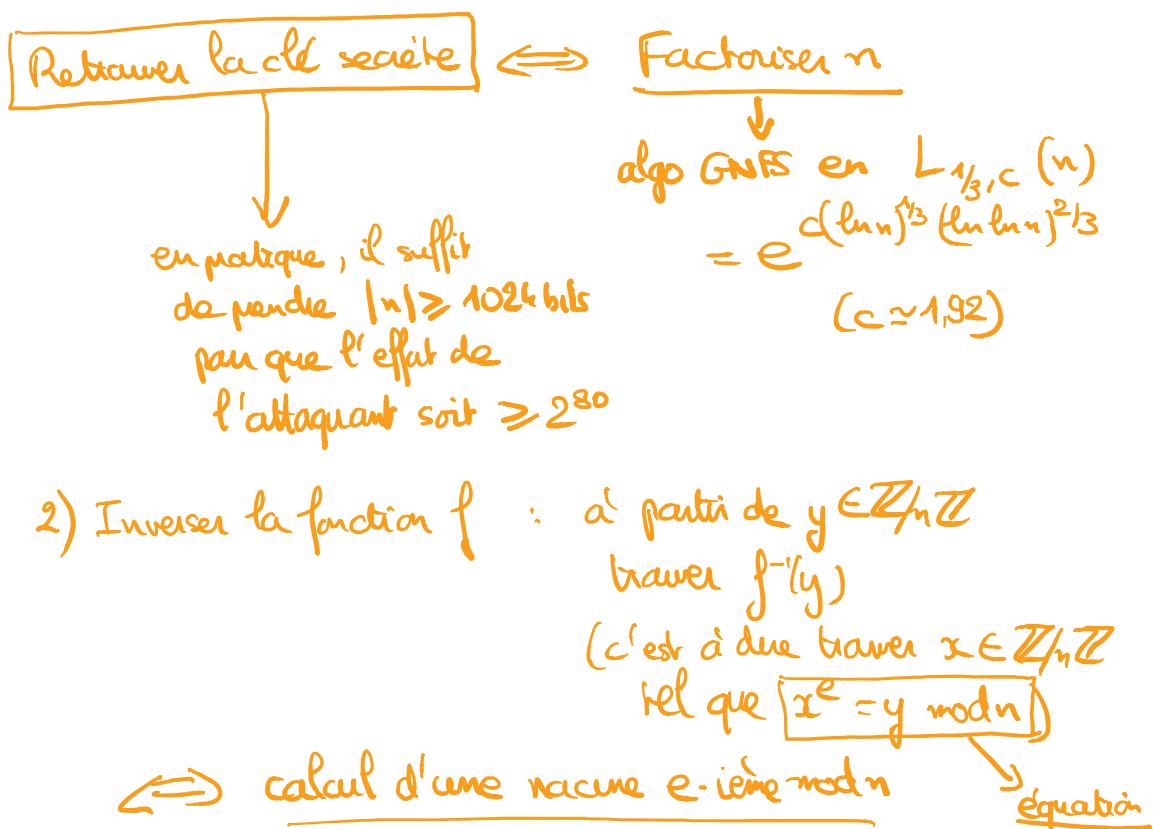
L'équation $y^2 = 1 \bmod n$ possède 4 solutions

$$\Leftrightarrow \begin{cases} y^2 = 1 \bmod p \Leftrightarrow y = \pm 1 \bmod p \\ y^2 = 1 \bmod q \Leftrightarrow y = \pm 1 \bmod q \end{cases}$$

$$\Leftrightarrow \left| \begin{array}{l} y \equiv 1 \pmod{n} \\ y \equiv -1 \pmod{n} \\ y \equiv \alpha \pmod{n} \\ y \equiv -\alpha \pmod{n} \end{array} \right| \rightarrow ?$$

on brise la factorisation en calculant $\text{pgcd}(n, y - \alpha) = p$ ou q

où α est défini par $\begin{cases} \alpha \equiv 1 \pmod{p} \\ \alpha \equiv -1 \pmod{q} \end{cases}$ (via le théorème des restes chinois)



II] RSA en signature

① Problématique

- Signature = une des motivations de l'introduction de la cryptographie asymétrique

en symétrique: Alice $\xrightarrow[\mathbf{K}]{M, \text{MAC}_K(M)} \mathbf{Bob}$

pb: Bob ne peut pas convaincre un tiers que le message M a bien été émis par Alice

M = "Moi Alice je donne 10000€ à Bob"

Cela illustre qu'on ne peut pas garantir la propriété de non répétition en cryptographie symétrique

→ introduction de la crypto asymétrique, où seule Alice connaît la clé secrète.

- Avec RSA, on peut imaginer de signer un message M de la façon suivante:

$$S = M^d \bmod n \quad (= f^{-1}(M))$$

Alice $\xrightarrow[M, \text{Sign}_{SK}(M)=S]{} \mathbf{Bob}$

SK/
Alice

Verif_{OPK_Alice}(M, S) ? OK

pb 1: Cela n'a de sens que si $0 \leq M < n$

Si on ~~essait~~ pouvait facilement produire plusieurs messages ayant la même signature

$M, M+n, M+2n, \dots$

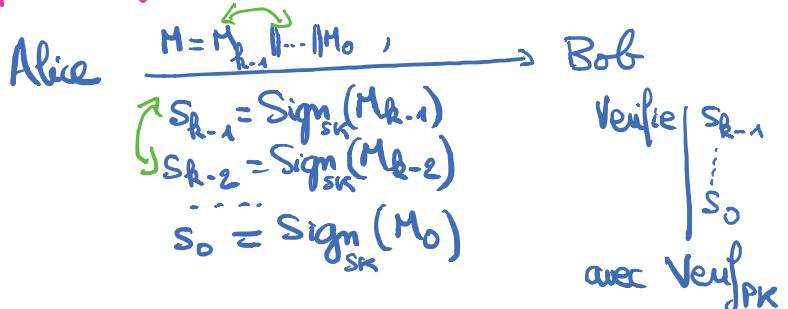
Solution possible : pour M quelconque, on peut écrire

$$M = M_{k-1} n^{k-1} + M_{k-2} n^{k-2} + \dots + M_1 n + M_0$$

(écriture de l'entier M "en base n ")

(avec $\forall i, M_i \in [0, n]$)

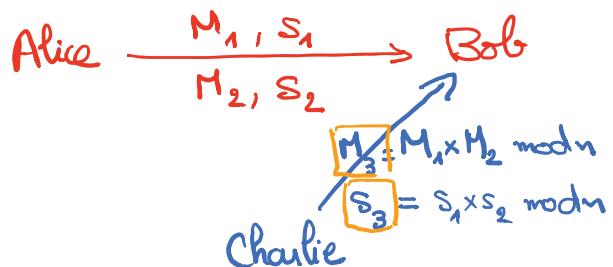
puis signer chaque élément M_i :



+ d'autres signatures pour garantir le bon ordonnancement des blocs

pb 2 : Sécurité

$$\begin{aligned} s_1 &= M_1^d \pmod{n} \\ s_2 &= M_2^d \pmod{n} \end{aligned}$$



$$M_3 = M_1 \times M_2 \pmod{n}$$

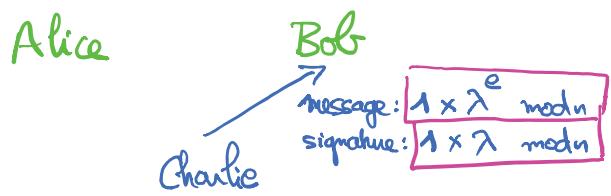
$$\text{à pour signature } S_3 = (M_1 \times M_2)^d = s_1 \times s_2 \pmod{n}$$

pb lié à la multiplicativité de f^{-1} (ou de f)

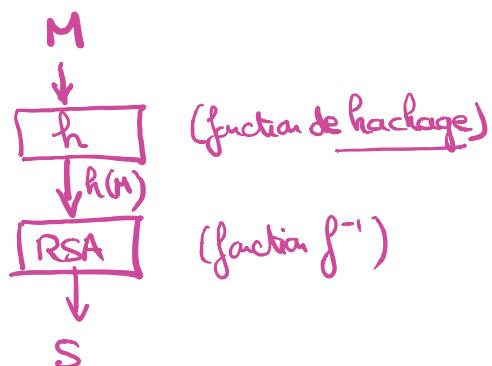
Remarque : Alice $\xrightarrow{M_1, s_1}$ Bob
 Charlie $\boxed{M_1^2 \pmod{n}}$ ou $\boxed{\lambda^e M_1 \pmod{n}}$ (quelque chose)

$$\lambda^e M_1 \pmod{n}$$

$$(\lambda^e M_1)^d = \lambda^d M_1^d = \lambda \cdot s_1 \pmod{n}$$



→ Solution (pour les pts 1 et 2) : Utiliser une fonction de hachage



② Fonction de hachage

$$h: \{0,1\}^* \longrightarrow \{0,1\}^l \quad \text{où } l \in \mathbb{N} \text{ est fixé}$$

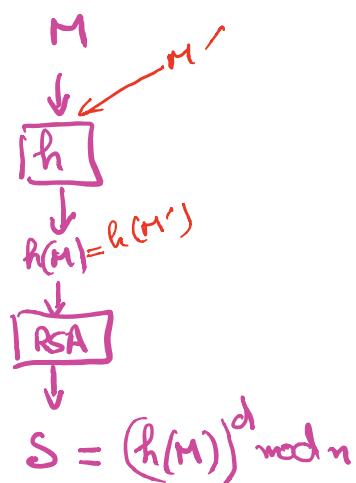
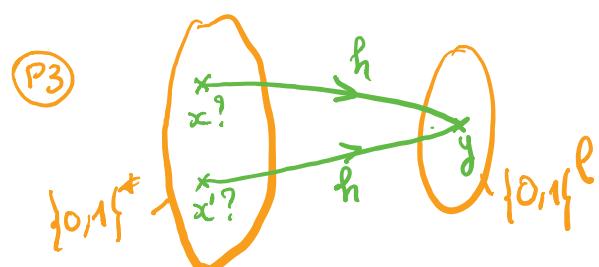
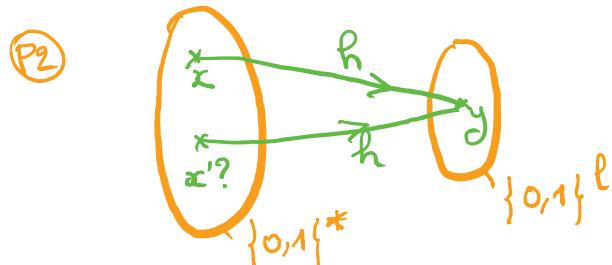
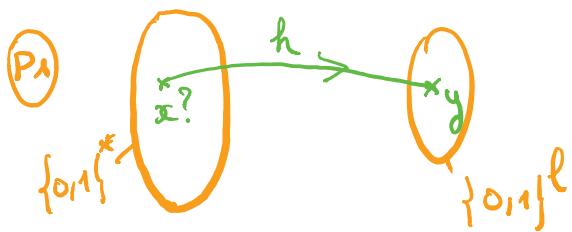
$$\{0,1\}^* = \bigcup_{n \in \mathbb{N}} \underbrace{\{0,1\}^n}_{\text{messages de } n \text{ bits}}$$

Définition: $h: \{0,1\}^* \longrightarrow \{0,1\}^l$ (l fixé)
est appelée fonction de hachage si elle vérifie les 3 propriétés suivantes :

(P1) h est à sens unique (one-way) : pour $y \in \{0,1\}^l$, il est calculatoirement difficile de trouver $x \in \{0,1\}^*$ tel que $h(x) = y$

(P2) h est à collisions faibles difficiles (second-preimage résistant) : pour $x \in \{0,1\}^*$, et $y = h(x)$, il est calculatoirement difficile de trouver $x' \in \{0,1\}^*$ tel que $x \neq x'$ et $h(x') = y$

(P3) h est à collisions faites difficiles (collision resistant) : il est calculatoirement difficile de trouver $x \in \{0,1\}^*$ et $x' \in \{0,1\}^*$ tels que $\begin{cases} x \neq x' \\ \text{et } h(x) = h(x') \end{cases}$



Si P1 est faux :

Alice $\xrightarrow[M_1, s_1]{M_2, s_2}$ Bob

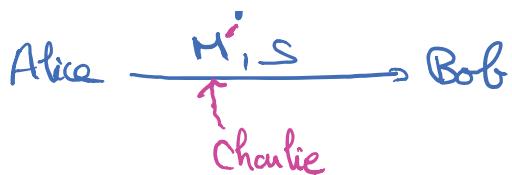
$$s_1 = (h(M_1))^d \bmod n$$

$$s_2 = (h(M_2))^d \bmod n$$

On calcule $M_3 / h(M_3) = \underline{h(M_1) \times h(M_2)} \bmod n$

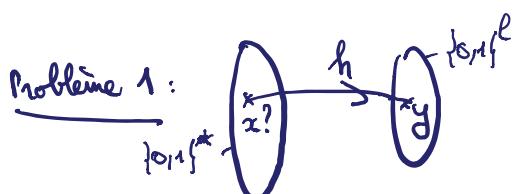
$$\Rightarrow s_3 = s_1 \times s_2$$

Si P2 est faux : Charlie peut remplacer M par M' sans changer la valeur de la signature.

