

Complexité algébrique et cryptographie

Alexandre Guillemot

25 janvier 2023

Table des matières

1	Problèmes difficiles en théorie des nombres	2
1.1	Complexité et cryptographie	2
1.1.1	Introduction	2
1.1.2	Calculabilité au sens de Turing	3
1.1.3	Complexités en temps et classes P , NP	4
1.1.4	Problèmes NP -complets	4
1.2	Factorisation	6
1.2.1	Complexité	6
1.2.2	Idée de Fermat	6
1.3	Logarithme discret	9
1.3.1	Complexité	9
1.3.2	Méthodes de calcul du log discret	9
1.3.3	Méthode du calcul d'indices	10
2	L'algorithme RSA en pratique	11
2.1	Rappels sur RSA	11
2.1.1	Définition	11
2.1.2	Sécurité de RSA	11
2.2	RSA en signature	12
2.2.1	Problématique	12
2.2.2	Fonctions de hachage	13
2.2.3	Exemple : la fonction de hachage de Chaum-van Heijst-Pfitzmann	14
2.2.4	Construction de Merkle-Damgard	15
2.2.5	Fonctions de hachage usuelles	17
2.2.6	Standards de signature RSA	17
2.3	RSA en chiffrement	17
2.3.1	Théorème de Coppersmith	17
2.3.2	Attaque de Hastad	20
2.3.3	Short pad attack	21
2.4	Attaques physiques	21

2.4.1	Carte à puce	21
2.4.2	Attaque SPA	21
2.4.3	Timing attack	21
2.4.4	Attaque DPA (Differential Power Analysis)	22
2.4.5	Attaques par injection de fautes	22

Chapitre 1

Problèmes difficiles en théorie des nombres

1.1 Complexité et cryptographie

1.1.1 Introduction

Idée : mesurer la "difficulté" algorithmique d'un problème.

|| **Définition 1.1.1.** (Problème de décision) Un problème de décision est une collection d'instances qui sont des ensembles de données qui admettent exactement une des deux réponses "oui" ou "non".

Ex 1.1.1. 1. Problème SAT (Satisfaisabilité)

Instance : Une fonction à variables booléenne $F : \{0, 1\}^n \rightarrow \{0, 1\}$ construite avec les connecteurs logiques \vee, \wedge, \neg . Par exemple,

$$f(x_1, x_2, x_3, x_4) = (\neg(x_1 \wedge (\neg x_3))) \vee (x_1 \wedge x_2 \wedge (\neg x_1))$$

Question : existe-t-il $x_1, \dots, x_n \in \{0, 1\}$ tels que $F(x_1, \dots, x_n) = 1$?

Algorithme : recherche exhaustive sur (x_1, \dots, x_n) , la complexité est en $\mathcal{O}(2^n)$.

2. FBQ (Formes Booléennes Quantifiées)

Instance : Une formule booléenne avec quantificateur e.g. $\forall x_i \exists x_j \dots F(x_1, \dots, x_n)$
(F est une fonction booléenne comme dans SAT)

Question : Cette formule est-elle vraie ?

Algorithme : Recherche exhaustive ($\mathcal{O}(2^n)$).

3. Equations diophantiennes (10^{ème} problème de Hilbert)

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

Question : Cette équation admet-elle des solutions entières ?

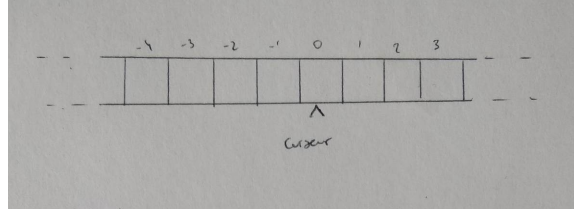
Algorithme : Matyasevich, 1971 : il n'y a pas d'algo qui répond à cette question.

1.1.2 Calculabilité au sens de Turing

Turing : Cryptanalyse d'Enigma, construction de machines dédiées à la cryptanalyse d'Enigma, Machine de Turing.

Modèle de Turing

On dispose d'un ruban infini



Chaque case contient un symbole (dans un alphabet fini Σ que l'on peut supposer être $\{0, 1\}$, ou le symbole blanc b). Le ruban v aêtre lu case par case par le curseur, la machine est à chaque instant dans un état $q_i \in Q$, où Q est l'ensemble fini des états possibles.

Définition 1.1.2. (Machine de Turing) Une opération élémentaire est entièrement déterminée par le symbole lu par le curseur, et par l'état actuel q_i :

1. Le curseur remplace le symbole par un élément de $\Sigma \cup \{b\}$
2. Le curseur se déplace soit d'une case vers la gauche, soit d'une case vers la droite, soit reste sur place.
3. La machine passe de l'état q_i à l'état q_j .

Une machine de Turing est donc la donnée d'une fonction

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

Définition 1.1.3. (Calcul déterministe) Le calcul déterministe d'une entrée x avec une machine de Turing M est la suite d'opération suivante :

1. La machine est commencée par être dans l'état q_0
2. Le curseur est placé sur la case 1
3. x est écrite sur les cases $1, \dots, n$ du ruban, les autres contenant b .

4. On applique itérativement M , le calcul se termine lorsque la machine atteint l'état final q_F . La sortie y est alors la donnée inscrite sur le ruban lorsque la machine termine.

Terminologie : L'ensemble des suites finies de symboles de Σ est noté Σ^* . Un mot est un élément de Σ^* . Une fonction $f : \Sigma^* \rightarrow \Sigma^*$ est turing calculable s'il existe une machine de Turing M qui sur toute entrée $x \in \Sigma^*$ calcule $y = f(x)$.

1.1.3 Complexités en temps et classes P , NP

Définition 1.1.4. La longueur d'un calcul sur une entrée $x \in \Sigma^*$ pour une machine de Turing M est le nombre $t_M(x)$ d'opérations élémentaires qui composent le calcul. Ainsi on définit la complexité en temps d'une machine de Turing comme

$$\begin{aligned} T_M : \mathbb{N} &\rightarrow \mathbb{N} \\ n &\mapsto \max_{\substack{x \in \Sigma^* \\ |x|=n}} \{t_M(x)\} \end{aligned}$$

Définition 1.1.5. Un algorithme polynomial \mathcal{A} pour calculer f est une machine de Turing M qui calcule f et telle qu'il existe un polynôme p tel que $\forall n \in \mathbb{N}, T_M(n) \leq p(n)$. On appelle classe P l'ensemble des problèmes de décision admettant un algorithme polynomial.

