

# Fondements théoriques de la cryptographie

Duong Hieu Phan & Philippe Guillot

26 octobre 2017

---

# Sommaire

<i>Introduction</i> .....	5
<i>Chapitre 1.</i> <b>Fonctions à sens unique</b> .....	9
§ 1. Fonction à sens unique concrète .....	9
§ 2. Attaques génériques .....	10
§ 3. Des exemples pratiques .....	11
§ 4. Fonction à sens unique asymptotique .....	12
§ 5. Famille de fonctions à sens unique .....	13
§ 6. Exercices .....	15
<i>Chapitre 2.</i> <b>Prédicat difficile</b> .....	17
§ 1. Définition .....	17
§ 2. Théorème de Goldreich-Levin .....	17
§ 3. Applications .....	21
§ 4. Exercices .....	22
<i>Chapitre 3.</i> <b>Générateurs pseudo-aléatoires</b> .....	25
§ 1. Indistinguabilité statistique .....	25
§ 2. Indistinguabilité calculatoire .....	26
§ 3. Construction de générateurs pseudo-aléatoires .....	29
§ 4. Exercices .....	32
<i>Chapitre 4.</i> <b>Fonctions pseudo-aléatoires</b> .....	35
§ 1. Motivation et définition .....	35
§ 2. Construction d'une famille de fonctions pseudo-aléatoires .....	36
§ 3. Exercices .....	39
<i>Chapitre 5.</i> <b>Chiffrement</b> .....	41
§ 1. Schéma de chiffrement .....	41
§ 2. Notions de sécurité .....	42
§ 3. Modèles d'attaques .....	47
§ 4. Construction d'un schéma de chiffrement IND-CPA .....	47
<i>Chapitre 6.</i> <b>Codes d'authentification</b> .....	51
§ 1. Code d'authentification de message .....	51
§ 2. Adversaire d'un code d'authentification .....	52
§ 3. Construction à partir d'une fonction pseudo-aléatoire .....	53
§ 4. Chiffrement CCA-sûr générique .....	54
§ 5. Exercices .....	56

<i>Chapitre 7.</i>	<b>Chiffrement par blocs</b>	<b>57</b>
§ 1.	Motivation et définition	57
§ 2.	Construction à partir d'une famille de FPA	58
§ 3.	Exercices	61
<i>Chapitre 8.</i>	<b>Modes d'utilisation</b>	<b>63</b>
§ 1.	Chiffrement de messages de tailles variables	63
<i>Chapitre 9.</i>	<b>Chiffrement à clé publique</b>	<b>67</b>
§ 1.	Problèmes d'appartenance à un sous-groupe	67
§ 2.	Difficulté du problème de l'appartenance	68
§ 3.	Le chiffrement ElGamal est IND-CPA	69
§ 4.	Exercices	71
<i>Chapitre 10.</i>	<b>Signatures numériques</b>	<b>73</b>
§ 1.	Les cinq mondes d'Impagliazzo	73
§ 2.	Définition et sécurité des signatures	74
§ 3.	Le paradigme de Fiat-Shamir	76
§ 4.	La signature « une fois » de Lamport	77
§ 5.	Une signature « plusieurs fois » avec état	78
§ 6.	Fonction de hachage	79
§ 7.	Le paradigme « hache puis signe ».	82
§ 8.	Fonctions de hachage universelles à sens unique	84
§ 9.	Exercices	86
<i>Chapitre 11.</i>	<b>Chiffrement à clé publique</b>	<b>89</b>
§ 1.	Sécurité	89
§ 2.	Historique	90
§ 3.	CPA-sécurité sémantique du schéma ElGamal	91
§ 4.	Chiffrement CCA-sûr dans le modèle standard	92
§ 5.	Exercices	97
<i>Corrigé des exercices</i>		<b>98</b>
<i>Index alphabétique</i>		<b>105</b>

# Introduction

## Insuffisance de l'approche classique

Dans l'approche traditionnelle le concepteur de procédé de chiffrement propose une fonction de chiffrement. Cette fonction est alors soumise à l'expertise de la communauté cryptographique qui va tenter de mettre en évidence des faiblesses. Si ces faiblesses sont avérées, la fonction proposée est déclarée comme non-sûre. Éventuellement des corrections du schéma initial sont proposées. La fonction modifiée est à nouveau soumise aux attaques de la communauté, qui pourra encore découvrir de nouvelles faiblesses. Le processus de conception entre alors dans une boucle d'attaques-corrections qui peut être sans fin, montrant les limites de cette approche.