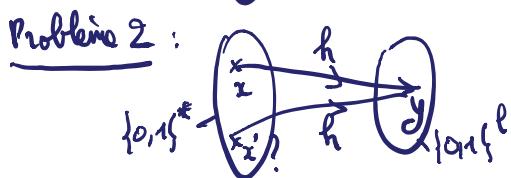


Si P_3 est faux : Alice peut ~~encore~~ obtenir M et $M' \neq M$
tels que $h(M) = h(M')$
puis envoier M, S à Bob
et prétendre ensuite qu'elle a envoyé M'
 \rightarrow pb de non-réputation avec la

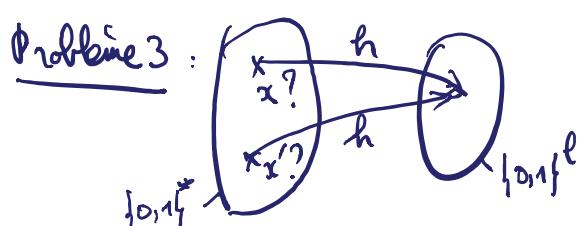
$P_3 \Rightarrow P_2 \Rightarrow P_1$



Savoir résoudre le pb 1 \Rightarrow Savoir résoudre le pb 2
OK



Savoir résoudre le pb 2 \Rightarrow Savoir résoudre le pb 3
OK



- Algorithmus générique pour le problème 1 :
(indépendant de la construction précise de h)

- tirer aléatoirement
 $x \in \{0,1\}^*$
jusqu'à ce qu'on obtienne
 $h(x) = y$

probabilité de succès
 $P = \frac{1}{2^l}$ \Rightarrow complexité
 $O(2^l)$

Consequence : on prend $\boxed{l \geq 80}$
 (si on veut une sécurité en 2^{80})

- Algorithme générique pour le problème 2 : - tirez aléatoirement

$$x' \in \{0,1\}^* (\neq \emptyset)$$

jusqu'à obtenir

$$h(x') = y$$

\Rightarrow même complexité $\boxed{O(2^l)}$

Consequence : on prend $\boxed{l \geq 80}$

- Algorithme générique pour le problème 3 :

On tire aléatoirement $x_1, x_2, x_3, \dots, x_k$
 et on calcule $y_i = h(x_i)$ ($1 \leq i \leq k$)
 jusqu'à obtenir une relation du type $\boxed{y_i = y_j \text{ (} i < j \text{)}}$
 collision

P = probabilité d'obtenir une telle collision

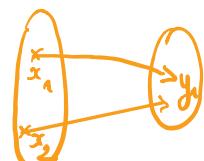
$1 - P =$ ————— qu'aucune collision n'apparaisse

$P_1 :=$ proba que le tirage de x_1 ne provoque pas de collision

$$\boxed{P_1 = 1}$$

$P_2 :=$ ————— le tirage de x_2 ne provoque pas de collision

$$P_2 = \frac{2^l - 1}{2^l} \Rightarrow \boxed{P_2 = 1 - \frac{1}{2^l}}$$



$P_3 :=$ ————— x_3 —————

$$P_3 = \frac{2^l - 2}{2^l} \Rightarrow \boxed{P_3 = 1 - \frac{2}{2^l}}$$



$$P_k = \frac{2^l - (k-1)}{2^l} \Rightarrow P_k = 1 - \frac{k-1}{2^l}$$

Au total, $1 - P = p_1 \times p_2 \times p_3 \times \dots \times p_k$

$$1 - P = 1 \times \left(1 - \frac{1}{2^l}\right) \times \left(1 - \frac{2}{2^l}\right) \times \dots \times \left(1 - \frac{k-1}{2^l}\right)$$

$$\ln(1-P) = \underbrace{\ln 1}_{=0} + \ln\left(1 - \frac{1}{2^l}\right) + \dots + \ln\left(1 - \frac{k-1}{2^l}\right)$$

$$\ln(1-u) \approx -u \quad (\ln(1-u) = -u - \frac{u^2}{2} + \dots)$$

$$\ln(1-P) \approx -\frac{1}{2^l} - \frac{2}{2^l} - \dots - \frac{k-1}{2^l}$$

$$\approx -\frac{1}{2^l} \left[\underbrace{1+2+3+\dots+(k-1)}_{=\frac{k(k-1)}{2}} \right]$$

$$\ln(1-P) \approx -\frac{k(k-1)}{2 \times 2^l} \Leftrightarrow \underbrace{\frac{k(k-1)}{2^l}}_{\approx k^2} \approx 2^l \times 2 \ln \frac{1}{1-P}$$

$$1 - P \approx e^{-\frac{k(k-1)}{2 \times 2^l}}$$

$P \approx 1 - e^{-\frac{k(k-1)}{2 \times 2^l}}$
$k \approx 2^{\frac{l+2}{2}} \sqrt{2 \ln \frac{1}{1-P}}$

$$\begin{cases} k \rightarrow +\infty \Rightarrow P \rightarrow 1 \\ l \rightarrow +\infty \Rightarrow P \rightarrow 0 \end{cases}$$

Ch 2 : L'algorithme RSA en pratique

I] Rappels sur RSA

- ① Définition
- ② Sécurité

II] RSA en signature

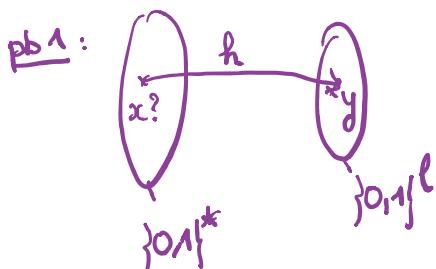
① Problématique

$$S = M^d \text{ mod } n$$

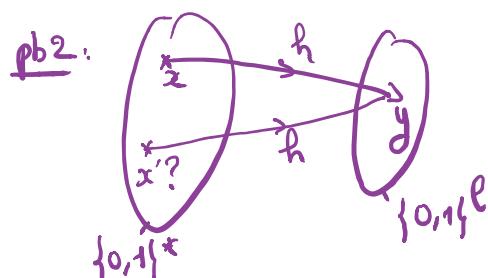
~~(incorrect)~~



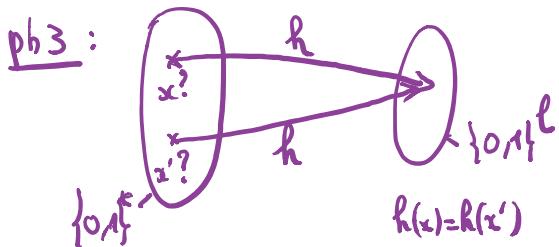
② Fonctions de hachage



Algo générique en $O(2^l)$
 → on prend $l \geq 80$



Algo générique en $O(2^l)$
 → on prend $l \geq 80$



Algo générique : On génère des messages aléatoires x_1, x_2, \dots, x_R et on calcule $y_i = h(x_i)$ ($1 \leq i \leq k$) jusqu'à ce qu'on trouve une relation du type $y_i = y_j$ ($i < j$)

P = probabilité d'obtenir une collision

$$P \approx 1 - e^{-\frac{k(k-1)}{2 \times 2^k}}$$

$$k \approx 2^{k/2} \sqrt{2 \ln \frac{1}{1-P}}$$

paradoxe
des anniversaires

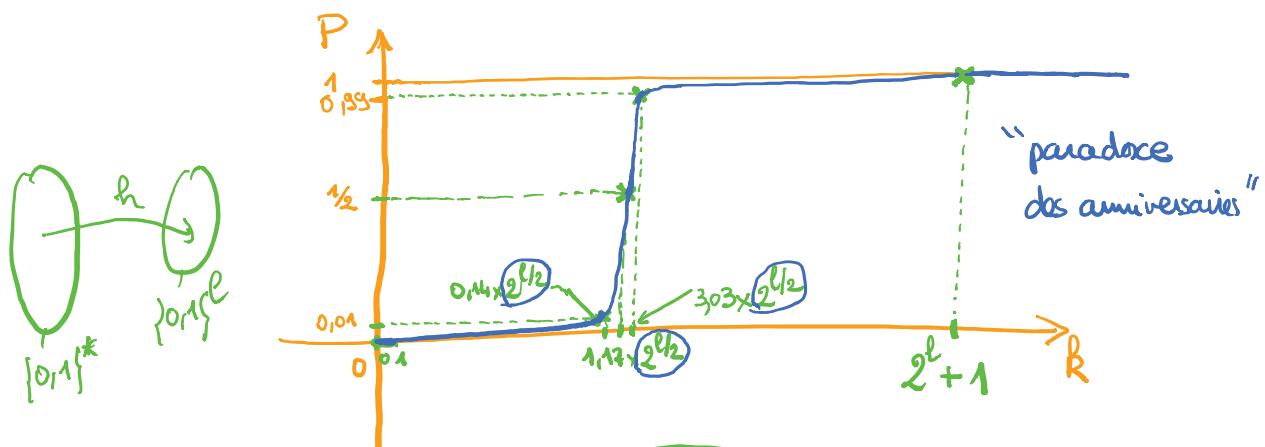
$$h : \{ \text{personnes} \} \longrightarrow \{ 1, 2, 3, \dots, 365 \}$$

P = probabilité que 2 personnes aient la même date de naissance
↳ jour + mois

$$P \approx 1 - e^{-\frac{k(k-1)}{2 \times 365}}$$

$$k \approx \sqrt{365} \sqrt{2 \ln \frac{1}{1-P}}$$

Ex: $P = \frac{1}{2} \Leftrightarrow k \approx \sqrt{365} \sqrt{2 \ln \frac{1}{1-\frac{1}{2}}} \approx 23$



$$P = \frac{1}{2} \Leftrightarrow k \approx 2^{l/2} \sqrt{2 \ln \frac{1}{1-\frac{1}{2}}} \approx 1.17 \times 2^{l/2}$$

≈ 1.17

$$P = 0.99 \Leftrightarrow k \approx 2^{l/2} \sqrt{2 \ln \frac{1}{1-\frac{99}{100}}} = 2^{l/2} \sqrt{2 \ln 100} \approx 3.03 \times 2^{l/2}$$

≈ 3.03

$$P = 0.01 \Leftrightarrow k \approx 2^{l/2} \sqrt{2 \ln \frac{1}{1-\frac{1}{100}}} = 2^{l/2} \sqrt{2 \ln \frac{100}{99}} \approx 0.14 \times 2^{l/2}$$

≈ 0.14

On sait que l'algorithme générique va donner une collision dès que $k \approx 2^{l/2}$

Pour la trouver,

on peut trier la table contenant les y_i

→ complexité $O(k \ln k)$
avec $k \approx 2^{l/2}$

i	y_i
1	—
2	—
⋮	—
i	—
⋮	—
k	—

calcul de $h(x_i)$

$$\rightarrow \text{complexité } O(2^{l/2} \ln 2^{l/2}) = O(l \cdot 2^{l/2})$$

puis trouver la collision en $O(k) = O(2^{l/2})$

$\boxed{O(l \cdot 2^{l/2})}$

Anélation : $\mathcal{O}(2^{l/2})$ pour l'algorithme générique du pb3

\Rightarrow on prend $\frac{l \geq 160}{l \geq 256}$ pour avoir une sécurité en 2^{80}
 (ou 2^{128})

③ Exemple de la fonction de hachage de Chaum - Van Heijst - Pfitzmann

Soit p premier tel que $q = \frac{p-1}{2}$ soit également premier
(i.e : p est un nombre premier fort)

$$p = 2q + 1$$

↓
menier

Soit g et h deux éléments d'ordre q dans \mathbb{Z}_p^*

H: fonction de compression

$$\begin{aligned} \text{message} &= (m_1, m_2) \simeq 2046 \text{ bits} \\ \text{haché} &\simeq 1024 \text{ bits} \xleftarrow{H} \end{aligned}$$

$$p \approx 1024 \text{ bits}$$

$$q = \frac{p-1}{2} \approx 1023 \text{ bits}$$

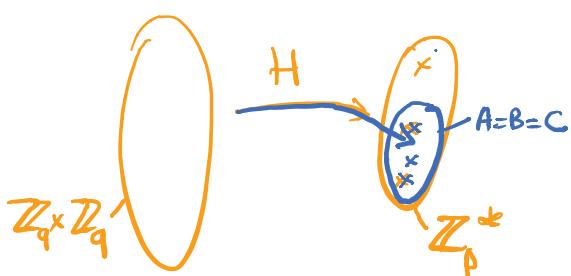
- Quelle est l'image de H

$$A = \langle g \rangle$$

$$B = \langle h \rangle$$

$$C = \{ u \in \mathbb{Z}_p^* \mid u^q \equiv 1 \pmod{p} \}$$

$$A \subseteq C \quad \text{et} \quad B \subseteq C$$



1

$$\text{Card } C \leq q = \text{Card } A = \text{Card } B \quad \left| \Rightarrow A=B=C \right.$$

Donc $\text{Im } H \subseteq A=B=C$

$$\text{De plus si } y \in A=B=C, \exists (m_1, m_2) \in \mathbb{Z}_q \times \mathbb{Z}_q / H(m_1, m_2) = y$$

il suffit de prendre
 $m_1=0$ et $m_2 / h^{m_2} = y \pmod{p}$
 existe un $y \in \langle h \rangle$

$$M = (0, m_2) \rightarrow H(M) = h^{m_2} \pmod{p}$$

- Supposons que Charlie ait trouvé une collision :

$$H(m_1, m_2) = H(m_3, m_4) \quad \text{avec } (m_1, m_2) \neq (m_3, m_4)$$

$$\Leftrightarrow g^{m_1} h^{m_2} = g^{m_3} h^{m_4} \pmod{p}$$

$$\Leftrightarrow g^{m_1 - m_3} = h^{m_4 - m_2} \pmod{p}$$

Considérons $\delta = \text{pgcd}(m_4 - m_2, p-1)$

$\xrightarrow{\text{diviseurs :}}$

1	2	q	p-1
$= 2q$ avec q premier			

1^e cas: si $\delta = 1 \rightarrow (m_4 - m_2)$ a un inverse mod $(p-1)$

alors $h = g^{(m_1 - m_3)[(m_4 - m_2)^{-1} \pmod{p-1}]} \pmod{p}$

[
 → on a trouvé un logarithme discret de h
 par rapport à la base g]

2^e cas: si $\delta = 2 \rightarrow \text{pgcd}(m_4 - m_2, p-1) = 2$

$$\Rightarrow \text{pgcd}(m_4 - m_2, q) = 1$$

$\Rightarrow (m_4 - m_2)$ a un inverse mod q

alors $h = g^{(m_1 - m_3)[(m_4 - m_2)^{-1} \pmod{q}]} \pmod{p}$

→ on a trouvé un logarithme discret de h
par rapport à la base g

3^e cas: Si $\delta = q$ ou $\delta = p-1$

$$\text{alors } \text{pgcd}(m_4 - m_2, p-1) = q \text{ ou } p-1$$

$$\Rightarrow q \text{ divise } \frac{m_4 - m_2}{\in \mathbb{Z}_q \in \mathbb{Z}_q} \Rightarrow m_2 = m_4 \text{ dans } \mathbb{Z}_q$$

$$g^{m_1 - m_3} = h^{\frac{m_4 - m_2}{=0 \text{ dans } \mathbb{Z}_q}} \bmod p \Rightarrow g^{m_1 - m_3} = 1 \bmod p$$

$$\Rightarrow m_1 - m_3 = 0 \text{ dans } \mathbb{Z}_q$$

$$\Rightarrow m_1 = m_3 \text{ dans } \mathbb{Z}_q$$

$$\Rightarrow \underbrace{(m_1, m_2)}_{\text{contradiction}} = (m_3, m_4), \text{ dans } \mathbb{Z}_q \times \mathbb{Z}_q$$

Conclusion : Si Charlie trouve une collision pour H
alors il connaît un logarithme discret de h
dans la base g

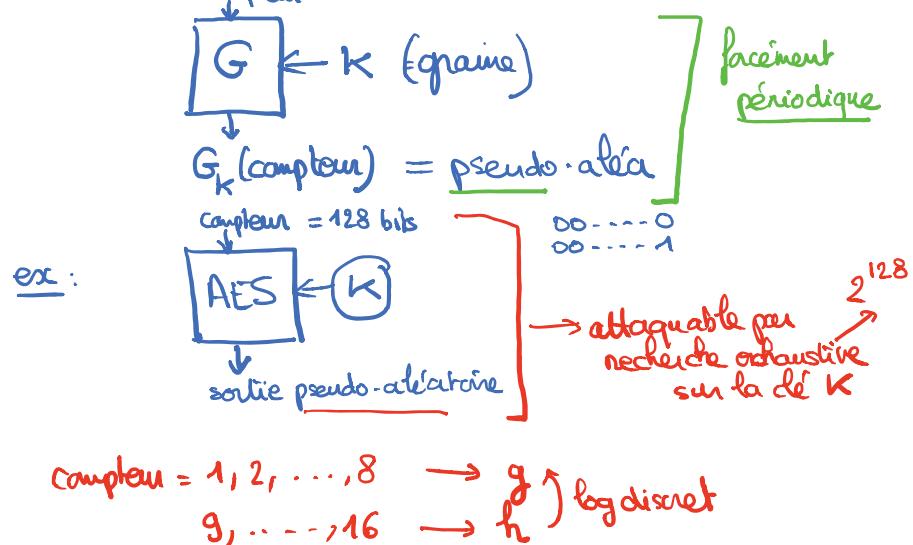
Consequence : "On" génère g et h aléatoires
 le concepteur (à priori sans connaître $\log_g h$)
 de la fonction puis publie $H: \mathbb{Z}_q \times \mathbb{Z}_q \rightarrow \mathbb{Z}_p^*$
 de la fraction

$$\begin{cases} (m_1, m_2) \mapsto g^{m_1} h^{m_2} \bmod p \end{cases}$$

Remarque : Pour que cela ait un sens cryptographique,
 il faut pouvoir convaincre qu'on ne connaît pas le
 logarithme discret de h dans la base g