Définition 1.1.6. On dit qu'un pb de décision est calculable par un algo non déterministe polynomial s'il existe une machine de Turing M et un polynôme p tel que

1. La réponse est oui pour l'entrée x ssi il existe $y \in \Sigma^*$ (certificat) tel que M calcule 1 lorsqu'on met $x \in \Sigma^*$ dans les cases 1 à n et y dans les cases -1 à $-m$.
2. Pour tout x donnant la réponse 1, M calcule 1 et temps $\leq p(n)$

On appelle classe NP la classe des problèmes de décision admettant un algorithme non déterministe polynomial.

Rq 1.1.1. $P \subseteq NP$.

1.1.4 Problèmes NP -complets

Définition 1.1.7. On dit que le problème de décision p_1 se réduit au problème de décision p_2 s'il existe une fonction $\varphi : \Sigma^* \rightarrow \Sigma^*$ calculable en temps polynomial telle que la réponse à p_1 est oui pour l'entrée x si et seulement si la réponse à p_2 est oui pour l'entrée $\varphi(x)$.

Notation. On note $p_1 \times p_2$.

|| **Proposition 1.1.1.** $p_1 \in P$ et $p_1 \times p_2 \Rightarrow p_1 \in P$

|| **Définition 1.1.8.** Un problème Π est NP -complet ssi $\forall p \in NP, p \times \Pi$.

|| **Théorème 1.1.1.** (Cook, 1971) SAT est NP -complet.

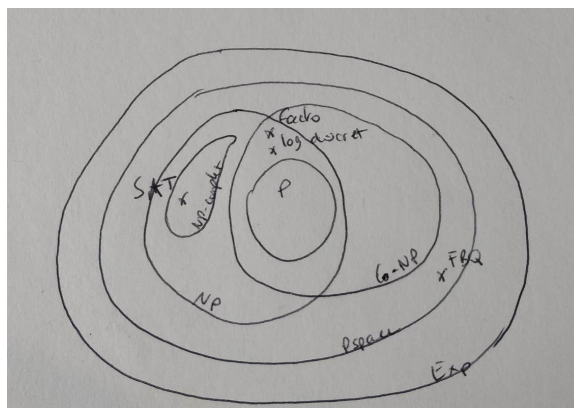
Rq 1.1.2. Si $SAT \times P$, alors P est NP -complet.

Ex 1.1.2. 1. SAT

2. 3-SAT
3. Circuit hamiltonien
4. 3-coloriabilité d'un graphe
5. TSP
6. Pb du sac à dos
7. Système de n équations quadratiques sur un \mathbb{F}_2 .

On conjecture que $P \neq NP$. Astuce de Levin : Supposons que $P = NP$: alors on peut construire un algorithme polynomial pour résoudre SAT . Considérons les machines de Turing M_1, M_2, \dots qui prennent en entrée une instance de SAT : Alors on fait tourner les machines simultanément, en faisant tourner de une étape M_1 , puis en faisant tourner de une étape M_2 , puis M_1 , puis M_3 , M_2 et M_1 , etc.

Résumé de la hiérarchie des complexités algorithmiques :



Rq 1.1.3. $Co-NP \cap NP\text{-complet} = \emptyset$.

1.2 Factorisation

1.2.1 Complexité

Problème de décision FACTM (problème des facteurs majorés)

- Instance : n entier, $M \leq n$.
- Question : Existe-t-il un diviseur de n qui est $\leq M$.

Si on a un algo polynomial de factorisation, alors on peut résoudre FACTM en temps polynomial. Inversement, Supposons \mathcal{A} algo polynomial pour FACTM. Comment factoriser n ? Soit p le plus petit facteur premier de n , alors

- On applique $\mathcal{A}(n, \sqrt{n})$. Si l'algorithme répond non, alors on termine et on répond non (car n est alors premier)
- Sinon, on applique $\mathcal{A}(n, \sqrt{n}/2)$. Si l'algo répond non, alors $p \in [\sqrt{n}/2, \sqrt{n}]$, et sinon $p \in [1, \sqrt{n}/2]$.
- On continue la dichotomie jusqu'à ce que la taille de l'intervalle obtenu soit plus petite que 1.

L'algorithme termine dès que $\sqrt{n}/2^k < 1$, où k est le nombre d'appels de \mathcal{A} . Ainsi il y a $k = \log_2(\sqrt{n})$ est donc de l'ordre de $\log n$. Une fois p trouvé, on recommence l'algo avec n/p . On va recommencer le nombre de facteurs premiers de n (comptés avec leur multiplicité). Mais

$$n = \prod_i p_i^{\alpha_i} \geq \prod_i 2^{\alpha_i} = 2^{\sum \alpha_i}$$

donc $\sum \alpha_i < \log_2 n$, et c'est aussi le nombre de facteurs premiers de n (comptés avec leur multiplicité). Au total, l'algorithme est polynomial.

1.2.2 Idée de Fermat

- L'idée naïve est d'essayer de diviser par les entiers successifs $\leq n$. C'est en $\mathcal{O}(n)$ donc exponentiel en la taille de l'entier.
- On peut aussi s'arrêter avant \sqrt{n} , mais l'algorithme reste exponentiel.
- On peut aussi diviser par les nombres premiers \sqrt{n} . D'après le théorème des nombres premiers (Hadamard, de la Vallée-Poussin), le cardinal des entiers premiers plus petits que x est asymptotiquement équivalent à $x/\ln x$. Ainsi l'algo est en $\mathcal{O}(\sqrt{n}/\log n)$, qui reste exponentiel en la taille de n .

Il suffit, pour factoriser n , de trouver x et y tels que $x^2 = y^2[n]$, avec $x \neq \pm y[n]$. En effet on a alors $(x - y)(x + y) = 0[n]$ et ainsi $\gcd(x, x - y)$ est un facteur de n . Fermat prend comme valeurs de x $\lfloor \sqrt{n} \rfloor + 1, \lfloor \sqrt{n} \rfloor + 2$, et espère que $x^2 - n$ est un carré parfait y^2 . SAUCISSE.

Ex 1.2.1. $n = 9167$, $\sqrt{n} = 95,7$, $96^2 = 49[n]$, et $49 = 7^2$, et alors $\gcd(9167, 96 + 7) = 103$, $\gcd(9167, 96 - 7) = 89$. On a bien $9167 = 103 \times 89$.