Au bout de combien de temps la sécurité soit-elle être acceptée ? trois ans ? cinq ans ?

Le chiffre de Vigenère, proposé au 16<sup>e</sup> siècle a été considéré comme indécryptable jusqu'au milieu du 19<sup>e</sup> siècle lorsque l'ancien officier prussien Friedrich Kasiski a publié la première attaque.

En novembre 1993, la version 1.5 du standard de chiffrement à clés publiques PKCS#1 est publié par l'entreprise *RSA Laboratories*. Cinq ans plus tard, en 1998, Daniel BLEICHENBACHER publie un article décrivant comment un adversaire peut déchiffrer un message de son choix par une attaque à cryptogrammes choisis.

Un système de chiffrement à clé publique, reposant sur le problème du sac à dos a été proposé par Benny CHOR et Ronald RIVEST en 1988 et a été cassé 10 ans plus tard, en 1998 par Serge VAUDENAY.

Ces exemples montrent l'insuffisance de l'approche classique et ont poussé les chercheurs à développer des arguments pour attester la sécurité d'un schéma cryptographique dès sa conception. Cela a conduit à la notion de **preuve de sécurité**.

Pour aboutir à ces preuves, les cryptologues revendiquent une approche scientifique, par une formalisation de l'adversaire, de son but et des moyens dont ils disposent pour atteindre son objectif.

## Un monde où les acteurs sont des algorithmes

La cryptographie est une activité qui répond à des besoins sociaux. Pour assurer la confiance dans le système de communication par ordinateurs, et garantir le respect du droit à une correspondance privée, les communications ont besoin d'être chiffrées. Rendre libre l'usage de la cryptographie est un des éléments qui a assuré la confiance dans la société numérique. Mais la protection des communications par leur chiffrement n'est plus assuré par les personnes elles-mêmes. C'est le système de communications qui s'en charge.

De nombreuses présentations en cryptographie désignent les correspondants sous les noms d'Alice et de Bob, dont les initiales sont *A* et *B*. Ces personnages virtuels sont apparus pour la première fois dans l'article de 1978 présentant le RSA. L'adversaire est souvent désigné par Ève, en référence au personnage biblique, mais aussi par homophonie avec le terme anglais *eavedropper*, qui désigne celui qui écoute de manière indiscrete. Mais dans le contexte théorique qui est l'objet de cette présentation, et contrairement à ce que laisse entendre ces appellations, les acteurs ne sont pas des sujets. Ils sont des algorithmes supposés résoudre des problèmes pour lesquels ils ont été conçus. L'émetteur du message est une algorithme qui choisit un message dans l'ensemble des messages possibles, et le chiffre avec un paramètre appelé la clé.

À l'autre bout de la ligne, le destinataire est l'algorithme qui reconstituera le message en clair à partir du cryptogramme et de la clé de déchiffrement.

Entre les deux, l'adversaire est un algorithme qui ne dispose pas de cette clé de déchiffrement, mais qui tentera de retrouver des informations sur le message en clair à partir du cryptogramme intercepté.

## Une sécurité devenue calculatoire

En dehors du masque jetable, et surtout en cryptographie asymétrique, la sécurité n'est plus inconditionnelle. Dès qu'un message dépasse la distance d'unicité, un adversaire tout puissant peut toujours explorer l'ensemble fini de toutes les clés possibles et ne retenir que l'unique clé qui conduit à un message compréhensible. Toutefois, les adversaires réels ne disposent pas en pratique de la puissance de calcul nécessaire pour mener cette recherche à son terme. La sécurité est devenue calculatoire. Elle repose sur des éléments quantitatifs de sécurité :

- Jusqu'à quelle puissance de calcul possédée par l'adversaire la sécurité est-elle assurée ?
- De quelle information en nature et en quantité a-t-il besoin pour résoudre le chiffre ?

La sécurité repose sur des éléments quantitatifs. L'adversaire peut disposer de sources auxiliaires d'informations, comme par exemple d'autres cryptogrammes dont le clair est connu. Ces hypothèses couvrent ce qui est raisonnable d'envisager en pratique.

Pour que la sécurité ne puisse pas être mise en doute, on se place généralement dans les conditions les plus favorables à l'adversaire. La sécurité sera avérée si une démonstration peut être faite que l'adversaire ne peut pas atteindre l'objectif qui lui est assigné malgré tous les moyens et toutes les informations mises à sa disposition.