Méthode possible : g et h sont dans \mathbb{Z}_p^*
 { d'ordre q

idée : prendre un générateur pseudo-aléatoire,



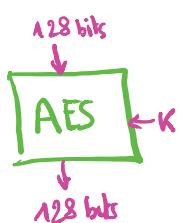
④ Construction de Merkle-Damgård

$$h: \{0,1\}^* \rightarrow \{0,1\}^t$$

Remarque : En cryptographie symétrique, si on a un message

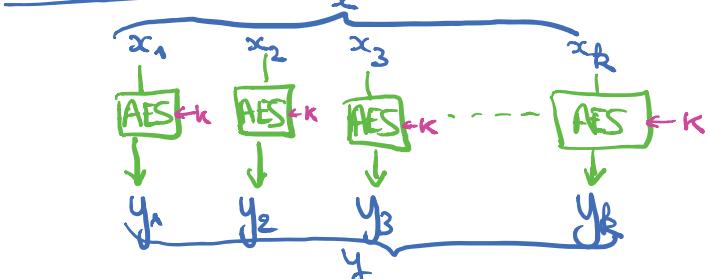
$$x = x_1 \parallel x_2 \parallel x_3 \parallel \dots \parallel x_k$$

128 bits 128 bits 128 bits 128 bits

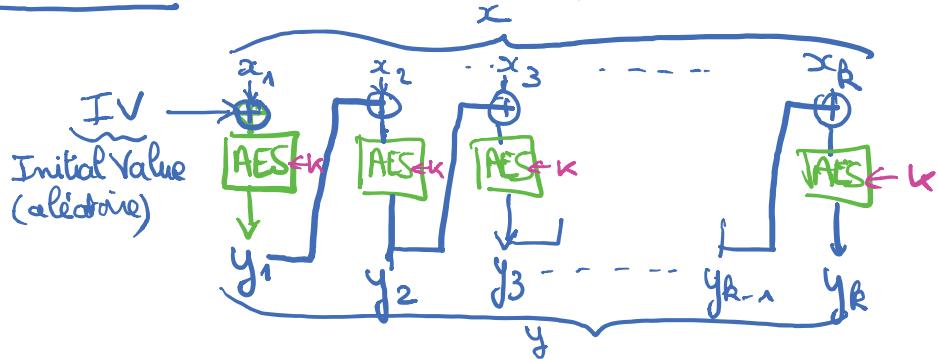


Modes d'opération = façon d'organiser les calculs d'AES pour chiffrer x

Mode ECB (Electronic Code Book)



Mode CBC (Cipher Block Chaining)



Alice IV, $y = (y_1, \dots, y_k)$ Bob

$$h: \{0,1\}^* \rightarrow \{0,1\}^l \quad : \text{hachage}$$

$$f: \{0,1\}^m \rightarrow \{0,1\}^l \quad \text{avec } m > l \quad (m \text{ et } l \text{ fixés})$$

fonction de compression

Construction de Merkle-Damgård (1989) : comment obtenir une fonction de hachage à partir d'une fonction de compression ?

Hypothèse : $m \geq l+2$

$x \in \{0,1\}^*$, comment définir $h(x)$?

$$x = x_1 \parallel x_2 \parallel x_3 \parallel \dots \parallel x_{k-1} \parallel x_k$$

$\overbrace{\hspace{10em}}$ $\overset{(m-l-1) \text{ bits}}{\curvearrowright}$ $\overbrace{\hspace{10em}}$ $\overset{(m-l-1) \text{ bits}}{\curvearrowright}$ $\overbrace{\hspace{10em}}$ $\overset{(m-l-1) \text{ bits}}{\curvearrowright}$

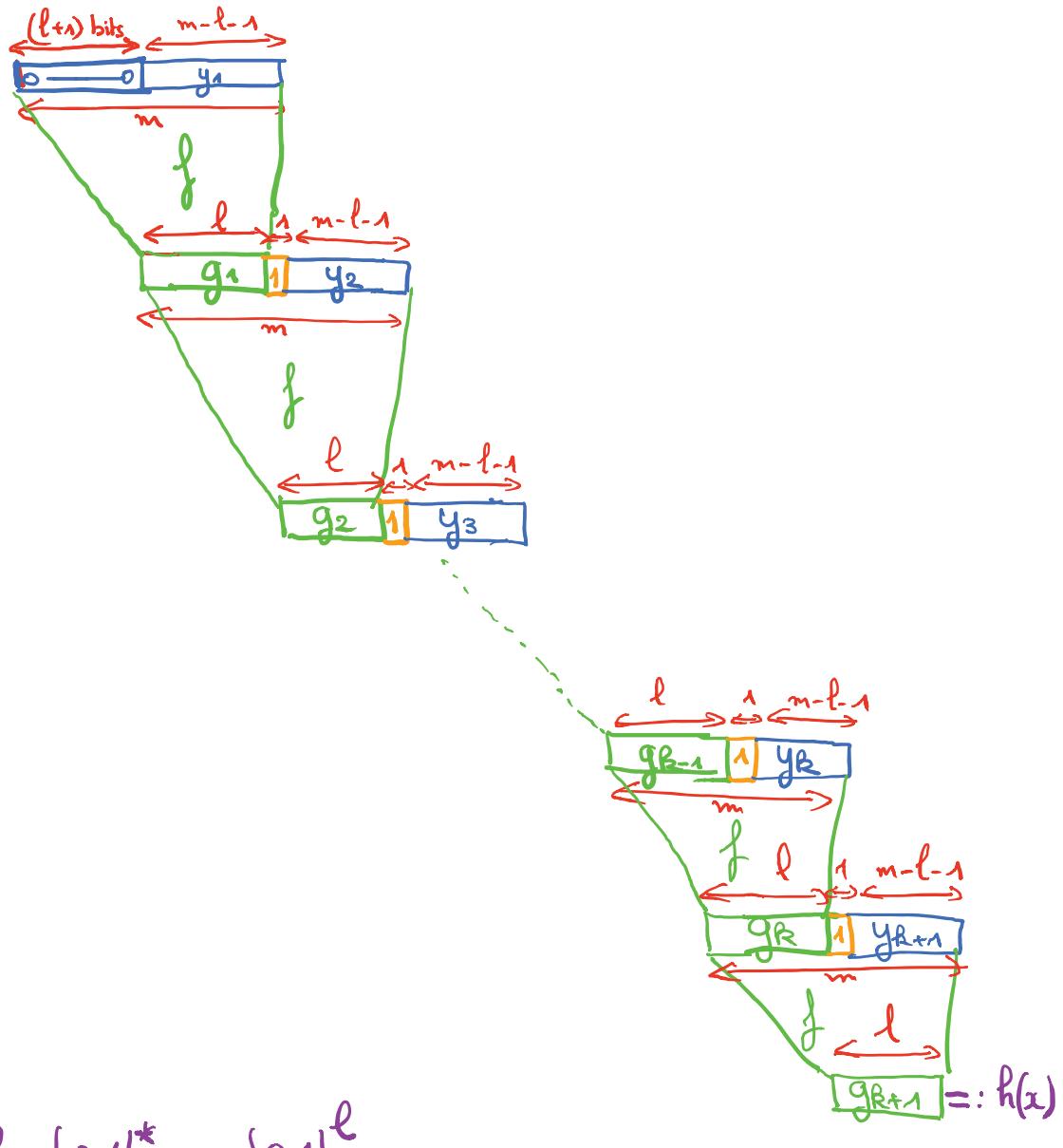
$$y = y_1 \parallel y_2 \parallel y_3 \parallel \dots \parallel y_{k-1} \parallel y_k \parallel y_{k+1}$$

$\overbrace{\hspace{10em}}$ $\overset{(m-l-1) \text{ bits}}{\curvearrowright}$ $\overbrace{\hspace{10em}}$ $\overset{(m-l-1) \text{ bits}}{\curvearrowright}$

$(=x_1) \quad (=x_2) \quad (=x_3) \quad (=x_{k-1}) \quad (=x_k \parallel 0 \dots 0)$

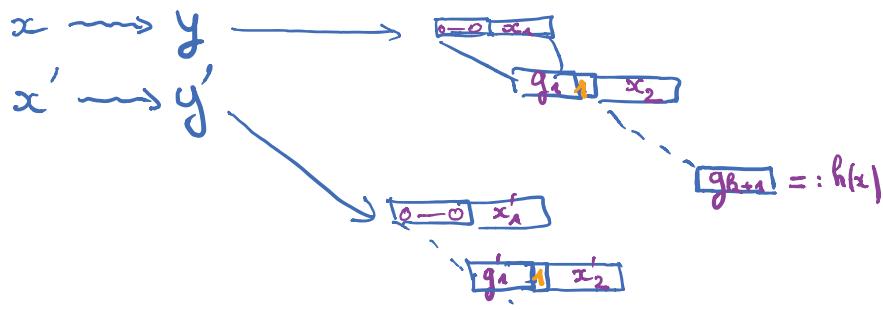
$\underbrace{\hspace{10em}}_{\text{contient la valeur } d \text{ (écrite en bininaire)}}$

$\underbrace{\hspace{10em}}_{\text{padding}}$



Théorème [Merkle, Damgård] : Si f est à collisions fortes difficiles alors h aussi

Démonstration : Supposons que h n'est pas à collisions fortes difficiles i.e. on peut trouver x et x' tels que $x \neq x'$ et $h(x) = h(x')$



$$\begin{aligned}
 h(x) = h(x') &\iff g_{R+1} = g'_{t+1} \\
 &\iff f(g_R \parallel 1 \parallel y_{R+1}) = f(g'_t \parallel 1 \parallel y'_{t+1}) \quad (*)
 \end{aligned}$$

- Si $y_{R+1} \neq y'_{t+1}$ \rightarrow collision pour f
- Si $g_R \neq g'_t$ \rightarrow collision pour f
- Si $g_R = g'_t$ et $y_{R+1} = y'_{t+1}$

$$f(g_{R-1} \parallel 1 \parallel y_R) = f(g'_{t-1} \parallel 1 \parallel y'_t) \quad (**)$$

- si $y_R \neq y'_t$ ou $g_{R-1} \neq g'_{t-1}$ \rightarrow collision pour f
- si $y_R = y'_t$ et $g_{R-1} = g'_{t-1}$

$$f(g_{R-2} \parallel 1 \parallel y_{R-1}) = f(g'_{t-2} \parallel 1 \parallel y'_{t-1}) \quad \text{etc}$$

Quand le naissonnement s'arrête-t-il ?

1^e cas : on trouve une relation de la forme

$$f(g_{k-i} \parallel 1 \parallel y_{k-i+1}) = f(g'_{t-i} \parallel 1 \parallel y'_{t-i+1})$$

avec $\begin{cases} y_{k-i+1} \neq y'_{t-i+1} \\ \text{ou } g_{k-i} \neq g'_{t-i} \end{cases}$

→ collision sur f \square

2^e cas : on aboutit à :

$$(i=k) \quad f(0 \longrightarrow 0 \parallel y_1) = f(g'_{t-k} \parallel 1 \parallel y'_{t-k+1})$$

avec $\boxed{k < t}$.
 nb de blocs de x'
 nb de blocs de x

⇒ collision sur f \square

3^e cas : on aboutit à :

$$(i=t) \quad f(g_{k-t} \parallel 1 \parallel y_{k-t+1}) = f(0 \longrightarrow 0 \parallel y'_1)$$

avec $\boxed{k > t}$

⇒ collision sur f \square

4^e cas : on aboutit à :

$$\text{avec } \boxed{k=t} \quad f(0 \longrightarrow 0 \parallel y_1) = f(0 \longrightarrow 0 \parallel y'_1)$$

- si $y_1 \neq y'_1 \rightarrow$ collision sur f \square

- si $y_1 = y'_1 \rightarrow$ comme on avait au préalable : $\begin{cases} y_2 = y'_2 \\ y_3 = y'_3 \\ \vdots \\ y_k = y'_k \end{cases}$

$\Rightarrow y = y' \Rightarrow x = x'$
 contradiction

Rémarque : On peut appliquer une construction du même type
à la fonction de compression de Chauvin - Van Heijst -
Pfitzmann pour obtenir une fonction de charge

ch 2 : L'algorithme RSA en pratique

I] Rappels sur RSA

II] RSA en signature

① Problématique

② Fonctions de hachage

③ Exemple de la fonction de hachage
de Chaum-van Heijst-Pfitzmann

④ Construction de Merkle-Damgård

Fonction de compression: $f: \{0,1\}^m \rightarrow \{0,1\}^l$ avec $m \geq l+2$ (m, l fixés)

Fonction de hachage: $h: \{0,1\}^* \rightarrow \{0,1\}^l$ (l fixé)

Attaque de Bleichenbacher

Hypothèse : h fonction de hachage utilisant la construction
de Merkle-Damgård

. On dispose d'une collision: $M \in \{0,1\}^*$, $M' \in \{0,1\}^*$
tels que $M \neq M'$ et $h(M) = h(M')$

Objectif : L'attaquant choisit deux textes T et T'

(ex: $T =$ "Moi Alice je donne 1000 € à Bob"
 $T' =$ "Moi Alice je donne 10000 € à Bob")

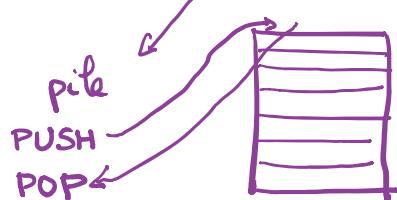
pb: Construire deux fichiers F et F' tels que

- { • F affiche le texte T
- { • F' affiche le texte T'

et
$$h(F) = h(F')$$

Exemple (Bleichenbacher): F et F' sont des fichiers de type postscript ($.ps$)

- F et F' sont en pdf

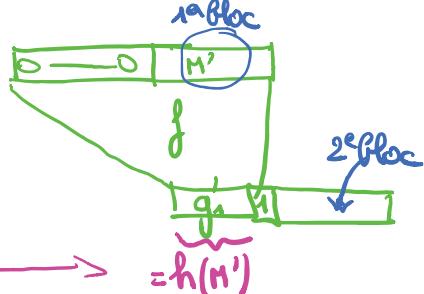
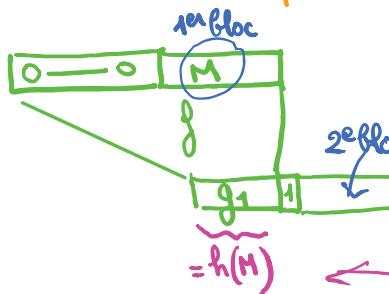


$M \rightarrow \text{pile}$
 $A \leftarrow \text{pile}$
If ($A = M$) Then
 Affiche (T)
else
 Affiche (T')

Fichier F

$M' \rightarrow \text{pile}$
 $A \leftarrow \text{pile}$
If ($A = M$) Then
 Affiche (T)
else
 Affiche (T')

Fichier F'



si $g_1 = g'_1$ (collision entre M et M')

alors on aura $g_2 = g'_2$, puis $g_3 = g'_3$, ..., et donc $h(F) = h(F')$

Consequence : Dès qu'une collision est trouvée sur une fonction de hachage, il est indispensable de cesser de l'utiliser.