La complexité est de l'ordre de \sqrt{n} , c'est donc toujours exponentiel. Donnons un raffinement de la méthode : prenons $n = 849239$, $\sqrt{n} = 921,5$.

- $922^2 = 845 = 5 \times 13^2[n]$
- $933^2 = 2 \times 5^4 \times 17[n]$
- $937^2 = 2 \times 5 \times 13^2 \times 17[n]$

Et alors $(922 \times 933 \times 937)^2 = (2 \times 5^3 \times 13 \times 17)^2[n]$ et donc on a trouvé l'équation qu'on voulait, et après calcul des pgcd on obtiens que $1229 \times 691 = 849239$. On vient de décrire le crible quadratique de Pomerance.

Algorithmes

Décrivons le dans sa généralité : on veut factoriser n , pour cela

1. On se fixe une base de factorisation $B = \{-1, p_1, p_2, \dots, p_h\}$.
2. On dit qu'un entier est friable (ou lisse/smooth) s'il n'a que des petits facteurs premiers. Précisément, il est B -friable si tous des facteurs premiers sont dans B .
3. On va dire que b est B -adapté si le représentant de $b^2[n]$ dans l'intervalle $[-n/2, n/2]$ est B -friable.

Etape 1 : Obtenir et stocker des entiers b_i qui sont B -adaptés. On note

$$b_i^2 = (-1)^{\varepsilon_i} p_1^{\alpha_{i,1}} \cdots p_h^{\alpha_{i,h}} [n] \quad (1.1)$$

Etape 2 : A chaque relation 1.1, on associe

$$u_i = (u_{i,0}, \dots, u_{i,h}) \in \mathbb{F}_2^{h+1}$$

où $u_0 = \varepsilon_i[2]$, $u_{i,j} = \alpha_{i,j}[2]$ si $j \geq 1$.

Etape 3 : on a

$$\begin{aligned} \Rightarrow \prod_{i \in I} (b_i^2)^{\beta_i} &= \prod_{i \in I} \left((-1)^{\varepsilon_i} \prod_{j=1}^h p_j^{\alpha_{i,j}} \right)^{\beta_i} [n] \\ &= (-1)^{\sum_{i \in I} \beta_i \varepsilon_i} \times \prod_{j=1}^h p_j^{\sum_{i \in I} \beta_i \alpha_{i,j}} [n] \end{aligned}$$

donc si on trouve une combinaison linéaire des u_i qui est nulle

$$\sum_{i \in I} \beta_i u_i = 0$$

les exposants dans la dernière ligne du calcul sont pairs. On a donc $x^2 = y^2[n]$, avec

$$x := \prod_{i \in I} b_i^{\beta_i}, y = \prod_{j=1}^h p_j^{\frac{1}{2} \sum \beta_i \alpha_{i,j}}$$

Complexité de l'algorithme

Etape 2 : Il faut $|I| \geq h + 2$.

Etape 1 : Pour évaluer la complexité de l'étape 1, on utilise

Théorème 1.2.1. *On pose*

$$\psi(x, T) = |\{n \leq x \mid n \text{ a tous ses facteurs premiers } \leq T\}|$$

Pour $1 \leq T \leq x$, on pose $v := \frac{\ln x}{\ln T}$, 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 qui sont $\leq T = \exp\left(\frac{1}{2}\sqrt{\ln n \ln \ln n}\right)$. Alors

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

donc $\ln v \simeq \frac{1}{2} \ln \ln n$. Ainsi le nombre de valeurs à essayer dans la première étape est $(h+2)v^v$. Et

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

Ainsi le nombre d'essais vaut $T(h+2)$, et $h+2 \sim T/\ln T$.

Etape 3 : Finalement, la complexité de l'algorithme complet vaut

$$\frac{T^2}{\ln T} + (h+2)^3 \sim \left(\frac{T}{\ln T}\right)^3 = e^{\frac{3}{2} \sqrt{\ln n \ln \ln n}}$$

On vient donc de décrire un algorithme sous-exponentiel.

Notation.

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

- $\alpha = 0$: alors $L_{0,c}(z) = e^{c \ln \ln z} = (\ln z)^c$ donc complexité polynomiale.

- $\alpha = 1$: $L_{1,c}(z) = e^{c \ln z}$ donc complexité exponentielle.
- $0 \leq \alpha \leq 1$, alors $L_{\alpha,c}(z)$ est sous-exponentiel.

Dans le cas de l'algorithme que l'on vient de décrire, la complexité vaut $L_{\frac{1}{2},c}$. Actuellement, le meilleur algorithme pour des nombres types clés de RSA est GNFS (General Number Field Sieve) dont la complexité est $L_{\frac{1}{3},c}(n)$ avec $c = 1,92$ (algorithme dû à H. Lenstra, A. Lenstra, Manasse, Pollard, 1990).

Rq 1.2.1. Si on veut $L_{\frac{1}{3},c}(n) \geq 2^{80}$, il faut prendre $|n| = 1024$ bits.

Rq 1.2.2. L'exposant 3 qui vient du pivot de gauss (3eme étape) peut être amélioré : le système à résoudre est un système creux :

$$b_i^2 = \prod_{j=1}^h p_j^{\alpha_{i,j}} [n]$$

Le nombre de facteurs premiers qui interviennent dans cette décomposition est de l'ordre $\mathcal{O}(\ln n)$. Ainsi les lignes du système à résoudre contiennent beaucoup de zéros, et il existe un algorithme (Block-Lanczos) pour ce genre de système qui est en $\mathcal{O}(dh^2)$ où h est la dimension du système et d est le nombre maximal d'éléments non nuls dans chaque ligne.

1.3 Logarithme discret

1.3.1 Complexité

Problème du log discret : soit p un nombre premier, g un générateur de $\mathbb{Z}/p\mathbb{Z}^*$. À partir de $y = g^x[p]$, retrouver x ? On peut lui associer le problème de décision suivant :

Instance : p, g, y, t

Question : Le log discret de y par rapport à g est-il $\leq t$?

Si on a un algo A polynomial pour le problème de décision, alors de manière similaire au problème de factorisation (dichotomie), on peut calculer le log discret en temps polynomial.