Le but de l'adversaire dépend de la fonction cryptographique évaluée. Pour un schéma de chiffrement, ce sera de retrouver une information sur le message en clair, pour un schéma de signature, il s'agira de forger une fausse signature ou d'usurper une identité en s'authentifiant à la place d'un autre.

## Une réduction à des problèmes difficiles

La cryptographie ne peut être possible que si le problème de retrouver l'information manquante sans la clé est un problème en pratique insoluble. La construction de fonctions cryptographiques repose donc naturellement sur des problèmes difficiles.

Le chiffrement RSA repose sur l'hypothèse qu'il est difficile de factoriser les nombres entiers qui sont le produit de deux grands nombres premiers. L'algorithme de signature numérique repose sur l'hypothèse que le calcul du logarithme discret dans un groupe cyclique est difficile.

Il n'existe pas aujourd'hui de preuve que ces problèmes soient véritablement difficiles, même si cette difficulté est aujourd'hui largement admise. Les nombreux chercheurs en mathématiques qui se penchent depuis longtemps sur ces problèmes sans trouver de solution efficace en attestent. Ces problèmes sont des problèmes mathématiques généraux qui dépassent largement de cadre de la cryptologie.

La preuve de sécurité d'un schéma cryptographique réduit la sécurité du schéma à celle du problème mathématique sous-jacent. On cherchera à prouver que si un adversaire peut résoudre le chiffre, alors cette aptitude pourra être appliquée à résoudre le problème mathématique, contredisant l'hypothèse de sa difficulté.

Supposons par exemple qu'un mécanisme de chiffrement repose sur la difficulté de factoriser les entiers. Supposons également qu'un adversaire efficace soit en mesure de résoudre ce chiffre. Si on peut exhiber un algorithme de factorisation qui utilise cet adversaire comme sous programme, cela démontrera l'implication suivante :

*Si un adversaire existe alors la factorisation est un problème facile*

Par contraposition, on aura aussi démontré que :

*Tant que factoriser restera difficile, il ne pourra pas exister d'adversaire.*

Et finalement, la sécurité du schéma s'impose.

L'approche moderne de la conception des procédés cryptographiques repose sur un processus qui consiste à :

1. S'appuyer sur des primitives dont les propriétés sont admises, comme la difficulté de factoriser, ou de calculer un logarithme discret, ou encore sur l'existence de fonctions à sens unique.
2. Formaliser la cible de sécurité, l'adversaire, ses moyens, le type d'attaque.
3. Construire un algorithme cryptographique et apporter la preuve de sa sécurité.

Une fois cette preuve apportée, la sécurité du schéma s'impose, sans avoir recours à la boucle infinie d'attaques et de corrections de l'approche traditionnelle.

## La sécurité est une notion asymptotique

Les seules attaques réalistes ne peuvent reposer que sur des algorithmes dont la complexité en temps et en mémoire ne dépasse pas un polynôme de la taille des données d'entrée. De tels algorithmes sont qualifiés d'*efficaces*. Au delà d'une complexité polynomiale, l'attaque n'est pas réaliste. Il suffit d'augmenter légèrement la taille des paramètres pour la rendre inaccessible.

La sécurité s'appuiera sur la valeur d'un paramètre, appelé *paramètre de sécurité* qui sera par exemple la taille de la clé, ou le nombre de chiffres d'un nombre entier. Les résultats porteront sur le comportement des algorithmes lorsqu'on fait croître ce paramètre.



# Fonctions à sens unique

Les fonctions à sens unique constituent le socle sur lequel repose l'édifice cryptographique. À partir d'elles, on peut construire des générateurs pseudo aléatoires, composant essentiel du chiffrement à flot, ainsi que des fonctions et des permutations pseudo aléatoires qui sont les composants essentiels des procédés de chiffrement par blocs. Les fonctions à sens unique permettent aussi de construire des mécanismes de signature asymétriques.

## § 1.1 Fonction à sens unique concrète

De façon informelle, une fonction  $f$  est dite *à sens unique* s'il est facile de calculer la valeur pour tout paramètre, mais à l'opposé, étant donnée une valeur, il est difficile, en pratique impossible, de trouver un antécédent ayant cette valeur pour image. Finalement, les seules valeurs qu'on peut inverser sont celles qui ont été préalablement calculées. Lorsqu'on a calculé  $f(x) = y$ , on sait qu'un antécédent de  $y$  est  $x$ . Une fonction sera dite *à sens unique*, si on ne peut pratiquement pas en dire davantage.