Remarque : F et F' peuvent être des programmes exécutables
 ↳ exemples : { mise à jour de windows } (F)
 { ou virus } (F')

⑤ Fonctions de hachage usuelles

	Nom	Année	Inventeurs	l (nb de bits de sortie)	attaque anniversaire	meilleure attaque connue
Construction de Merkle-Damgård	MD4 <small>Message Digest</small>	1990	Rivest	128	2^{64}	1995 : Dobbertin → collision
	MD5	1991	Rivest	128	2^{64}	2005 : Wang → collisions en 2^{24}
	SHA-0 <small>Secure Hash Algorithm</small>	1993	NIST	160	2^{80}	2004 : Joux → collision en 2^{51}
	SHA-1	1994	NIST	160	2^{80}	2006 : Wang → algo en 2^{69} → 2016 : collision (Stevens, ...)
	SHA-256	2002	NIST	256	2^{128}	
	SHA-384 = Famille SHA-2			384	2^{192}	
construction "éponge"	SHA-512			512	2^{256}	
	SHA-3	2011 (concours SHA-3)	G. Bertoni, J. Daemen, M. Peeters, G. van Assche (KECCAK)	224	2^{112}	
				256	2^{128}	
				384	2^{192}	
				512	2^{256}	

⑤ Standards de signature RSA

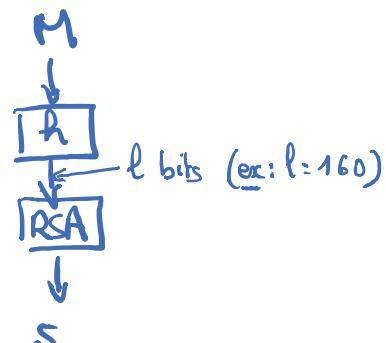
- Standard PKCS #1 v1.5

Public key
Cryptographic Standard
(publié par RSA Data Security)

$$S = (\mu(M))^d \bmod n$$

avec $\mu(M) = \underbrace{00}_{\text{octet}} \underbrace{01}_{\text{octet}} \underbrace{FF \dots FF}_{\text{octet}} \underbrace{00}_{\text{octet}} \parallel \underbrace{G_x}_{\substack{\text{padding} \\ \text{octet}}} \parallel h(M)$

1024 bits = 128 octets



Alice $\xrightarrow{M, S}$ Bob

• calcule

$$S^e \bmod n = \underbrace{00 \ 01 \ FF \dots FF \ 00}_{\substack{1024 \text{ bits} \\ \text{octet}}} \parallel G_x \parallel h(M)$$

\downarrow
 h

Remarque : Avec PKCS#1 v1.5, les seules attaques courantes sont :

- soit trouver une collision pour h (casser la fonction de hachage)
- soit factoriser n (casser la primitive RSA)

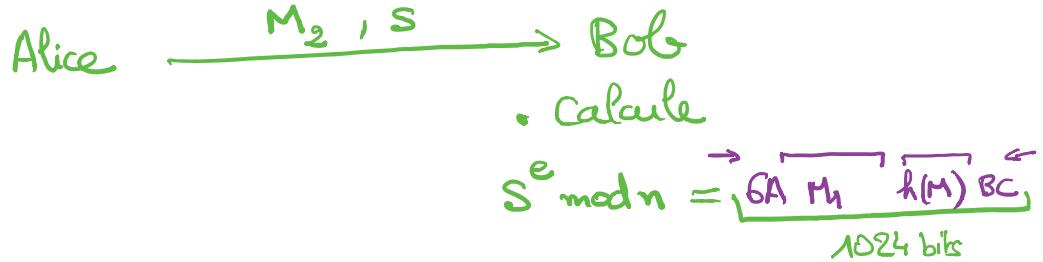
- Standard ISO 9796-2

International Standard Organization

$$S = (\mu(M))^d \bmod n$$

avec $\mu(M) = \underbrace{6A}_{\text{octet}} \parallel \underbrace{M_1}_{\substack{\text{hache de } M \\ \text{octet}}} \parallel h(M) \parallel \underbrace{BC}_{\text{octet}}$

$$M = \underline{M_1} \parallel M_2$$



Remarque : La signature ISO 9796-2 s'appelle un algorithme de signature "avec message recovery"

Remarque : le standard PKCS #1 v2.1 utilise la méthode PSS (Probabilistic Signature Scheme) qui fait intervenir de l'aléa, et est probable siue si la fonction de hachage est à collisions fates difficiles et si la fonction RSA est à sens unique.

III] RSA en chiffrement

① Théorème de Coppersmith

Remarque : résolution de systèmes d'équations linéaires

$\rightarrow O(n^3)$: pivot de Gauss

$O(n^{\log_2 7})$: Strassen

$O(n^{2.3755\dots})$: Coppersmith

Coppersmith a participé à la mise au point du DES en 1976

Théorème (Coppersmith, 1997) : N entier, $f \in \mathbb{Z}[x]$ polynôme unitaire de degré d

$$B = N^{\frac{1}{d} - \varepsilon} \quad (\varepsilon > 0)$$

Alors étant donné N et f , on peut trouver efficacement tous les entiers x_0 tels que $\begin{cases} |x_0| < B \\ f(x_0) = 0 \pmod{N} \end{cases}$

Remarque : Supposons qu'on utilise RSA pour chiffrer des messages, de la façon suivante : $C = M^e \pmod{N}$

Alors si $M < \sqrt[e]{N}$, alors $C = \underbrace{M^e}_{< N} \pmod{N}$

(cela correspond au th. de Coppersmith avec $f(x) = \underbrace{x^e - C}_{= 0}$) et donc $M = \sqrt[e]{C}$

Remarque : Si N est premier, il est "facile" de trouver x_0 tel que $|x_0| < B$ et $f(x_0) = 0 \pmod{N}$

En effet, il est "facile" de néanmoins trouver les racines d'un polynôme dans le corps \mathbb{F}_p

On peut par exemple calculer $\text{pgcd}(f(x), X^p - X)$
 ↳ algorithme de Berlekamp
 (en temps polynomial)

Démonstration du th. de Coppersmith

Définition : pour $h(x) = \sum a_i x^i \in \mathbb{Z}[x]$
 on note $\|h\|^2 = \sum |a_i|^2$

lemme : Pour $h(x) \in \mathbb{Z}[x]$ de degré d , et $B > 0$ entier
 supposons $\|h(B.)\| < \frac{N}{\sqrt{d+1}}$

Si $|x_0| < B$ satisfait $h(x_0) = 0 \pmod{N}$
 alors $h(x_0) = 0$

Preuve : $|h(x_0)| = \left| \sum_{i=0}^d a_i x_0^i \right| = \left| \sum_{i=0}^d a_i B^i \left(\frac{x_0}{B}\right)^i \right|$
 $\leq \sum_{i=0}^d |a_i B^i \left(\frac{x_0}{B}\right)^i| \leq \sum_{i=0}^d 1 \cdot |a_i B^i|$
 $\leq \sqrt{d+1} \|h(B.)\| < N$

\uparrow
inégalité
de Cauchy-Schwarz

$h(x_0) = 0 \pmod{N}$
 $|h(x_0)| < N$

Remarque

$$h(Bx) : x \mapsto h(Bx)$$

$$\|h(B.)\| < \frac{N}{\sqrt{d+1}}$$

$$\begin{aligned} h : x &\mapsto \sum a_i x^i \\ h(Bx) : x &\mapsto \sum a_i (Bx)^i \\ &= \sum a_i B^i x^i \end{aligned}$$

Idee de Coppersmith : considérer la famille de polynômes

$$g_{u,v}(x) := N^{m-v} x^u f(x)^v$$

si x_0 est racine de $f \pmod{N^m}$, alors x_0 est racine de $g_{u,v} \pmod{N^m}$
 et aussi des combinaisons linéaires des $g_{u,v}$

On considère la famille des $g_{u,v}$ pour $\begin{cases} u = 0, 1, \dots, d-1 \\ v = 0, \dots, m \end{cases}$

Pb: trouver une combinaison linéaire (à coef. entiers) de ces $g_{u,v}$ [telle que la norme est "petite"] (au sens du lemme)

à déterminer
par la suite

Réseaux euclidiens:

Définition: Soit $u_1, u_2, \dots, u_w \in \mathbb{Z}^w$ vecteurs linéairement indépendants

on appelle réseau L engendré par (u_1, \dots, u_w)
l'ensemble des combinaisons linéaires à coefficients entiers de u_1, u_2, \dots, u_w .

Définition: Déterminant de L = déterminant de la matrice $w \times w$ dont les lignes sont les vecteurs u_1, \dots, u_w

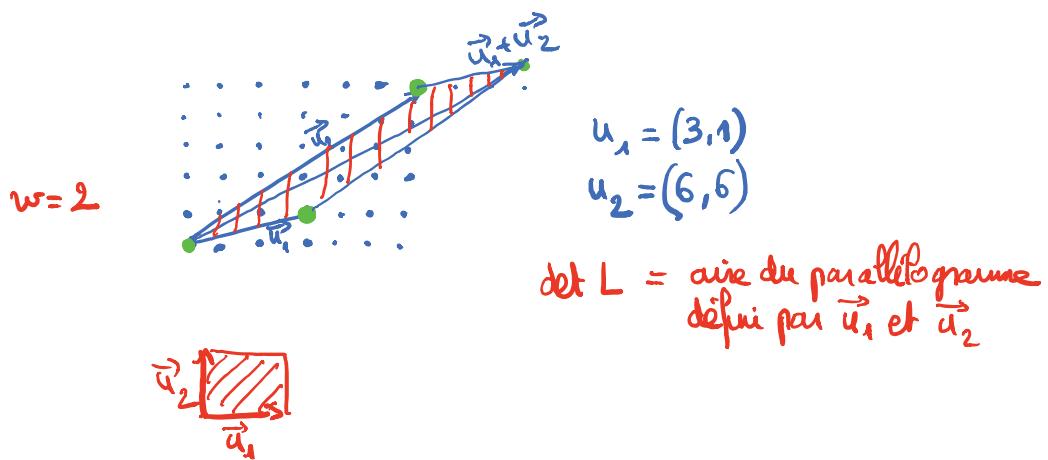
Th (Hermite): Tout réseau L de dimension w contient un vecteur $v \in L \setminus \{0\}$ dont la norme vérifie $\|v\| \leq \gamma_w (\det L)^{1/w}$

constante
dépendant
uniquement de w

→ Algorithme LLL (Lovasz, A. Lenstra, H. Lenstra, 1982)

Soit L un réseau engendré par (u_1, \dots, u_w) .
Lorsque (u_1, \dots, u_w) sont donnés en entrée de l'algorithme, celui-ci sort un vecteur $v \in L \setminus \{0\}$

tel que $\|v\| \leq (2^{w/4}) (\det L)^{1/w}$
 et le temps de calcul est quartique en la longueur
 de l'entrée



ch 2 : L'algorithme RSA en pratique

I) Rappels sur RSA

II) RSA en signature

III) RSA en chiffrement

① Théorème de Coppersmith

Th (Coppersmith, 1997) : N entier, $f \in \mathbb{Z}[x]$ polynôme unitaire de degré d

$$B = N^{\frac{1}{d} - \varepsilon} \quad (\varepsilon > 0)$$

on peut trouver efficacement tous les entiers x_0 tels que $|x_0| < B$ et $f(x_0) \equiv 0 \pmod{N}$

Lemma : $\begin{cases} \text{pour } h(x) \in \mathbb{Z}[x] \text{ de degré } d, \quad B > 0 \text{ entier} \\ \text{si } \|h(B)\| < \frac{N}{\sqrt{d+1}} \quad \text{et si } |x_0| < B \\ \quad \text{satisfait } h(x_0) \equiv 0 \pmod{N} \\ \quad \text{alors } h(x_0) = 0 \end{cases}$

$$g_{u,v}(x) := N^{m-v} x^u f(x)^v \quad \text{pour } \begin{cases} u = 0, 1, \dots, d-1 \\ v = 0, \dots, m \end{cases}$$

$\left[\begin{array}{l} \text{si } f(x_0) \equiv 0 \pmod{N} \\ \quad \text{alors } g_{u,v}(x_0) \equiv 0 \pmod{N^m} \end{array} \right]$
idem pour les combinaisons linéaires des coefficients entiers des $g_{u,v}$

Algorithm LLL (Lovasz, A. Lenstra, H. Lenstra, 1982)

Soit L un réseau engendré par (u_1, \dots, u_w)

On peut trouver en temps polynomial un vecteur

$v \in L \setminus \{0\}$ tel que $\|v\| \leq 2^{w/4} (\det L)^{1/w}$

exemple : $m=3, d=2$

$$g_{u,v} \quad \begin{cases} u = 0, \dots, d-1 \\ v = 0, \dots, m \end{cases}$$

$$g_{u,v}(x) = N^{\frac{m-v}{2}} \underbrace{x^u f(x)^v}_{d^{\text{max}} = 7}$$

	1	x	x^2	x^3	x^4	x^5	x^6	x^7
$g_{0,0}(Bx)$	N^3							
$g_{1,0}(Bx)$		BN^3						
$g_{0,1}(Bx)$			B^2N^2					
$g_{1,1}(Bx)$				B^3N^2				
$g_{0,2}(Bx)$					B^4N			
$g_{1,2}(Bx)$						B^5N		
$g_{0,3}(Bx)$							B^6	
$g_{1,3}(Bx)$								B^7

$$g_{u,v}(Bx) = N^{\frac{m-v}{2}} (Bx)^u f(Bx)^v$$

$$\Rightarrow \det L = N^3 \times BN^3 \times B^2N^2 \times B^3N^2 \times B^4N \times B^5N \times B^6 \times B^7$$