1.3.2 Méthodes de calcul du log discret

- Méthode naïve : recherche exhaustive sur x , complexité en $\mathcal{O}(p)$ (donc exponentiel).
- Méthode baby step giant step : On regarde la division euclidienne de x par a où $a = \lfloor \sqrt{n} \rfloor$. Trouver x est équivalent à trouver q, r et poser $x = aq + r$. Maintenant

$$y = g^x[p] \iff yg^{-aq} = g^r[p]$$

On peut créer 2 tables de 0 à a indexées par q et r où on calcule yg^{-aq} et g^r , et on cherche une valeur commune. Si les tables sont triées, alors la recherche d'une valeur commune est en $\mathcal{O}(a)$.

1.3.3 Méthode du calcul d'indices

On cherche x tel que $g^x = y[p]$.

Etape 1 : On choisit $B = \{-1, p_1, \dots, p_h\}$

- On choisit c_i aléatoire
- On calcule le représentant de $g^{c_i}[p]$ dans $[-p/2, p/2]$ et on espère que $g^{c_i}[p] = \prod_{j=0}^h p_j^{\alpha_{i,j}} (*)$, dans ce cas on aura $c_i = \sum_{j=0}^h \alpha_{i,j} \log_g(p_j)[p_1]$.

Etape 2 : Si on a obtenu $\geq h + 1$ relations du type $(*)$, on pourra trouver les $\log_g(p_j)$ pour $0 \leq j \leq h$.

Etape 3 : On calcule $yg^e[p]$ où e est aléatoire. Avec une certaine probabilité, $yg^e = \prod_{j=0}^h p_j^{\beta_j}$, et alors

$$\log_g(y) = \left(\sum_{j=0}^h \beta_j \log_g(p_j) \right) - e$$

Par un argument similaire à l'analyse de complexité de l'algorithme de factorisation, on obtiens une complexité en $\mathcal{O}(e^{1+o(1)} \sqrt{\ln p \ln \ln p})$, doit du $\mathcal{O}(L_{\frac{1}{2}, 1+o(1)}(p))$. Le meilleur algorithme est en $L_{\frac{1}{3}, c}$ avec $c = 1, 92$.

Chapitre 2

L'algorithme RSA en pratique

2.1 Rappels sur RSA

2.1.1 Définition

Histoire

1976 : Diffie Hellman, New Directions in Cryptography. 1977 : Merkle, "puzzle de Merkle". Rivest, Shamir, Adleman, RSA.

On fixe e impair ($e = 3, e = 17, e = 257$). On calcule des entiers p, q premiers distincts tels que $\gcd(e, (p-1)(q-1)) = 1$. On pose $n = pq$. Finalement, on calcule $d = e^{-1}[\varphi(n)]$. Clé publique : (n, e) . Clé secrète : $(p, q, d, \varphi(n))$.

Théorème 2.1.1. *Si p, q sont premiers distincts, $n = pq$, $\gcd(e, \varphi(n)) = 1$, $d = e^{-1}[\varphi(n)]$, alors*

$$\begin{array}{ccc} f : \mathbb{Z}/n\mathbb{Z} & \rightarrow & \mathbb{Z}/n\mathbb{Z} \\ x & \mapsto & x^e[n] \end{array}$$

est bijective d'inverse

$$\begin{array}{ccc} f^{-1} : \mathbb{Z}/n\mathbb{Z} & \rightarrow & \mathbb{Z}/n\mathbb{Z} \\ x & \mapsto & x^d[n] \end{array}$$

2.1.2 Sécurité de RSA

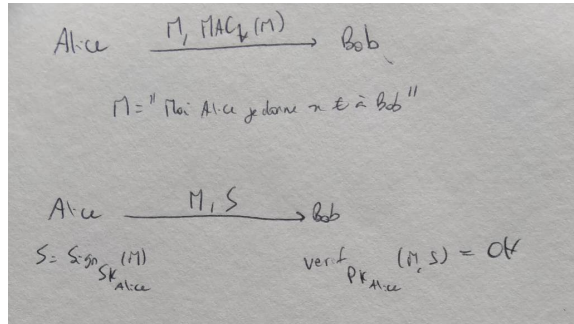
Objectif de l'attaquant :

1. Trouver la clé secrète
2. Calculer $f^{-1}(y)$ pour certains y .

- Trouver $p, q, d, \varphi(n)$ à partir de n et e : déjà, si on connaît p ou q , on peut facilement retrouver tout le reste de la clé. Ensuite si on connaît $\varphi(n) = pq - (p + q) + 1$ et $pq = n$, donc $p + q = n - \varphi(n) + 1$, $pq = n$, et alors on peut trouver p, q . Enfin si on connaît d , alors $ed = 1[\varphi(n)]$. Prenons x aléatoire, on calcule $y = x^{\frac{ed-1}{1}}[n]$. Et alors $y^2 = x^{ed-1} = 1[n]$. Mais alors y est solution de l'équation $y^2 = 1[n]$, qui a 4 solutions : $(\pm 1, \pm 1) \in \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z}$ au travers du théorème des restes chinois. Mais alors si y correspond à $(1, -1)$ ou $(-1, 1)$ (notons $\alpha, -\alpha$ les éléments correspondants dans $\mathbb{Z}/n\mathbb{Z}$), alors $\gcd(y - 1, n) = p$ ou q (vu que $\alpha = 1[p]$ et $\alpha = -1[q]$).
- Trouver $f^{-1}(y)$ pour certains y (problème de la racine e -ième modulo n) : si on sait factoriser, on sait résoudre le problème de la racine e -ième grâce au théorème des restes chinois. On ne sait cependant pas si savoir résoudre le problème des racines e -ièmes nous permettrait de résoudre facilement le problème de factorisation.

2.2 RSA en signature

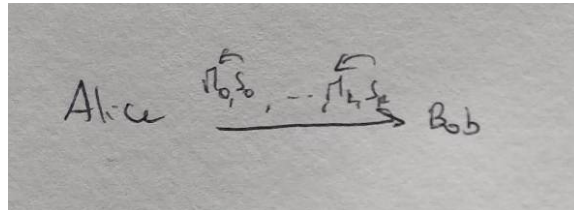
2.2.1 Problématique



Algorithme de signature naïf

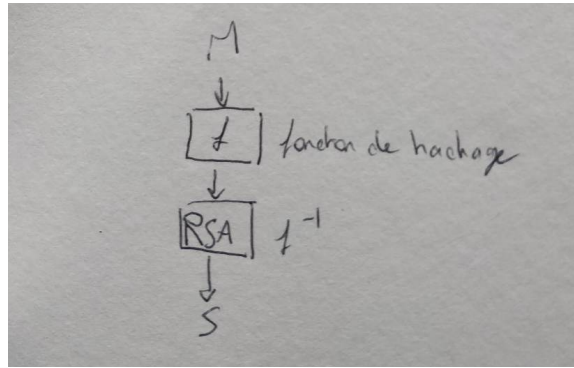
On signe avec $f^{-1}(M) = M^d[n]$. Déjà, on doit supposer que $0 \leq M < n$ car sinon on aurait plusieurs messages avec la même signature.