L'impossibilité pratique de trouver un antécédent repose sur le fait qu'il n'existe pas d'algorithme applicable pour résoudre le problème. L'applicabilité de la méthode dépend de la puissance de calcul disponible. Il est aujourd'hui admis qu'un travail qui demanderait plus de  $2^{80}$  opérations élémentaires est impossible à mener à son terme. La quantité  $2^{80}$  vaut approximativement le nombre de dés à coudre contenus dans l'ensemble des océans terrestres. Lorsque l'algorithme DES a été défini en 1976, la taille des clés de 56 symboles binaires était suffisante pour empêcher la recherche exhaustive par force brutale. Les progrès réalisés dans la fabrication des ordinateurs ont rendu cette recherche aujourd'hui accessible, obligeant à adopter un standard plus robuste. La notion de fonction à sens unique concrète prend en compte les valeurs absolues des complexités.

### Définition 1.1 [ Fonction à sens unique concrète ]

Soient  $m$  et  $n$  deux entiers fixés. Pour trois réels positifs  $c$ ,  $t$  et  $\varepsilon$ , une fonction  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  est dite  $(c, t, \varepsilon)$ -à sens unique si :

1. la fonction  $f$  est facilement calculable par un algorithme de complexité inférieure ou égale à  $c$  ;
2. pour tout algorithme de complexité inférieure ou égale à  $t$  en temps et en espace, la probabilité qu'il trouve un antécédent  $t$  à un élément  $y = f(x)$ , obtenu à partir d'un élément  $x$  aléatoire de  $\{0, 1\}^n$ , ne dépasse pas  $\varepsilon$ .



La notion de fonction à sens unique est une notion probabiliste. Il sera toujours possible de trouver un antécédent aux valeurs qui déjà été calculées. La difficulté est exigée pour un élément aléatoire obtenu en calculant  $f(x)$  pour un élément  $x$  aléatoire tiré avec la probabilité uniforme dans l'ensemble de départ, et non pas un élément  $y$  aléatoire de l'ensemble d'arrivée.



## § 1.2 Attaques génériques

Les algorithmes d'attaques génériques sont ceux qui n'utilisent pas la structure particulière des fonctions  $f$ . Elles ne connaissent de la fonction  $f$  qu'une routine, une boîte noire, qui calcule  $f(x)$  pour une entrée  $x$  donnée. Ces attaques conduisent à des bornes sur les valeurs de  $c$ ,  $t$  et  $\varepsilon$ .

L'algorithme de la force brutale, ou force aveugle, consiste à faire un calcul exhaustif des valeurs et à s'arrêter une fois une valeur convenable trouvée. Cet algorithme conduit à une première borne.

### Proposition 1.2 [ Borne de la force brutale ]

Soit  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  une fonction  $(c, t, \varepsilon)$ -à sens unique, alors les paramètres  $t$  et  $\varepsilon$  vérifient :

$$\frac{t}{\varepsilon} \leq c \times 2^n.$$

*Preuve.* Pour prouver ce résultat, il suffit d'exhiber un algorithme de complexité  $t$  qui réussit avec une probabilité  $\geq \varepsilon$  à trouver un antécédent  $x$  à une valeur  $y$  aléatoire de l'image  $f(\{0, 1\}^n)$ .

Considérons l'algorithme suivant :

#### Algorithme de la force brutale

**Entrée :** une valeur  $y$  obtenue en calculant  $y = f(x)$  pour une valeur de  $x$  prise au hasard.

1. Pour  $i = 1$  jusqu'à  $k$  calculer  $k$  images  $y_i = f(x_i)$ , pour  $k$  valeurs  $x_i$  prises au hasard.
2. Si  $y$  vaut l'un des  $y_i$ , alors renvoyer le  $x_i$  correspondant.
3. Si aucun des  $y_i$  n'est égal à  $y$ , alors renvoyer une valeur  $x^*$  aléatoire.

La complexité de cet algorithme ne dépasse pas le nombre d'images calculées, donc  $t \leq c \times k$ .

Sa probabilité de succès  $\varepsilon$  est supérieure ou égale à la probabilité que  $y$  soit l'un des  $y_i$ . Il se peut aussi que la valeur  $x^*$  prise au hasard convienne. De toutes façons :

$$\frac{t}{c} \leq k \leq \varepsilon \times 2^n.$$

□



Prenons comme unité de calcul le temps pour calculer une valeur de  $f$ , soit  $c = 1$ . Pour atteindre un paramètre de sécurité de  $2^{80}$  pour le temps de calcul et de  $1/2^{60}$  pour la probabilité de succès, il faut que  $2^n \geq 2^{80+60}$ , soit  $n \geq 140$ .