Plus généralement, on obtient

$$\det L = \prod_{u=0}^{d-1} \prod_{v=0}^m N^{m-v} B^{vd+u}$$

On veut trouver m tel que

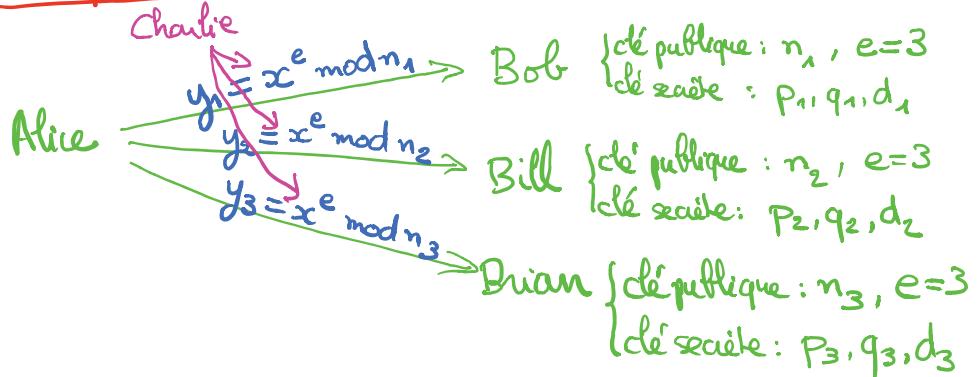
$$\|v\| < \underbrace{2^{w/4} (\det L)^{1/w}}_{\text{donné par LLL}} < \frac{N^m}{\sqrt{w}} \quad \text{où } w=d(m+1)$$

$$\begin{aligned} \det L &= \prod_{u=0}^{d-1} \prod_{v=0}^m N^{m-v} B^{vd+u} \\ &= \prod_{u=0}^{d-1} B^{(m+1)u} \underbrace{\prod_{v=0}^m N^{m-v} B^{vd}}_{B^{dm}} \\ &= N^{\frac{m(m+1)}{2}d} \prod_{u=0}^{d-1} B^{(m+1)u} \underbrace{\prod_{v=0}^m B^{vd}}_{B^{\frac{d(m+1)}{2}}} \\ &= N^{\frac{m(m+1)}{2}d} B^{d^2 \frac{m(m+1)}{2}} \prod_{u=0}^{d-1} B^{(m+1)u} \\ &= B^{\frac{d(d-1)}{2}(m+1)} \\ \Rightarrow \det L &= N^{\frac{d m(m+1)}{2}} B^{\frac{d(m+1)}{2} (dm + (d-1))} \\ \det L &= N^{\frac{d m(m+1)}{2}} B^{\frac{(d(m+1)-1)d(m+1)}{2}} \end{aligned}$$

$$\text{On veut : } 2^{w/4} (\det L)^{1/w} < \frac{N^m}{\sqrt{w}} \quad \text{avec } w=d(m+1)$$

$$\begin{aligned}
 (\det L)^{\frac{1}{\lambda w}} &= N^{\frac{m}{2}} B^{\frac{d(m+1)-1}{2}} \\
 2^{\frac{d(m+1)}{4}} N^{\frac{m}{2}} B^{\frac{d(m+1)-1}{2}} &< \frac{N^m}{\sqrt{d(m+1)}} \\
 \Leftrightarrow B &< \underbrace{\alpha(d,m) N^{\frac{m}{d(m+1)-1}}}_{B < \underbrace{\alpha(d,m) N^{\frac{1}{d} - \Theta(\frac{1}{m})}}_{N^{\frac{1}{d} - \varepsilon} \text{ pour } m \text{ assez grand}}}
 \end{aligned}$$

② Attaque de Hastad (1988)



Charlie connaît $y_1 = x^3 \pmod{n_1}$
 $y_2 = x^3 \pmod{n_2}$

$\left \begin{array}{l} \text{Th Chinois} \\ \text{si pgcd}(n_1, n_2) = 1 \end{array} \right. \rightarrow x^3 \pmod{(n_1 n_2)}$
--

Charlie connaît $x^3 \pmod{(n_1 n_2)}$
 $y_3 = x^3 \pmod{n_3}$

$\left \begin{array}{l} \text{Th Chinois} \\ \text{si pgcd}(n_1, n_2, n_3) = 1 \end{array} \right. \rightarrow x^3 \pmod{(n_1 n_2 n_3)}$

Sachant que $\begin{cases} 0 \leq x < n_1 \\ 0 \leq x < n_2 \\ 0 \leq x < n_3 \end{cases} \Rightarrow 0 \leq x^3 < n_1 n_2 n_3$

\Rightarrow Charlie connaît $x^3 \Rightarrow$ il connaît x

Parade possible: on suppose que chaque destinataire possède (en plus de sa clé publique RSA) un polynôme public $f_i \in \mathbb{Z}/n_i \mathbb{Z}[x]$

Chiffrement à destination de l'utilisateur n° i :

$$c_i = f(M)^{e_i} \pmod{N_i} \quad (1 \leq i \leq k)$$

Théorème [attaque de Hastad améliorée]

- N_1, N_2, \dots, N_k entiers premiers entre eux 2 à 2
- On pose $N_{\min} = \min_{1 \leq i \leq k} N_i$
- $g_i \in \mathbb{Z}/N_i \mathbb{Z}[x] \quad (1 \leq i \leq k)$: k polynômes de degré maximum d

S'il existe un unique $M < N_{\min}$ tel que

$$\forall i, 1 \leq i \leq k, \quad g_i(M) = 0 \pmod{N_i}$$

et si $k \geq d$, alors on peut trouver M en temps polynomial

Remarque: on peut prendre $g_i = \underbrace{(f_i)^{e_i}}_{=0 \pmod{N_i} \text{ pour l'entrée } M} - c_i$

Démonstration : Soit $\bar{N} = N_1 N_2 \dots N_k$

- On peut supposer g_i unitaire

(quitte à multiplier par l'inverse du coefficient dominant,
et si le coefficient dominant n'est pas inversible
dans $\mathbb{Z}/N_i\mathbb{Z}$ → on peut factoriser N_i)

- On peut supposer que les g_i sont tous de degré d
(quitte à les multiplier par des monômes)

- On définit le polynôme

$$g(x) = \sum_{i=1}^k T_i g_i(x) \quad \text{avec } T_i = \begin{cases} 1 \pmod{N_i} \\ 0 \pmod{N_j} (j \neq i) \end{cases}$$

$\left[\begin{array}{l} g \text{ est unitaire et de degré } d \\ g(M) = 0 \pmod{\bar{N}} \end{array} \right]$

T_i est ainsi défini $\pmod{\bar{N}}$
par le th. chinois

$$\left[\begin{array}{l} M < N_{\min} \leq \bar{N}^{1/k} \leq \bar{N}^{1/d} \\ M < \bar{N}^{1/d} \Rightarrow \exists \varepsilon / M < \bar{N}^{1/d-\varepsilon} \end{array} \right]$$

$\xrightarrow{\text{th. de Coppersmith}}$ on trouve M en temps polynomial

Remarque : L'application du th. chinois des restes
se fait en temps polynomial

Si $\text{pgcd}(a,b)=1$, il y a un isomorphisme d'anneaux
entre $\mathbb{Z}/ab\mathbb{Z}$ et $\mathbb{Z}/a\mathbb{Z} \times \mathbb{Z}/b\mathbb{Z}$

En particulier $\Psi : \begin{cases} \mathbb{Z}/ab\mathbb{Z} \rightarrow \mathbb{Z}/a\mathbb{Z} \times \mathbb{Z}/b\mathbb{Z} \\ x \mapsto (x \pmod{a}, x \pmod{b}) \end{cases}$

est une bijection

Comment calculer $\Psi^{-1}(\alpha, \beta)$? (cela revient à trouver x dans $\mathbb{Z}/ab\mathbb{Z}$ tel que

- On considère les coefficients de Bezout :
$$\boxed{ua + vb = 1}$$

(que l'on peut obtenir en temps polynomial avec Euclide étendu)

- Posons $\boxed{x = ua\beta + vb\alpha}$

Alors on a bien

$$\begin{aligned} x &= ua\beta + (1-ua)\alpha \\ &= \alpha + (\beta - \alpha)ua \\ &= \alpha \bmod a \end{aligned}$$

et

$$\begin{aligned} x &= (1-vb)\beta + vb\alpha \\ &= \beta + (\alpha - \beta)vb \\ &= \beta \bmod b \end{aligned}$$

$$\boxed{\Psi^{-1}(\alpha, \beta) = ua\beta + vb\alpha}$$

③ Short pad attack

Idee : pour chiffrer un message M , on calcule

$$C = (2^m M + r)^e \bmod N$$

message clair valeur aléatoire de m bits
d'au plus $(k-m)$ bits

$$C = (M \| r)^e \bmod N$$

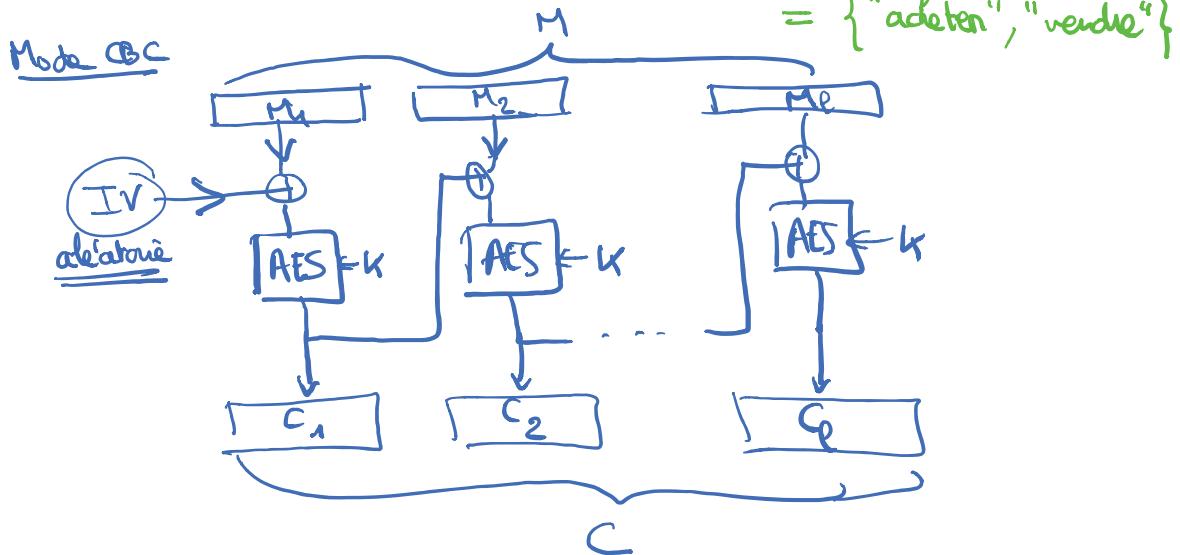
Remarque : Un "bon" algorithme de chiffrement ne peut pas être déterministe

En particulier, si on chiffre deux fois le même message, les deux chiffrés ne doivent pas révéler qu'il s'agit du même message.

(= sécurité sémantique)

ex : Espace des messages clairs = {"oui", "non"}

= {"acheter", "vendre"}



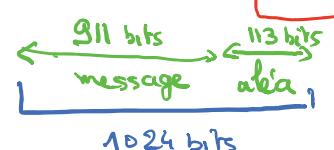
Attaque (Coppersmith) : $(N, e) = \text{clé publique RSA}$

$$m = \lfloor \frac{k}{e^2} \rfloor$$

($k = \text{nb de bits de } N$)

ex: $k = 1024$
 $e = 3$

$$m = 113$$



On suppose que l'attaquant peut récupérer deux chiffrés de M

$$\begin{cases} C_1 = M_1^e \bmod N & \text{avec } M_1 = 2^m M + r_1 \\ C_2 = M_2^e \bmod N & \text{avec } M_2 = 2^m M + r_2 \end{cases}$$

aléa
aléa

On définit

$$\begin{cases} g_1(x,y) = x^e - c_1 \\ g_2(x,y) = (x+y)^e - c_2 \end{cases}$$

Quand $y = r_2 - r_1$, alors M_1 est une racine commune aux deux polynômes

car

$$\begin{cases} g_1(M_1, r_2 - r_1) = M_1^e - c_1 = 0 \pmod{N} \\ g_2(M_1, r_2 - r_1) = \underbrace{(M_1 + r_2 - r_1)}_{M_2}^e - c_2 = 0 \pmod{N} \end{cases}$$

Soit $h(y) = \text{Res}_x(g_1, g_2)$

$$A(x) = a_{n-1}x^{n-1} + \dots + a_1x + a_0$$

$$B(x) = b_{m-1}x^{m-1} + \dots + b_1x + b_0$$

$$\text{Res}(A, B) = \left| \begin{matrix} a_{n-1} & a_0 & & & \\ a_{n-1} & a_0 & & & \\ & a_0 & & & \\ & & a_{n-1} & a_0 & \\ b_{m-1} & b_0 & & & \\ b_{m-1} & b_0 & & & \\ & b_0 & & & \end{matrix} \right|$$

$\xrightarrow[m+n]{}$

m
n

Remarque : $\text{Res}(A, A') \rightarrow$ discriminant

Dans le cadre de la short pad attack, on considère

$$h(y) = \text{Res}_x(g_1, g_2) \in \mathbb{Z}/N\mathbb{Z}[y]$$

$\Delta = r_2 - r_1$ est une racine de h

$$h(y) = \begin{vmatrix} 1 & 0 & -c_1 \\ 1 & ey & -c_1 \\ 1 & ey & -c_1 \end{vmatrix} \quad \text{with } e = \frac{2e}{\deg h}$$

$$(x+y)^e - c_2 = x^e + eyx^{e-1} + \dots - c_2$$

On peut montrer que $d^0 h \leq e^2$

$$h(y) = \det A = \sum_{\sigma \in S_{2e}} (-1)^{\varepsilon(\sigma)} \prod_{i=1}^{2e} A_{i, \sigma(i)}$$

$$\Rightarrow d^0 h \leq e^2$$

$$0 \leq n_1 < 2^m \quad \text{avec } m = \lfloor k/e^2 \rfloor$$

$$\Rightarrow 0 \leq n_1 < N^{1/e^2}$$

De même $0 \leq n_2 < N^{1/e^2}$

$$\Rightarrow |\Delta| < N^{1/e^2}$$

$$\Rightarrow |\Delta| < N^{1/d^0 h}$$

H. de Coppersmith on peut trouver Δ en temps polynomial

On sait donc que M_1 est racine de $g_1(x) = x^e - c_1$
et de $g_2(x) = (x+\Delta)^e - c_2$

\Rightarrow il suffit de calculer $\text{pgcd}(g_1, g_2)$ pour obtenir M_1



④ Exemples de standards de chiffrement

PKCS #1 v 1.5

(N, e) : clé publique RSA

Chiffrement: $C = (\mu(M))^e \text{ mod } N$
 où $\mu(M) = \underbrace{\phi_1}_{\text{octet}} \underbrace{\phi_2}_{\text{octet}} \parallel \pi \parallel \underbrace{\phi_3}_{\text{octet}} \parallel M$

padding

message

↓
aléa d'au moins 8 octets
tous non nuls

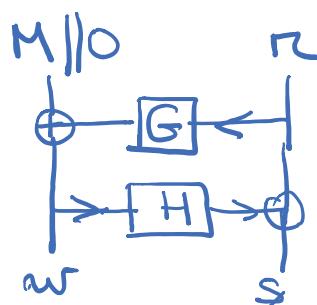
Déchiffrement: $C^d \text{ mod } N = \underbrace{\phi_1 \phi_2 \parallel \pi \parallel \phi_3}_{\substack{\text{aléa} \\ (\text{octets non nuls}) \\ (\text{au moins 8})}} \parallel M$

message

Remarque: cette version de PKCS#1
 a été cassé en 2001 (Cohen, Naccache, ...
 et F. Grieu)