- Si M est grand, on pourrait écrire $M = \sum M_k n^k$ avec $M_k \in [0, n - 1]$, puis on signe par blocs, pas terrible ...



- Problème 2 : Si Alice envoie deux messages signés (M, S) et (M', S') , alors charlie peut signer MM' en calculant SS' .
- Problème 3 : Si Alice envoie un message M, S , alors charlie peut envoyer $M^2, S^2, \lambda^e M, \lambda S$.
- Problème 4 : charlie peut envoyer $(0, 0), (1, 1), (\lambda^e, \lambda)$.

Paradigme "hash and sign"



2.2.2 Fonctions de hachage

$h : \{0, 1\}^* \rightarrow \{0, 1\}^l$ avec $\{0, 1\}^* = \sqcup_{n \geq 0} \{0, 1\}^n$ et l un entier fixé.

Définition 2.2.1. Une telle fonction $h : \{0, 1\}^* \rightarrow \{0, 1\}^l$ est appelée fonction de hachage si elle vérifie les 3 propriétés suivantes :

- p_1 : h est à sens unique, i.e. pour $y \in \{0, 1\}^l$, il est calculatoirement difficile de trouver un antécédent x et y .
- p_2 : h est à collisions faibles difficiles (second preimage resistant) i.e. 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 $h(x') = y$.
- p_3 : h est à collisions fortes difficiles (collision resistant) i.e. il est calculatoirement difficile de trouver $x, x' \in \{0, 1\}^*$ tel que $x \neq x'$ et $h(x) = h(x')$.

Rq 2.2.1. $p_2 \Rightarrow p_1, p_3 \Rightarrow p_2$. Ainsi d'un point de vu mathématique, p_3 suffit, mais il est intéressant de les écrire vu qu'elles ont un intérêt cryptographique.

1. Pour la propriété p_1 , on a un algo qui trouve un antécédent par recherche exhaustive (on tire aléatoirement x et on regarder si $h(x) = y$). La complexité est en 2^l .
2. On peut faire la même chose pour la propriété p_2 .

3. On génère aléatoirement x_1, x_2, \dots , et on calcule leurs images $y_i = h(x_i)$ jusqu'à trouver une égalité du type $y_i = y_j$. P = proba d'obtenir une égalité $y_i = y_j$, on peut calculer la probabilité de ne pas avoir de collision $1 - P$.

Algorithme pour rechercher des collisions : on fabrique une table de l'ordre de $k \simeq 2^{l/2}$, donc en $\mathcal{O}(k)$, on la trie $\mathcal{O}(k \ln k)$ et la recherche d'une collision est en $\mathcal{O}(k)$. Donc au total une complexité en $\mathcal{O}(l2^{l/2})$. Ainsi on doit prendre $l \geq 256$.

2.2.3 Exemple : la fonction de hachage de Chaum-van Heijst-Pfitzmann

Soit p un nombre premier tel que $q = \frac{p-1}{2}$ est aussi premier, g, h deux éléments d'ordre q dans $\mathbb{Z}/p\mathbb{Z}^*$. On définit

$$H : \mathbb{Z}/q\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z} \rightarrow \mathbb{Z}/p\mathbb{Z}^* \\ (m_1, m_2) \mapsto g^{m_1} h^{m_2}$$

(Ce n'est pas à proprement parler une fonction de hachage vu l'ensemble de départ, mais une fonction de compression).

Quelle est l'image de H ? Considérons $A = \langle g \rangle$, $B = \langle h \rangle$, et $C = \{u \in \mathbb{Z}/p\mathbb{Z}^* \mid u^q = 1\}$.

Alors $A \subseteq C$, $B \subseteq C$, et $|C| \leq q = |A| = |B|$, donc au final $A = B = C$. Et finalement si $y \in A = B = C$, il existe $t \mid y = g^t$ donc $y = H(t, 0)$ et ainsi $A \subseteq \text{Im}H$ (l'inclusion réciproque est claire).

Supposons que Charlie trouve une collision : $H(m_1, m_2) = H(m_3, m_4)$ avec $(m_1, m_2) \neq (m_3, m_4)$.
Alors

$$g^{m_1} h^{m_2} = g^{m_3} h^{m_4} \iff g^{m_1 - m_3} = h^{m_4 - m_2}$$

Posons $d = \gcd(m_4 - m_2, p - 1) \in \{1, 2, q, p - 1\}$, alors

- Si $d = 1$, $m_4 - m_2$ a une inverse w modulo $p - 1$, et alors

$$g^{(m_1 - m_3)} = h^{(m_4 - m_2)w} = h$$

- Si $d = 2$, alors $\gcd(m_4 - m_2, q) = 1$. Notons alors θ l'inverse de $m_4 - m_2$ modulo q , alors