En pratique, pour résister à l'attaque de la force brutale, une fonction à sens unique doit prendre des valeurs de plus de 140 symboles binaires.

Des attaques plus subtiles que la force brute sont apparues, en particulier, l'attaque par compromis temps-mémoire, introduite par Martin HELLMAN dans son article *A cryptanalytic time-memory trade-off*, publié en 1980. Ce résultat s'applique à des fonctions qui ont le même ensemble de départ et d'arrivée.

### Proposition 1.3 [ Borne du compromis temps-mémoire ]

Soit  $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$  une fonction  $(c, t, \varepsilon)$ -à sens unique, alors les paramètres  $c$ ,  $t$  et  $\varepsilon$  vérifient

$$\frac{t^2}{\varepsilon} \leq c^2 \times 2^n.$$

*Preuve.* Considérons l'algorithme suivant :

### Algorithme du compromis temps-mémoire

**Entrée :** une valeur  $y$  obtenue en calculant  $y = f(x)$  pour une valeur de  $x$  aléatoire.

**1. Précalcul.** Pour  $j = 1$  jusqu'à  $k$ , calculer les images  $z_j = f^k(x_j)$  pour  $k$  valeurs aléatoires  $x_1, \dots, x_k$ . Mémoriser les couples  $(x_j, z_j)$  dans une table.

**2. Itérations.** Pour l'entrée  $y$ , et pour  $i = 1$  jusqu'à  $k - 1$ , calculer les valeurs itérées  $y_j = f^i(y)$ .

**3.** Si l'un des  $y_i$  vaut l'un des  $z_j$  alors c'est qu'une certaine image itérée  $f^{k-j}(x_j)$  de  $x_j$  vaut  $y$ . Rendre l'image itérée précédente, qui est  $f^{k-j-1}(x_i)$ . Dans le cas contraire, rendre une valeur aléatoire  $x^*$ .

La complexité du précalcul est  $c \times k^2$  calculs de valeurs de  $f$ . Ces calculs ne sont effectués qu'une fois pour toutes. Pour évaluer la complexité de l'algorithme, on ne prend pas en compte cette phase.

Pour une valeur  $y$  donnée, le nombre de calculs de valeurs de  $f$  est inférieur à  $k$ . La complexité de l'algorithme vérifie donc :

$$t \leq c \times k.$$

L'algorithme réussit s'il existe un élément commun dans les listes  $(z_j)$  et  $(y_i)$ . La probabilité de succès est supérieure à la probabilité de trouver un élément commun dans deux listes de  $k$  éléments choisis parmi  $2^n$ . Le paradoxe des anniversaires indique que cette probabilité est asymptotiquement équivalente à  $\varepsilon = k^2/2^{n+1}$ , donc inférieur à  $k^2/2^n$ . On en déduit :

$$\begin{aligned} \frac{t^2}{c^2} &\leq k^2 \leq \varepsilon \times 2^n \\ \frac{t^2}{\varepsilon} &\leq c^2 \times 2^n \end{aligned}$$

□



Cette borne impose que, pour atteindre les paramètres de sécurité  $t \geq 2^{80}$  et  $\varepsilon \leq 1/2^{60}$ , la valeur de  $n$  doit vérifier  $2^n \geq 2^{2 \times 80 + 60} = 2^{220}$ , donc  $n \geq 220$ . En pratique, pour qu'une fonction à sens unique résiste à l'attaque du compromis temps-mémoire, ses valeurs doivent comporter plus de 220 symboles binaires.

## § 1.3 Des exemples pratiques

**La multiplication.** Si  $p$  et  $q$  sont des entiers premiers assez grands, calculer leur produit est facile, mais étant donné leur produit  $n = p \times q$ , les seuls algorithmes connus aujourd'hui pour retrouver les facteurs  $p$  et  $q$  ont une complexité sur-polynomiale. Le meilleur algorithme connu aujourd'hui (2017) est le *crible du corps de nombre* et sa complexité pour des entiers de  $k$  chiffres binaires est :

$$C = \exp\left(\left(\frac{64}{9}\right)^{1/3} k^{1/3} \ln(k)^{2/3}\right)$$

**La fonction puissance dans un corps fini.** Soit  $q$  une puissance d'un nombre premier assez grand. Pour tout élément générateur  $g$  du corps  $\mathbb{F}_q$  et tout entier  $\ell$  compris entre 1 et  $q - 1$ , il existe un algorithme efficace pour calculer  $g^\ell$ , par exemple par une succession de multiplications et d'élévation au carré. Par contre, étant donné un élément  $y$  de  $\mathbb{F}_q$ , on ne connaît pas aujourd'hui d'algorithme de complexité polynomiale pour trouver l'exposant  $\ell$  qui vérifie  $y = g^\ell$ . Ce problème est le *problème du logarithme discret*.