Deux méthodes plus récentes

- OAEP (PKCS #1 v 2.0 , 1994)



$$C = (\mu(M, r))^e \text{ mod } N$$

avec $\mu(M, r) = w \parallel s$

- PSS-R (PKCS #1 v 2.1)

avec aussi plus de sécurité

ch 2 : le RSA en pratique

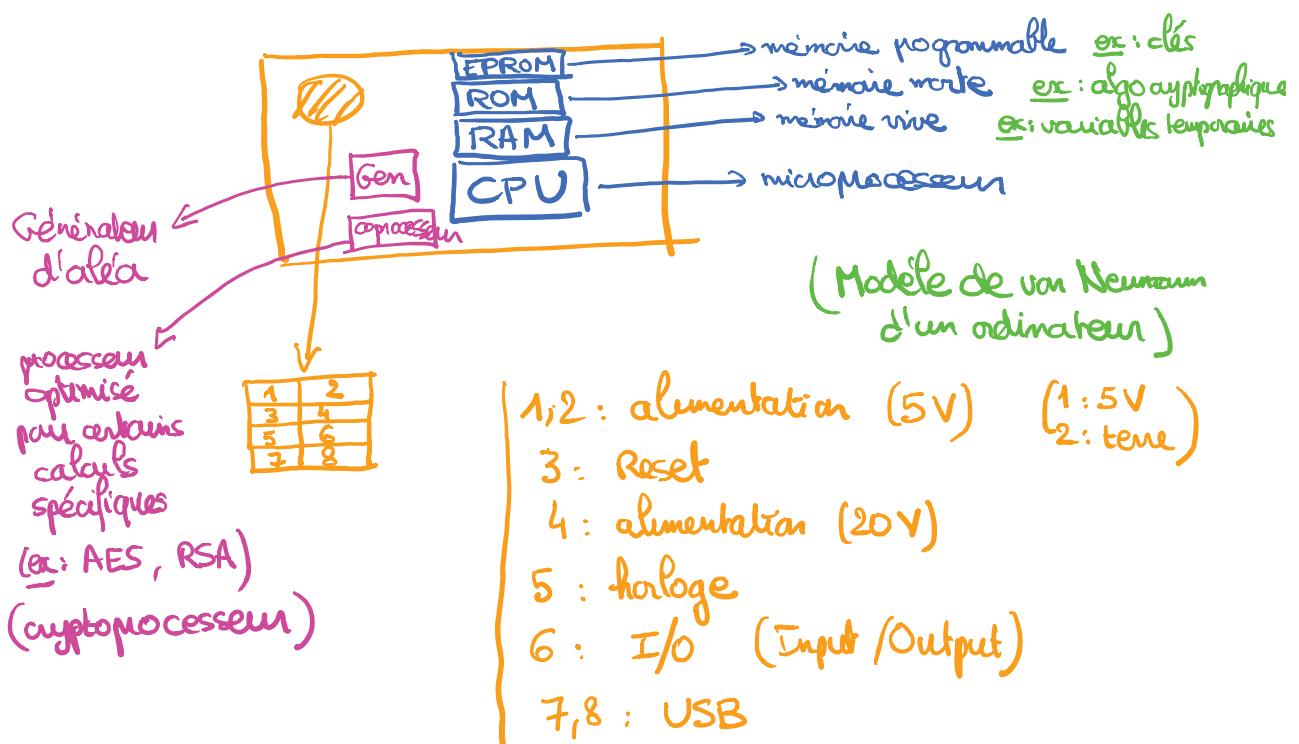
I] Rappels sur RSA

II] RSA en signature

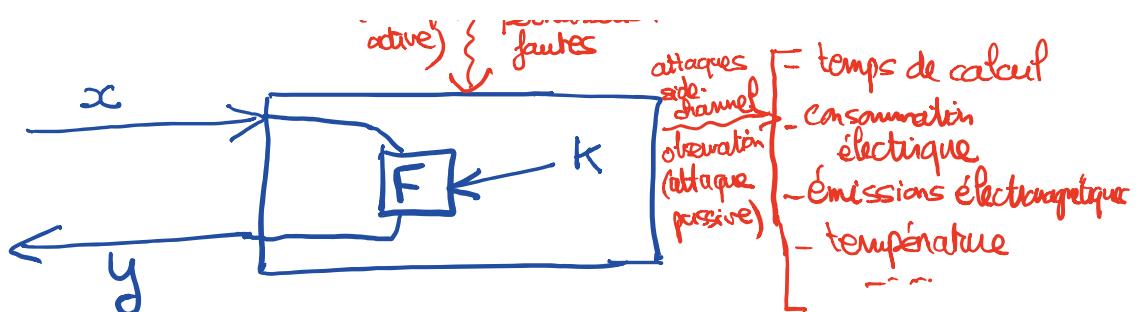
III] RSA en chiffrement

IV] Attaques physiques

① Carte à puce (carte à microprocesseur)



- perturbation de l'alimentation électrique
 - perturber l'horloge
 - champ électromagnétique (courants de Foucault)
 - lumière, laser
- (attaque) $\xrightarrow{\text{perturbation}}$

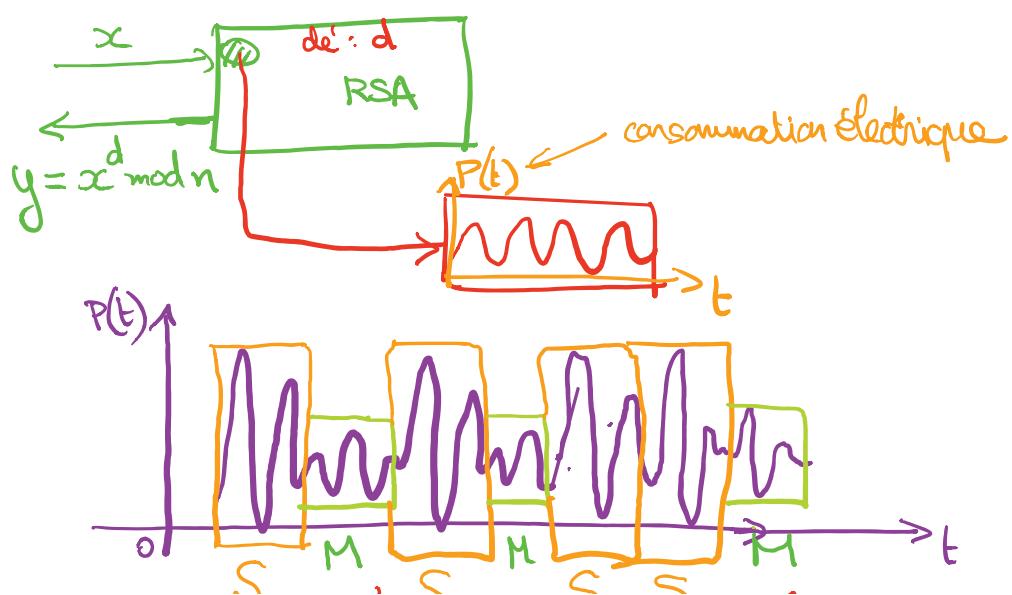


Cryptanalyse : à partir de la connaissance d'entrée x et de sorties $y = F_K(x)$, par exemple retrouver une clé K .

- *** Attaques :
 - passives : attaques side-channel (attaques par canaux auxiliaires)
 - actives : injection de fautes
attaques par perturbation

② Attaque SPA (Simple Power Analysis)

Paul Kocher (1998)



Square and multiply

$$y := x^d \bmod n$$

$$d = \sum_{i=0}^{k-1} d_i 2^i$$

↓ ième bit de d

$$y := 1$$

Pour i de $(k-1)$ à \emptyset

$$y := y^2 \bmod n$$

square

$$\text{Si } (d_i = 1) \text{ alors } y := y \times x \bmod n$$

branchement conditionnel multiply

Square and Multiply Always

$$y := x^d \bmod n$$

$$d = \sum_{i=0}^{k-1} d_i 2^i$$

$$y := 1$$

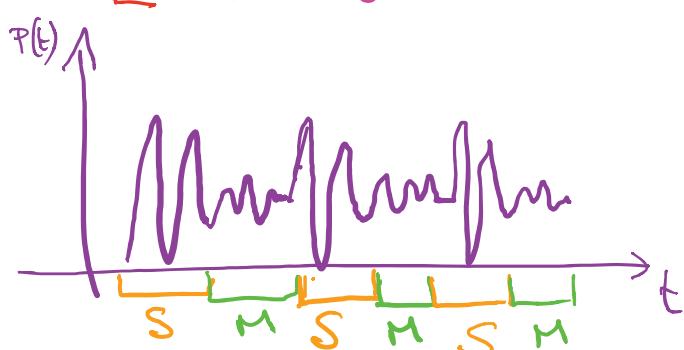
Pour i de $(k-1)$ à \emptyset

$$y_0 := y^2 \bmod n$$

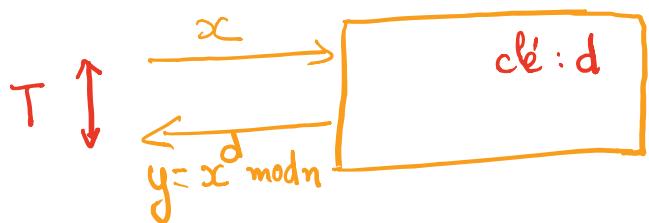
$$y_1 := y_0 \times x \bmod n$$

$$y := y_{d_i}$$

(autrement dit si $d_i = 0$, $y = y_0$
si $d_i = 1$, $y = y_1$)



③ Timing Attack



Paul Kocher (Crypto 1996)

Implementation : Square and Multiply Always

$$y = x^d \bmod n \quad d = \sum_{i=0}^{k-1} d_i \cdot 2^i$$

$y := 1$
 Pour i de $k-1$ à \emptyset
 $\left| \begin{array}{l} y_0 = y^2 \bmod n \\ y_n := \boxed{y_0 \times x \bmod n} \\ y := y d_i \end{array} \right.$

Hypothèse : La multiplication prend un temps α pour $(a \times b) \bmod n$

sous la soustraction
ou avec la soustraction

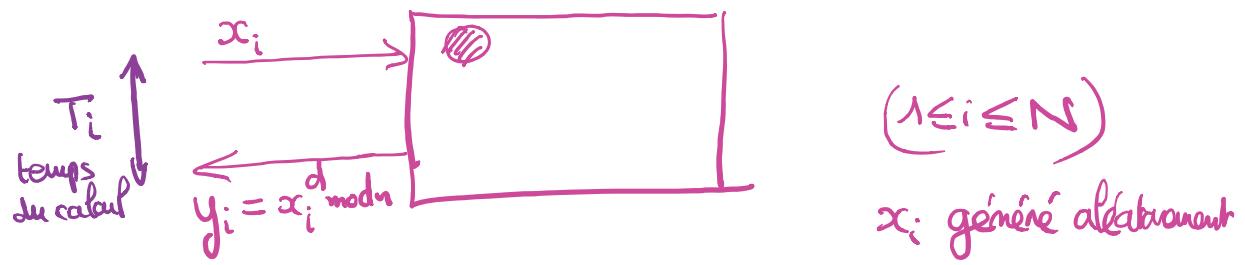
(mais pour l'algorithme de Montgomery pour $a \times b \bmod n$)

Remarque : l'algorithme de Montgomery ~~dans~~ calcule

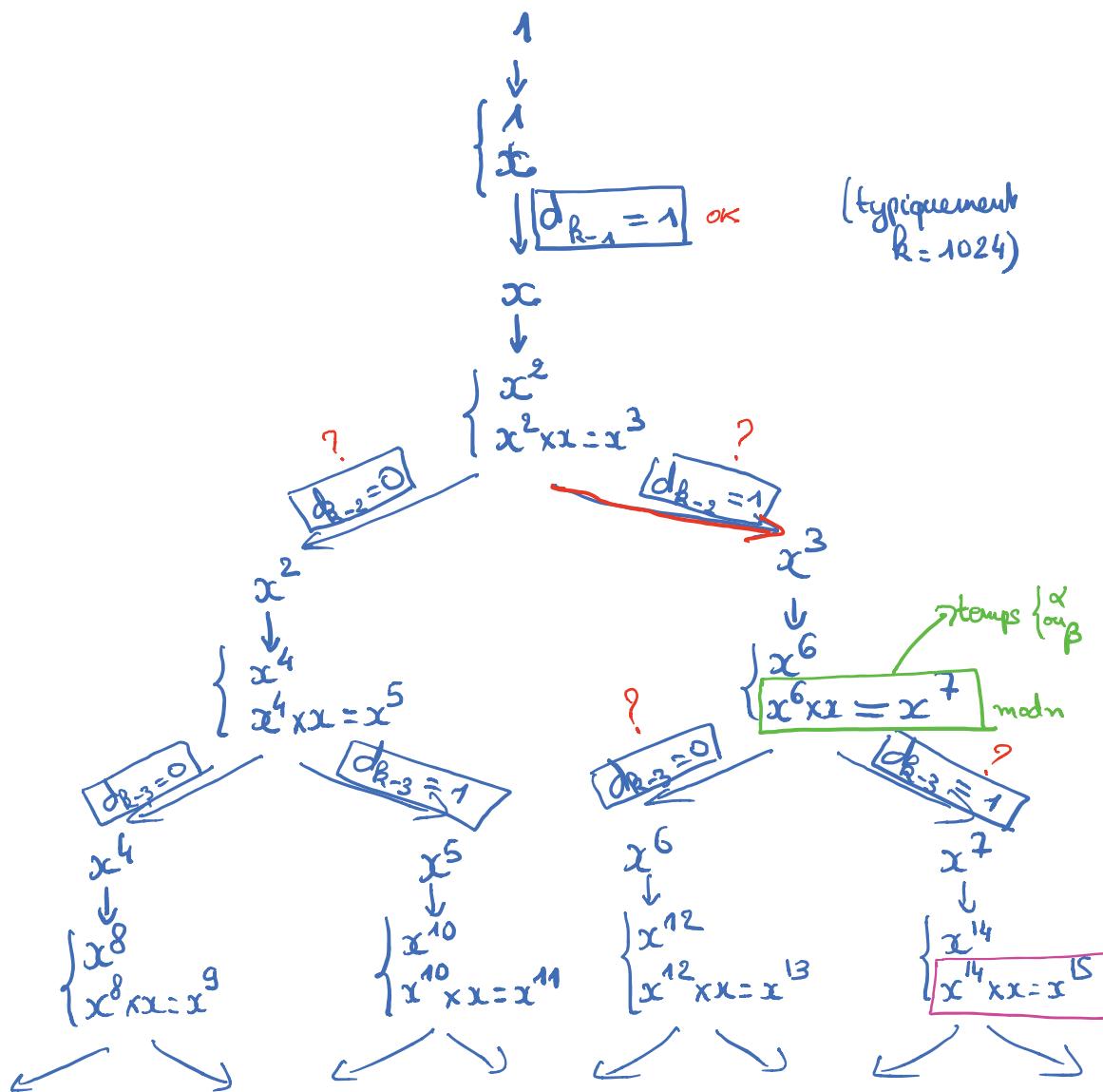
une valeur de $a \times b \bmod n$ dans $[0, 2n[$