$$g^{(m_1 - m_3)\theta} = g^{(m_4 - m_2)\theta} = h$$

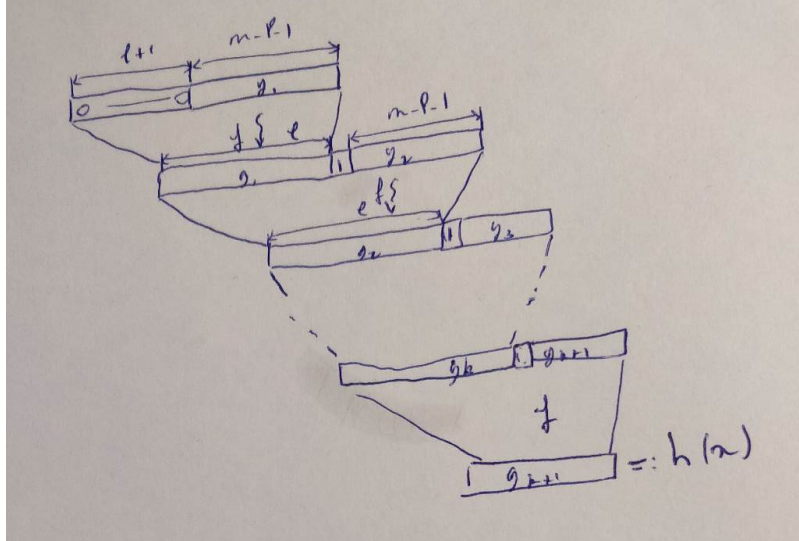
- Si $d = q$, alors $q \mid m_4 - m_2$, donc $m_2 = m_4$. Ainsi $g^{m_1 - m_3} = 1[p]$, donc $q \mid m_1 - m_3$ et donc $m_1 = m_3$, donc les deux messages sont égaux, absurde.
- Si $d = p - 1$, de même $q \mid m_4 - m_2$...

Ainsi savoir trouver une collision de cette fonction de hachage implique que l'on sait résoudre le problème du log discret. Ainsi réciproquement si on trouve g et h tels que le problème du log discret est difficile avec, alors il sera difficile de trouver une collision pour cette fonction de hachage.

2.2.4 Construction de Merkle-Damgard

En chiffrement symétrique, si on veut chiffrer un gros bloc $x = x_1x_2 \cdots x_k$, on dispose de modes d'opérations pour chiffrer ces blocs : on peut par exemple chiffrer chaque bloc les uns après les autres (ECB). On peut aussi utiliser le mode CBC.

Hypothèse : $f : \{0,1\}^m \rightarrow \{0,1\}^l$ est une fonction de compression ($m > l + 1$). Peut-on fabriquer une fonction de hachage à partir de cette fonction de compression ? Soit $x \in \{0,1\}^*$, on le découpe en blocs $x = x_1 \| x_2 \| \cdots \| x_k$, en blocs de $m - l - 1$ bits pour les $k - 1$ premiers blocs, et le dernier bloc est de taille $\leq m - l - 1$. On définit ensuite $y = y_1 \| y_2 \| \cdots \| y_k \| y_{k+1}$ avec $y_i = x_i$ pour tout $1 \leq i \leq k - 1$, et $y_k = x_k \| 0 \cdots 0$ est la complétion de x_k avec des zéros (disons d) pour faire $m - l - 1$ bits. Finalement on pose $y_{k+1} = d$ (son écriture binaire). Finalement on hash y de la manière suivante :



|| **Théorème 2.2.1.** Si f est à collision fortes difficiles, alors h aussi.

Démonstration. Supposons qu'on ait une collision $h(x) = h(x')$ avec $x \neq x'$. x, x' correspondent à y, y' par la construction décrite précédemment. Remarquons alors qu'au vu de la construction, $y \neq y'$. Notons k, t le nombre de blocs de x, x' (donc y, y' ont $k+1, t+1$ blocs). Ainsi avec les notations de la figure précédente, $g_{k+1} = g'_{t+1}$, i.e. $f(g_k \| 1 \| y_{k+1}) = f(g'_t \| 1 \| y'_{t+1})$.

- Si $y_{k+1} \neq y'_{t+1}$, alors on obtiens une collision pour f .
- Si $g_k \neq g'_t$, alors on obtiens aussi une collision pour f .
- Si $g_k = g'_t$ et $y_{k+1} = y'_{t+1}$, alors $f(g_{k-1} \| 1 \| y_k) = f(g'_{t-1} \| 1 \| y'_t)$, et on réapplique le point précédent. On peut alors soit trouver une collision dans ce procédé, soit remonter complètement la construction de h . Il y a alors trois cas : soit on remonte

la construction pour y mais pas pour y' , (donc $k < t$), alors $g_1 = g'_{t-k+1}$, ainsi $f(0 \cdots 0 \| y_1) = f(g'_{t-k} \| 1 \| y'_{t-k+1})$, donc on obtiens une collision. De même si on remonte la construction pour y' mais pas pour y . Et finalement si on remonte les deux constructions en même temps, alors $f(0 \| y_1) = f(0 \| y'_1)$, alors si $y_1 \neq y'_1$, on obtiens une collision, et sinon $y = y'$ et donc $x = x'$, contradiction.

□

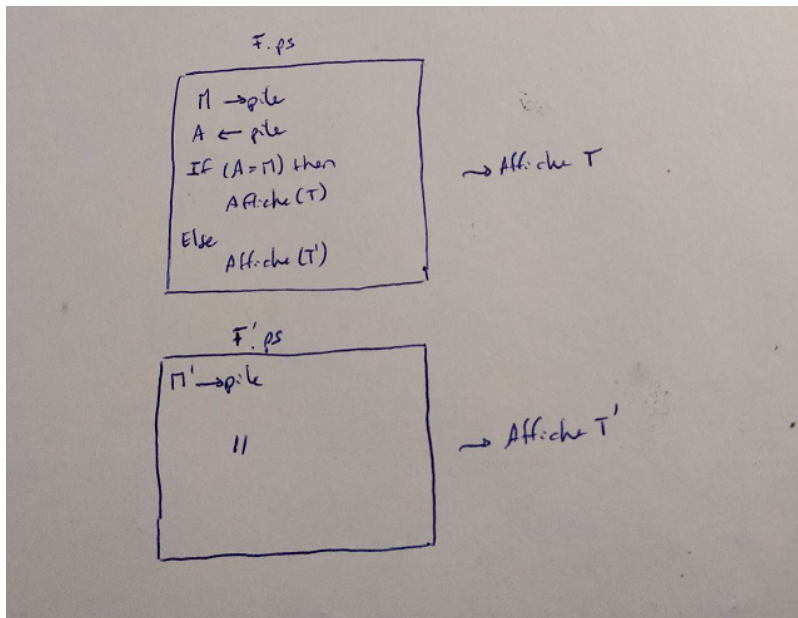
Attaque de Bleichenbacher

Hypothèses : • h fonction de hachage construite avec Merkle-Damgard

- On dispose d'une collision M et M' avec $h(M) = h(M')$ et $M \neq M'$.

Objectif : l'attaquant choisit deux textes T et T' . Problème : construire deux fichiers F et F' tels que F affiche T , F' affiche T' , et $h(F) = h(F')$.

Fichiers postscript .ps :



Mais comme M et M' collisionnent, alors le haché de ces deux fichiers est le même et on a bien obtenu ce que l'on voulait.

2.2.5 Fonctions de hachage usuelles

Nom	Année	Inventeurs	l	Anniv	Meilleure attaque connue
MD4	1990	Rivest	128	2^{64}	1995, Dobbertim trouve une collision
MD5	1991	Rivest	128	2^{64}	2005, Wang : plusieurs collisions en 2^{24}
SHA-0	1993	NIST	160	2^{80}	2004, Joux : collision en 2^{51}
SHA-1	1994	NIST	160	2^{80}	2006, Wang : algo pour trouver des collisions en 2^{69} . En 2016, Stevens, ... trouvent une collision
Famille SHA-2	2002	NIST	256	2^{128}	\emptyset
SHA-256			384	2^{192}	
SHA-384			512	2^{256}	
SHA-512					
SHA-3	2011	Bertoni, Daemen, Peters, van Asche (KECCAK)	224	2^{112}	\emptyset
			256	2^{128}	
			384	2^{192}	
			512	2^{256}	

2.2.6 Standards de signature RSA

Pour signer avec RSA, on réalise $s = (\mu(M))^d[n]$ où μ est un standard de signature RSA.

Ex 2.2.1. PKCS (RSA DATA Security), Public Key Cryptographic Standard, propose (PKCS 1 v1.7)

$$\mu(M) = 00\|01\|FF \cdots FF\|00\|c_h\|h(M) < n$$

où c_h est le numéro de la fonction de hachage.

Ex 2.2.2. ISO 9796-2 : on découpe $M = M_1\|M_2$ de sorte que

$$\mu(M) = 6A\|M_1\|h(M)\|BC < n$$

Alice envoie alors M_2 et la signature (Bob peut alors récupérer M_1 dans la signature).

2.3 RSA en chiffrement

2.3.1 Théorème de Coppersmith

|| **Théorème 2.3.1.** (Coppersmith, 1997) N entier, $f \in \mathbb{Z}[X]$ polynôme unitaire de degré d . Posons $B = N^{\frac{1}{d}-\varepsilon}$ ($\varepsilon > 0$), alors étant donné f et N , on peut trouver efficacement tous les entiers x_0 tels que $|x_0| < B$ et $f(x_0) \equiv 0[N]$.

Or si on chiffre $C = M^e[N]$, cela revient à résoudre le polynôme $f(x) = x^e - C$. Et dans ce cas, si $M < N^{\frac{1}{e}}$, alors $M^e < N$ et donc $C = M^e$, d'où $M = \sqrt[e]{C}$.

Pour la démonstration, nous aurons besoin d'un lemme. Pour cela, soit $h(x) = \sum a_i x^i \in \mathbb{Z}[X]$, alors on pose $\|h\|^2 = \sum |a_i|^2$. On note $h(B\cdot)$ le polynôme $h(BX)$. Alors on a

|| **Lemme 2.3.1.** *Pour $h \in \mathbb{Z}[X]$ de degré d et $B > 0$ entier, supposons que $\|h(B\cdot)\| < \frac{N}{\sqrt{d+1}}$ et si $|x_0| < B$ satisfait $h(x_0) = 0[N]$, alors $h(x_0) = 0$.*

Démonstration.

$$\begin{aligned} |h(x_0)| &= \left| \sum a_i x_0^i \right| = \left| \sum a_i \frac{x_0^i}{B^i} B^i \right| \\ &\leq \sum \left| a_i \frac{x_0^i}{B^i} B^i \right| \leq \sum |a_i B^i| \\ &\leq \sqrt{d+1} \|h(B\cdot)\| < N \end{aligned}$$

Et ainsi $h(x_0) = 0$. □

L'idée de Coppersmith est de considérer la famille

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

Si x_0 est racine de f modulo N , alors x_0 est racine de $g_{u,v}$ modulo N^m . En effet, on peut alors écrire $f(x_0) = \lambda N$, et alors $N^{m-v} x_0^u \lambda^v N^v = 0[N^m]$. On va prendre les $g_{u,v}$ pour $u = 0, 1, \dots, d-1$, et $v = 0, \dots, m$. Mettons brièvement en pause la démonstration pour parler de réseaux.

Réseaux euclidiens

|| **Définition 2.3.1.** Soit $u_1, u_2, \dots, u_w \in \mathbb{Z}^w$ Entiers ou réels? peut-être peu importe pour la suite des vecteurs linéairement indépendants. On appelle réseau L engendré par u_1, \dots, u_w l'ensemble des combinaisons linéaires à coefficients entiers des u_1, \dots, u_w .

|| **Définition 2.3.2.** Le déterminant de L est le déterminant de la matrice dont les lignes sont les vecteurs u_1, \dots, u_w .

|| **Théorème 2.3.2.** (Hermite) Tout réseau L de dimension w contient un vecteur $v \in L \setminus \{0\}$ dont la norme vérifie $\|v\| \leq \gamma_w(\det L)^{\frac{1}{w}}$.

Algorithme LLL (Lovasz, A. Lenstra, H. Lenstra, 1982) : étant donné L un réseau engendré par les vecteurs u_1, \dots, u_w , l'algorithme renvoie un vecteur v non nul du réseau tel que $\|v\| < 2^{\frac{w}{4}} (\det L)^{\frac{1}{w}}$ en temps polynomial.

Revenons au problème initial :

Ex 2.3.1. $m = 3, d = 2$,

	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

Ainsi $\det L$ est le produit des termes sur la diagonale.