→ si c'est dans $[0, n[\rightarrow$ OK

→ si c'est dans $[n, 2n[\rightarrow$ il faut soustraire n pour obtenir le résultat dans $[0, n[$



Square and multiply always



Idee de l'attaque: On fait l'hypothèse que $d_{k-2} = 1$
et on va déterminer si cette hypothèse était exacte
ou non

Alors on peut partitionner les entiers $(x_i)_{1 \leq i \leq N}$ en deux
sous-ensembles.

$$A = \{i, 1 \leq i \leq N \mid \underbrace{\text{temps } (x_i^6 \times x_i \bmod n)}_{\text{temp de calcul pour RSA}(x_i) = x_i^d \bmod n} = \alpha\}$$

$$B = \{i, 1 \leq i \leq N \mid \underbrace{\text{temps } (x_i^6 \times x_i \bmod n)}_{\text{temp de calcul pour RSA}(x_i) = x_i^d \bmod n} = \beta\}$$

Soit $T_A = \frac{1}{|A|} \sum_{i \in A} T_i$

temp de calcul pour $\text{RSA}(x_i) = x_i^d \bmod n$

et $T_B = \frac{1}{|B|} \sum_{i \in B} T_i$

pour $i \in A, T_i = \underbrace{\text{temps } (x_i^6 \times x_i \bmod n)}_{= \alpha} + \text{temps (autres opérations de RSA)}$

$T_i = \alpha + \underbrace{\text{temps (autres opérations de RSA)}}_{\approx \text{temps aléatoire}}$

pour $i \in B, T_i = \beta + \underbrace{\text{temps (autres opérations de RSA)}}_{\approx \text{temps aléatoire}}$

$$\Rightarrow T_A = \frac{1}{|A|} \sum_{i \in A} (\alpha + T_{\text{aléatoire}, i})$$

$$= \alpha + \boxed{\frac{1}{|A|} \sum_{i \in A} T_{\text{aléatoire}, i}}$$

$$T_{\text{constant}} + \underbrace{G_i}_{\substack{\text{distribution} \\ \text{gaussienne}}} \quad \checkmark$$

$$T_A = \alpha + T_{\text{constant}} + \underbrace{\frac{1}{|A|} \sum_{i \in A} G_i}_{O\left(\frac{1}{\sqrt{|A|}}\right)}$$

$$\boxed{T_A = \alpha + T_{\text{constant}} + O\left(\frac{1}{\sqrt{|A|}}\right)}$$

$$\boxed{T_B = \beta + T_{\text{constant}} + O\left(\frac{1}{\sqrt{|B|}}\right)}$$

$$T_B - T_A = \underbrace{\beta - \alpha}_{\text{différence des temps moyens}} + O\left(\frac{1}{\sqrt{|A|}}\right) + O\left(\frac{1}{\sqrt{|B|}}\right)$$

↓ ↓ ↓
 temps de la soustraction "supplémentaire" dans Montgomery

Remarque : si l'hypothèse $d_{k-2} = 1$ était en fait fausse

$$\text{alors } T_B - T_A = O\left(\frac{1}{\sqrt{|A|}}\right) + O\left(\frac{1}{\sqrt{|B|}}\right)$$

En Résumé :

- si $d_{k-2} = 0$, $|T_B - T_A| \approx 0$ pour $|A|$ et $|B|$ assez grands
- si $d_{k-2} = 1$, $|T_B - T_A| \approx \beta - \alpha$ pour $|A|$ et $|B|$ assez grands

Puis on recommence pour trouver successivement
 $d_{k-3}, d_{k-4}, \dots, d_0$

Remarque : L'attaque fonctionne quand

$$\beta - \alpha \gg O\left(\frac{1}{\sqrt{|A|}}\right) + O\left(\frac{1}{\sqrt{|B|}}\right)$$

c'est à dire $N \gg \frac{1}{(\beta - \alpha)^2}$

Contre-mesure : Il suffit de rendre constant le temps de calcul de la multiplication modulaire

ex: pour Montgomery, on obtient

- soit $axb \bmod n$ $\in [0, n[$	- soit $axb \bmod n$ $\in [n, 2n[$
--------------------------------------	---------------------------------------

on peut établir systématiquement la soustraction, même si elle est nulle

temps, $C_0 = \text{Montgomery}(a, b, n)$ $C_1 = C_0 - n$

Si $C_0 \geq n$ alors $j=1$ sinon $j=0$

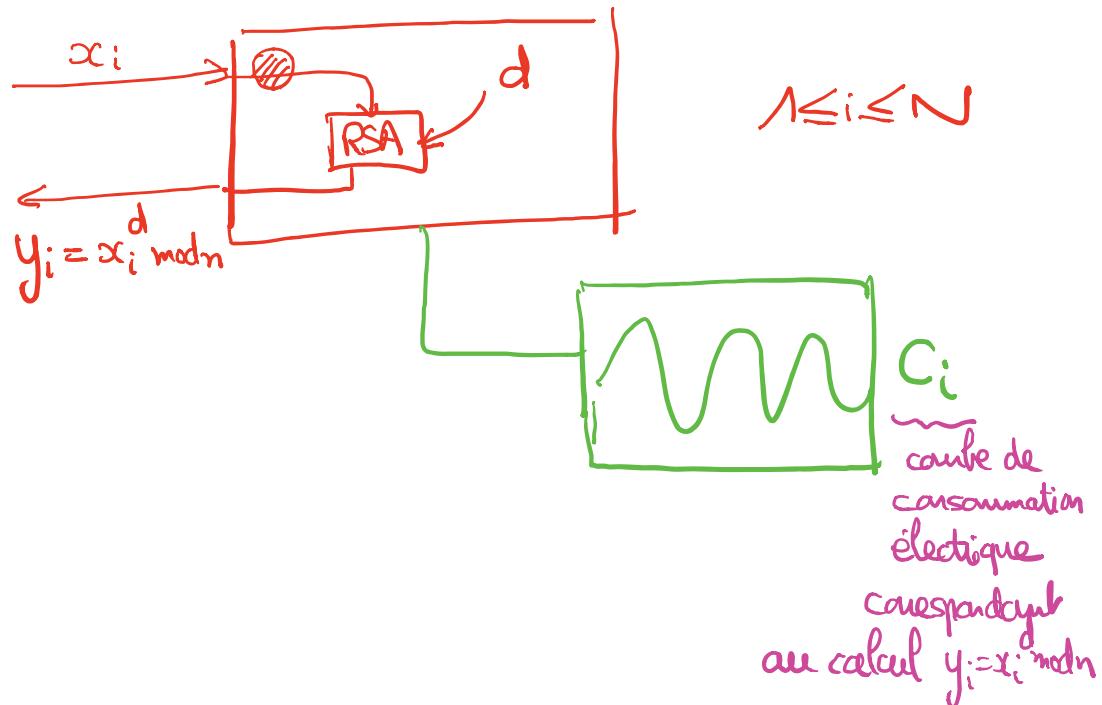
constant

$$|c = c_j$$

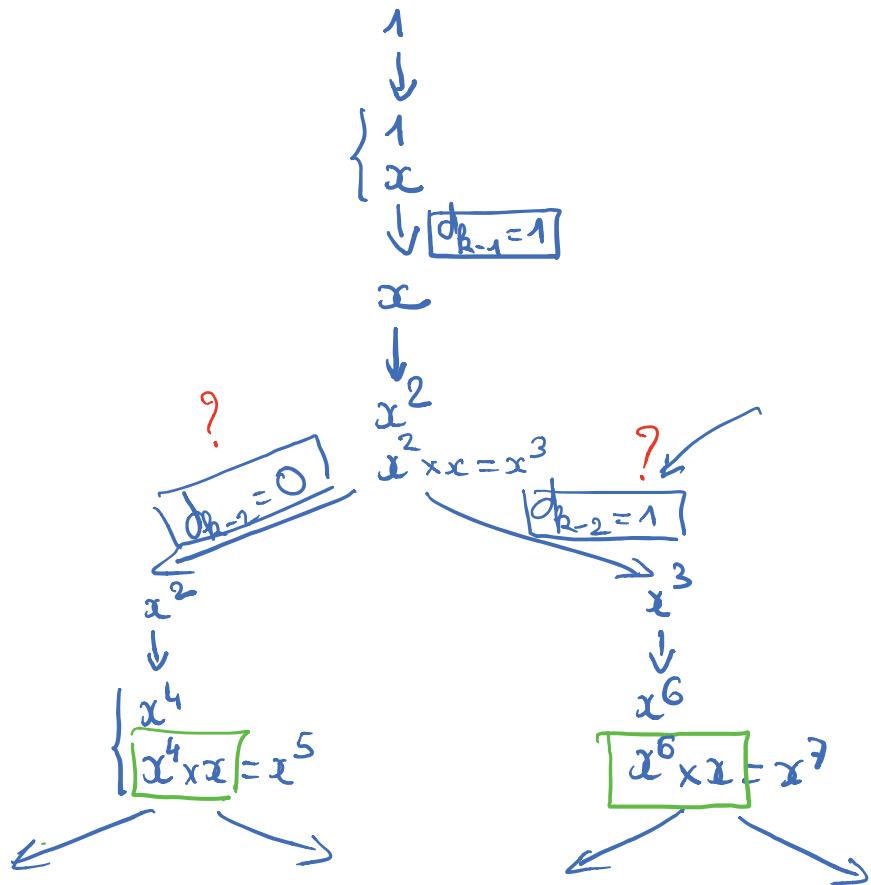
③ Attaque DPA

DPA = Differential Power Analysis
Difféentielle Analyse de la consommation électrique

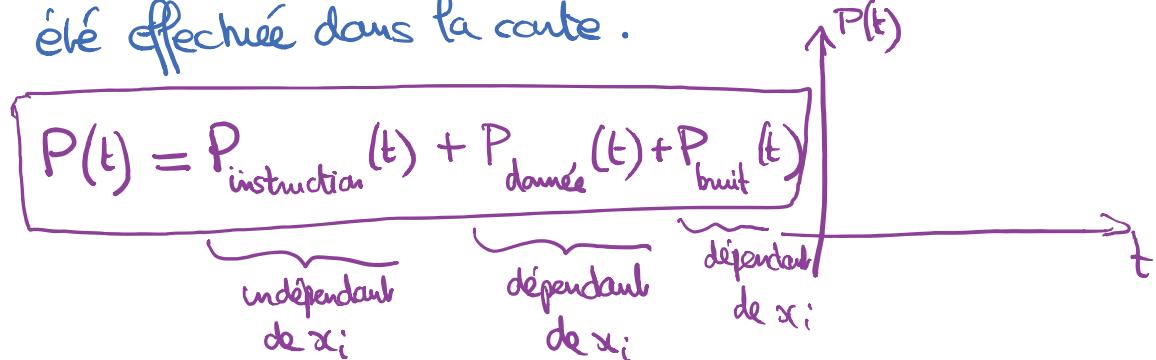
Paul Kocher (1998)



Square and Multiply Always



On cherche à savoir si l'opération $x^6 \times x$ mod n a vraiment été effectuée dans la carte.



Pour le message x_i :

$$P_i(t) = P_{\text{instruction}}(t) + P_{\text{donnée}, i}(t) + P_{\text{bruit}, i}(t)$$

Idee: On se place à l'instant t_0 où est exécuté le calcul $x_i^6 \times x_i \bmod n$, et plus précisément:

t_0 = instant où est utilisé le bit de poids fort de $x_i^6 \times x_i \bmod n$

$$A = \{i, 1 \leq i \leq N \mid \text{le bit de poids fort de } (x_i^6 \times x_i \bmod n) \text{ vaut } 0\}$$

$$B = \{i, 1 \leq i \leq N \mid \text{---} \equiv 1 \}$$

Pour $i \in A$, $P_i(t) = P_{\text{instruction}}(t) + P_{\text{donnée},i}(t)$
 $+ P_{\text{bruit},i}(t)$

en particulier

$$P_i(t_0) = P_{\text{instruction}}(t_0) + P_{\text{donnée}=0}(t_0)$$
 $+ P_{\text{bruit},i}(t_0)$

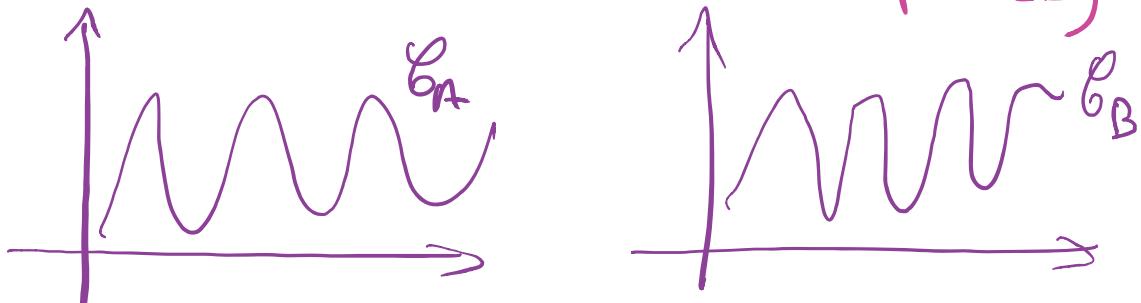
$$\Rightarrow \frac{1}{|A|} \sum_{i \in A} P_i(t_0) = P_{\text{instruction}}(t_0)$$
 $+ P_{\text{donnée}=0}(t_0)$
 $+ \frac{1}{|A|} \sum_{i \in A} \underbrace{P_{\text{bruit},i}(t_0)}_{\approx \text{gaussienne}}$

Donc $\frac{1}{|A|} \sum_{i \in A} P_i(t_0) = \frac{P_{\text{instruction}}(t_0) + P_{\text{donnée}=0}(t_0)}{+ O\left(\frac{1}{\sqrt{|A|}}\right)}$

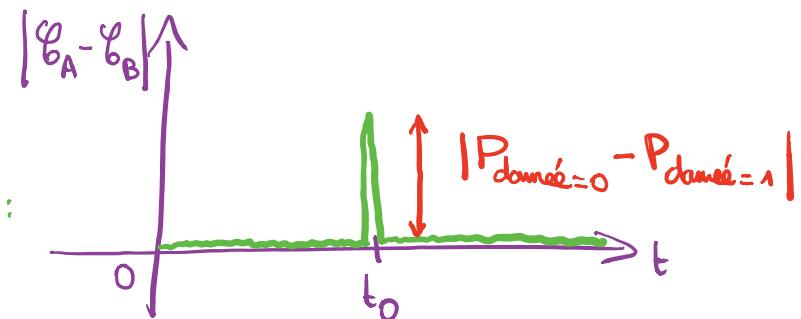
De même $\frac{1}{|B|} \sum_{i \in B} P_i(t_0) = \frac{P_{\text{instruction}}(t_0) + P_{\text{donnée}=1}(t_0)}{+ O\left(\frac{1}{\sqrt{|B|}}\right)}$

Notons $\mathcal{C}_A(t) = \frac{1}{|A|} \sum_{i \in A} P_i(t)$ (moyenne des courbes P_i pour $i \in A$)

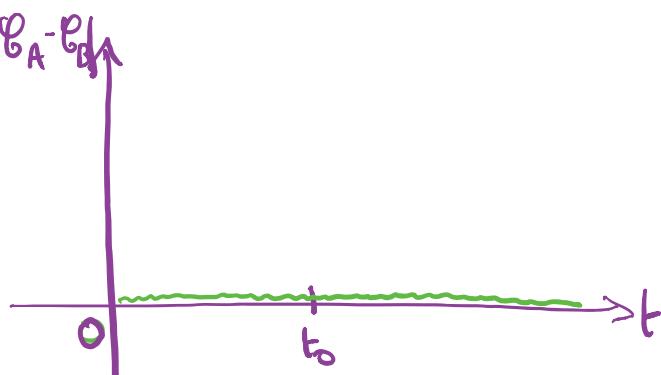
$\mathcal{C}_B(t) = \frac{1}{|B|} \sum_{i \in B} P_i(t)$ (moyenne des courbes P_i pour $i \in B$)



Si $x_i^6 \times x_i^k$ modn est nécessairement exécuté (c'est à dire si $d_{k-2} = 1$)



Si $x_i^6 \times x_i \bmod n$
 n'est pas exécuté
 (c'est à dire
 si $d_{k-2} = 0$)



Remarque : le nombre de messages nécessaires

est de l'ordre de $\frac{1}{|P_{\text{donnée}=1} - P_{\text{donnée}=0}|^2}$

différence entre
la consommation
due à un bit = 1
et celle due à un
bit = 0

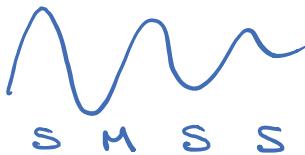
Une fois que l'on a déterminé la valeur du bit d_{k-2} ,
on réitère le même raisonnement pour trouver d_{k-3} , puis
 d_{k-4}, \dots, d_0

ch2 : Le RSA en pratique

IV] Attaques physiques

① Carte à puce

② Attaque SPA



contre-mesure: square and multiply always

③ Tuning Attacks

avec l'algorithme de Montgomery

contre-mesure: faire une implémentation à temps constant

④ Attaque DPA

$$y = x^d \bmod n$$

$$\begin{aligned}
 & 1 \\
 & \downarrow \\
 & \} 1 \\
 & \quad | \\
 & \quad 1 \times x = x \\
 & \quad \downarrow d_{k-1} = 1 \\
 & \quad x \\
 & \quad \downarrow \\
 & \quad x^2 \\
 & \quad | \\
 & \quad x^2 \times x = x^3 \\
 & \quad | \\
 & \quad d_{k-2} = 0 \\
 & \quad | \\
 & \quad (x^2)^2 = x^4 \\
 & \quad | \\
 & \quad x^4 \times x = x^5 \\
 & \quad | \\
 & \quad d_{k-2} = 1 \\
 & \quad | \\
 & \quad (x^3)^2 = x^6 \\
 & \quad | \\
 & \quad x^6 \times x = x^7 \\
 & \quad | \\
 & \quad \text{bit de poids fut}
 \end{aligned}$$



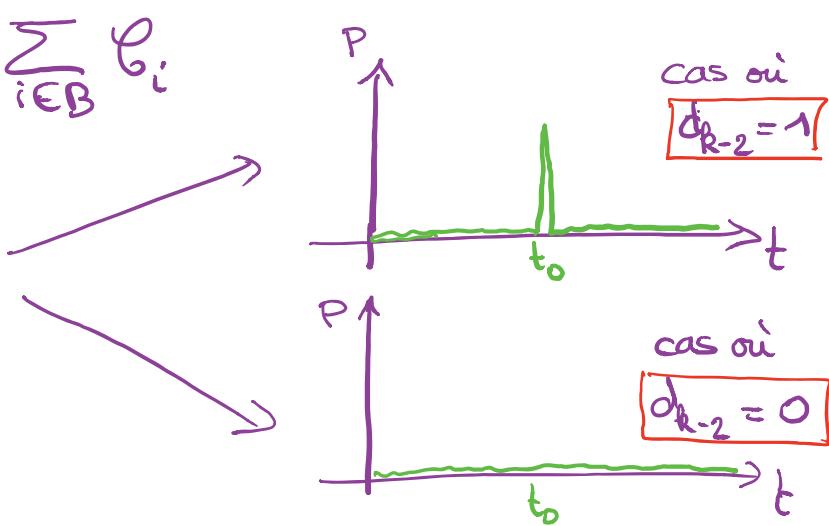
$$A = \{ i, 1 \leq i \leq N \mid \text{bit de poids fort } (x_i^6 \times x_i \bmod n) = 0 \}$$

$$B = \{ \dots \}^1 \}$$

$$f_A = \frac{1}{|A|} \sum_{i \in A} f_i$$

$$\ell_B = \frac{1}{|B|} \sum_{i \in B} \ell_i$$

$$|\phi_A - \phi_B|$$



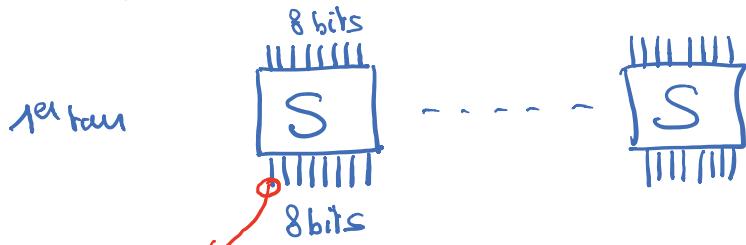
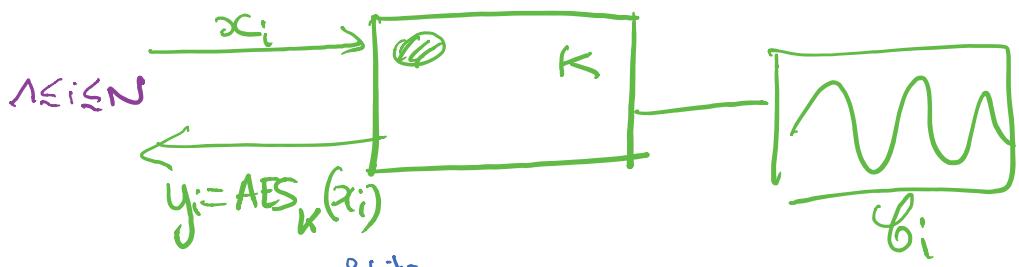
puis d_{k-3}, d_{k-4}, \dots

Qu'est-ce qui rend l'attaque possible ?

→ une variable intermédiaire du calcul (ici : le bit manipulé à l'instant t_0) ne dépend que d'un petit nombre de bits de l'accès (ici : uniquement de d_{R-2})

\Rightarrow on peut faire une recherche exhaustive sur ces bits de clé, sachant que seule la "bonne" valeur pour ces bits va donner un pic dans $|C_A - C_B|$

eric : AES



$b = 1^{\text{er}} \text{ bit de sortie de la } 1^{\text{ere}} \text{ boîte S du } 1^{\text{er}} \text{ tour de l'AES}$

b dépend des 8 bits d'entrée de cette boîte S
qui eux-mêmes ne dépendent que de 8 bits de K

On fait une hypothèse sur ces 8 bits de K .

ex: 01010011 $f(x_i)$

$$A = \{i, 1 \leq i \leq N / b_i = 0\}$$

$$B = \{i, 1 \leq i \leq N / b_i = 1\}$$

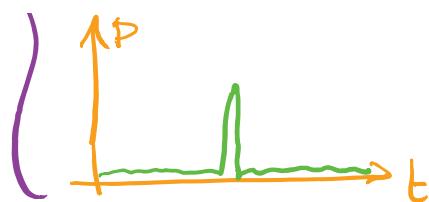
$$\mathcal{C}_A = \frac{1}{|A|} \sum_{i \in A} g_i$$

$$\mathcal{C}_B = \frac{1}{|B|} \sum_{i \in B} g_i$$

$$|\mathcal{C}_A - \mathcal{C}_B| \approx \begin{cases} P & \text{if } P \text{ is the peak value} \\ t & \text{if } t \text{ is the time step} \end{cases}$$

La définition de ces ensembles A et B dépend de l'hypothèse faite sur les 8 bits de K

pour toutes les hypothèses fausses sur les 8 bits de K



pour la bonne hypothèse
sur les 8 bits de K

Contre-mesures

- "égaliser" la consommation (par exemple en introduisant pour chaque porte logique une porte logique "symétrique")

1^{er} inconvénient : coût en consommation

2^{ème} inconvénient : effet de dipôle → champ électromagnétique qui peut quand même faire une attaque physique

↳ attaque DEMA

Electro-Magnetic

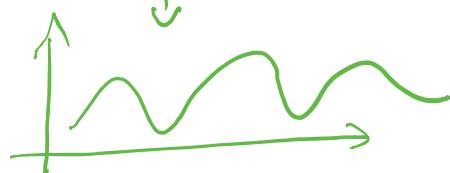
- ne faire les calculs que pour des entrées "certifiées"

→ mais l'attaque DPA peut quand même fonctionner car elle est uniquement à base connue

- "brouiller" la consommation de façon aléatoire

↳ augmente le nombre N de messages nécessaires

- ajouter des décalages temporels aléatoires



↳ augmente le nombre N de messages nécessaires

- Contre-mesures algorithmiques

Idee: Randomiser les calculs

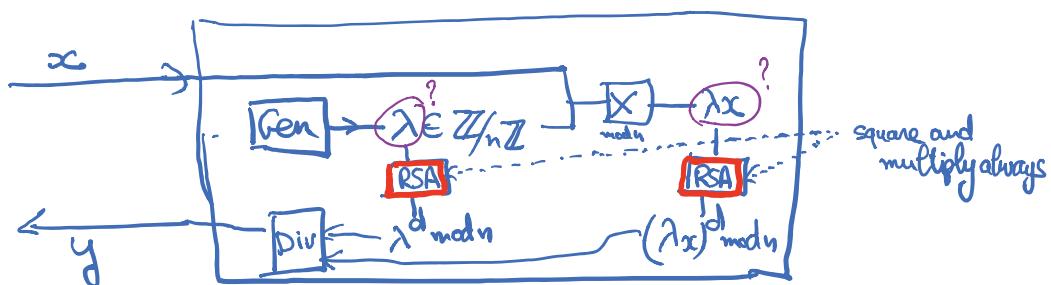
Hypothèse: certaines variables intermédiaires du calcul cryptographique qui n'ont que de l'entrée x et d'un "petit" nombre de bits de la clé sont faisables

Peut-on programmer l'algorithme cryptographique de façon à ce que cette hypothèse ne soit plus vraie

→ rendre les variables intermédiaires independantes de la clé.

RSA : $y = x^d \text{ mod } n$ square and multiply always ↓ DPA possible.

pre idee: $y = (\lambda \cdot x)^d / x^d \text{ mod } n$ (λ : aléa)



Avantage: L'attaque DPA n'est plus possible

Inconvénient: Il y a deux exécutions de RSA

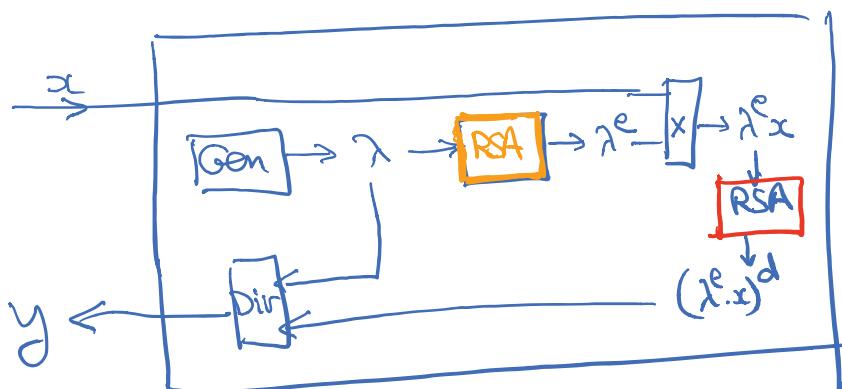
$$\begin{aligned} & \text{une division mod } n \xrightarrow{\text{Euclide étendu}} \lambda \text{ mod } n \\ & \qquad \qquad \qquad \xrightarrow{\text{inverse mod } n} \lambda^{(e(n)-1)} \text{ mod } n \\ & \text{exponentiation mod } n \end{aligned}$$

Remarque : Dans certaines cartes à puce, on dispose d'un coprocesseur RSA, qui peut calculer une exponentiation modulaire très rapidement (+ rapidement qu'Euclide étendue sur le CPU)

2^e idée :

$$y = \frac{(\lambda^e x)^d}{(\lambda^e)^d}$$

$$\Rightarrow \boxed{y = (\lambda^e \cdot x)^d / \lambda \pmod{n}} \quad (\lambda \text{ abordable})$$

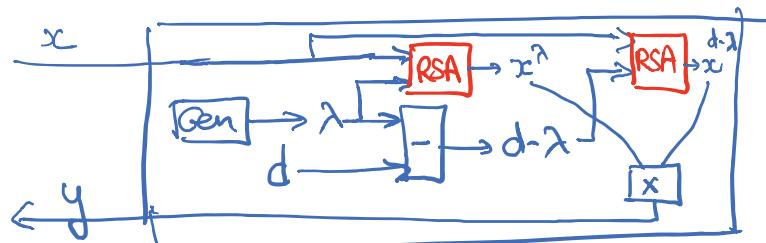


Avantage : On a

1) RSA avec l'exposant d	{	1 RSA avec _____ e
1 division mod n		
1 division mod n		

3^e idée :

$$\boxed{y = x^\lambda \cdot x^{d-\lambda} \pmod{n}} \quad (\lambda \text{ abordable})$$



Calcul \Rightarrow 2 RSA

4^{eme} idée: $y = (x^d \bmod (\lambda \cdot n)) \bmod n$

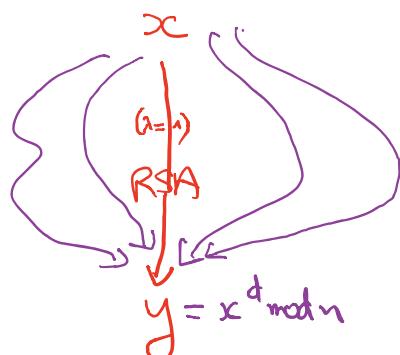
On peut prendre par exemple

$$\begin{aligned} |\lambda| &= 80 \text{ bits} \\ |n| &= 1024 \text{ bits} \end{aligned} \quad \left. \rightarrow |\lambda \cdot n| = 1104 \text{ bits} \right.$$

λ aléatoire

λ premier avec n

\hookrightarrow temps de calcul multiplié par $\left(\frac{1104}{1024}\right)^3 \approx 1,25$



Remarque: DPA = DPA d'ordre 1