Plus généralement,

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

On veut trouver m tel que $2^{\frac{w}{4}} (\det L)^{\frac{1}{w}} < \frac{N^m}{\sqrt{w}}$, alors on pourra trouver un vecteur v tel que $\|v\| \leq \frac{N^n}{\sqrt{w}}$ en utilisant l'algo LLL (on sera alors dans les conditions du lemme)

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

Et donc il suffit de trouver m tel que

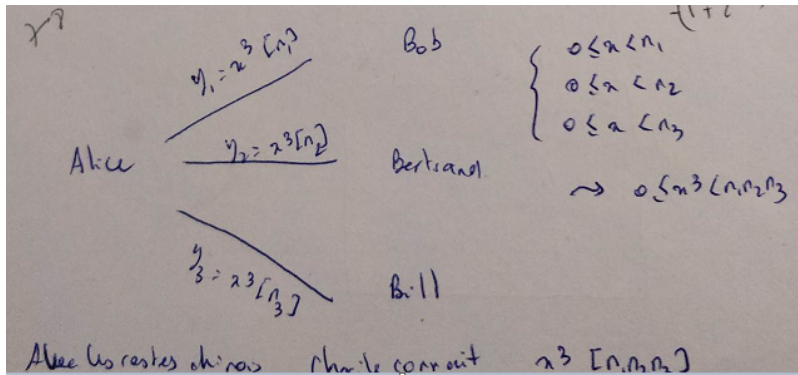
$$2^{\frac{d(m+1)}{4}} N^{\frac{m}{2}} B^{\frac{d(m+1)-1}{2}} < \frac{N^m}{\sqrt{d(m+1)}}$$

Mais

$$2^{\frac{d(m+1)}{4}} N^{\frac{m}{2}} B^{\frac{d(m+1)-1}{2}} < \frac{N^m}{\sqrt{d(m+1)}} \iff B < \alpha(d, m) N^{\frac{m}{d(m+1)-1}} < N^{\frac{1}{d}-\varepsilon}$$

pour m assez grand.

2.3.2 Attaque de Hastad



Piste de réparation : l'utilisateur numéro i publie N_i , e_i (son modulo et son exposant public), et $f_i \in \mathbb{Z}/N_i\mathbb{Z}[x]$, et il envoie $(f(M_i))^{e_i}[N_i]$.

Théorème 2.3.3. (*Attaque de Hastad améliorée*)

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

S'il existe un unique $M < N_{\min}$ tel que $\forall 1 \leq i \leq k, g_i(M) = 0[N_i]$, et si $k \geq d$, alors on peut trouver M en temps polynomial.

Démonstration. Notons $\overline{N} = N_1 N_2 \dots N_k$.

- On peut supposer les g_i unitaires (si ce n'est pas possible, on peut factoriser N_i par un pgcd du coefficient dominant de g_i).

- On suppose que tous les g_i sont de degré d (quitte à multiplier par une puissance de X)
- On définit le polynôme

$$g(x) = \sum_{i=1}^k T_i g_i(x) \in \mathbb{Z}/\overline{N}\mathbb{Z}[X]$$

avec $T_i = 1[N_i]$ et $T_i = 0[N_j]$ pour tout $j \neq i$.

g est unitaire de degré d , $g(M) = 0[\overline{N}]$, $M < N_{min} \leq \overline{N}^{\frac{1}{k}} \leq \overline{N}^{\frac{1}{d}}$. Ainsi d'après le théorème de Coppersmith, on trouve M en temps polynomial. \square

2.3.3 Short pad attack

Idée : pour chiffrer M , on calcule $C = (2^m M + r)^e[N]$ (où M est le message clair, d'au plus $k - m$ bits, et r fait m bits). Cela revient à chiffrer la concatenation $M||r$.

Attaque (Coppersmith) : (N, e) clé publique de RSA, N de taille k bits, $m = \lfloor \frac{k}{e^2} \rfloor$. Supposons que l'attaquant récupère deux chiffrés de M $C_1 = M_1^e[N]$ avec $M_1 = 2^m M + r_1$, $C_2 = M_2^e[N]$ avec $M_2 = 2^m M + r_2$. On définit $g_1(x, y) = x^e - c_1$, $g_2(x, y) = (x+y)^e - c_2$. Quand $y = r_2 - r_1$, alors M_1 est une racine commune pour $g_1(\cdot, y)$ et $g_2(\cdot, y)$. Soit $h(y) = \text{Res}_x(g_1, g_2)$. Alors $h(y) = 0$ quand $y = r_2 - r_1 \dots$ ff

2.4 Attaques physiques

2.4.1 Carte à puce

FIGURE 1

2.4.2 Attaque SPA

Paul Kocher, 1998. FIGURE 2 Square and multiply always : on fait multiply à chaque itération de l'algo.

2.4.3 Timing attack

FIGURE 3

Hypothèse : le calcul de $a \times b[n]$ prends un temps α ou $\beta > \alpha$ (par exemple la multiplication modulaire de Montgomery).

Question : On réalise square and multiply always, la question étant a-t-on calculé $x^6 * x[n]$?

$$\begin{aligned} A &= \{1 \leq i \leq n \mid x_i^6 \times x_i[n] \text{ prends un temps } \alpha\} \\ B &= \{1 \leq i \leq n \mid x_i^6 \times x_i[n] \text{ prends un temps } \beta\} \\ T_A &= \frac{1}{|A|} \sum_{i \in A} T_i \\ T_B &= \frac{1}{|B|} \sum_{i \in B} T_i \end{aligned}$$

Pour $i \in A$, $T_i = \text{temps}(x_i^6 \times x_i) + \text{autre temps}$. Pareil pour $i \in B$. Donc

$$\begin{aligned} T_A &= \alpha + \frac{1}{|A|} \sum_{i \in A} \text{bruit} \\ T_B &= \beta + \frac{1}{|B|} \sum_{i \in B} \text{bruit} \end{aligned}$$

Ainsi $T_b - T_a = \beta - \alpha + \mathcal{O}\left(\frac{1}{\sqrt{|A|}} + \frac{1}{\sqrt{|B|}}\right)$ si $x^6 \times x$ existe, sinon seulement $\mathcal{O}\left(\frac{1}{\sqrt{|A|}} + \frac{1}{\sqrt{|B|}}\right)$.

Cela nous permet de savoir si le calcul a été réalisé, et on peut donc de proche en proche déduire la marche de l'algorithme square and multiply

2.4.4 Attaque DPA (Differential Power Analysis)

FIGURE 4

$$\begin{aligned} A &= \{1 \leq i \leq N, \text{ le bit de poids fort de } x_i^6 \times x_i[n] \text{ vaut } 0\} \\ B &= \{1 \leq i \leq N, \text{ le bit de poids fort de } x_i^6 \times x_i[n] \text{ vaut } 1\} \\ C_A &= \frac{1}{|A|} \sum_{i \in A} C_i \\ C_B &= \frac{1}{|B|} \sum_{i \in B} C_i \end{aligned}$$

2.4.5 Attaques par injection de fautes

Figure 1

- Lasers

- Champ électromagnétique, courants de foucault
- Pic d'alimentation électrique

1996 : Boneh, Lipton, de Millo