**La fonction RSA.** Soient  $p$  et  $q$  deux nombres premiers assez grand et soit  $n = p \times q$  leur produit. Soit  $e$  un entier premier avec  $\lambda(n) = \text{ppcm}(p-1, q-1)$ . La fonction qui à  $x \in \mathbb{Z}/n\mathbb{Z}$  associe  $x^e$  est aujourd'hui difficile à inverser sans la connaissance des facteurs  $p$  et  $q$  de  $n$ . Inverser cette fonction sans connaître la factorisation de  $n$  est le *problème RSA*.

**La fonction carré modulo le produit de deux nombres premiers.** Soient  $p$  et  $q$  deux nombres premiers assez grand et soit  $n = p \times q$  leur produit. La fonction carré modulo  $n$  est facile à calculer. Mais le calcul des racines carrées modulo  $n$  est aussi difficile que la factorisation de  $n$ .



Les deux derniers exemples, la fonction RSA et la fonction carré modulo un produit de deux nombres premiers, ont une propriété supplémentaire: si la factorisation du module est connue, ces fonctions deviennent inversibles en complexité polynomiale. Ce sont des *fonctions à sens unique avec porte dérobée*. Cette notion est au cœur du chiffrement à clé publique.

## § 1.4 Fonction à sens unique asymptotique

Définissons tout d'abord ce qu'est une fonction négligeable.

### Définition 1.4 [ fonction négligeable ]

Une fonction  $\mu : \mathbb{N} \rightarrow \mathbb{R}$  est négligeable si elle est, en valeur absolue, inférieure à l'inverse de tout polynôme à partir d'un certain rang, c'est-à-dire si pour tout polynôme  $P(x) \in \mathbb{R}[x]$ , on a :

$$\exists N \in \mathbb{N}, \quad \forall n \geq N, \quad |\mu(n)| \leq \frac{1}{|P(n)|}.$$

**Exemples.** La fonction  $n \mapsto \frac{1}{\sqrt{2^n}}$  est négligeable, la fonction  $n \mapsto \frac{1}{n^k}$  ne l'est pas.



Une fonction non négligeable est une fonction qui est asymptotiquement supérieure à l'inverse d'un polynôme. L'ensemble des fonctions non négligeables est stable par addition, par multiplication et par élévation à toute puissance.

Si  $\mu$  n'est pas une fonction négligeable, alors pour tout entier naturel  $k$ , la fonction  $n \mapsto \mu(n)^k$  n'est pas non plus négligeable.

L'ensemble  $\{0, 1\}^*$  est par définition le monoïde libre sur l'alphabet  $\{0, 1\}$ , c'est-à-dire l'ensemble des mots de longueur quelconque, y compris le mot vide noté  $\varepsilon$ , écrits avec des 0 et des 1.

Pour définir les fonctions à sens unique asymptotique, la taille des paramètres et des valeurs n'est pas fixée. Ce sont des mots de longueur quelconque.

### Définition 1.5 [ Fonction à sens unique asymptotique ]

Une fonction  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  est dite à sens unique si les deux conditions suivantes sont remplies :

1. La fonction  $f$  est facile à calculer, c'est-à-dire il existe un algorithme efficace qui, pour tout élément  $x \in \{0, 1\}^*$  calcule  $f(x)$ .
2. La fonction  $f$  est difficile à inverser, c'est-à-dire, pour tout algorithme  $A$  qui s'exécute en temps polynomial en  $n$ , la probabilité de trouver un antécédent de taille  $n$  d'un élément aléatoire  $y$  de l'image de  $f$  est une fonction négligeable en  $n$ .

On exprime la seconde condition en énonçant que pour tout algorithme efficace  $A$ , la fonction :

$$(1) \quad n \mapsto \Pr_{x \in_R \{0, 1\}^n} [A(1^n, f(x)) = x' \text{ et } f(x) = f(x')],$$

est une fonction négligeable.



L'élément  $y$  de l'ensemble d'arrivée dont il faut trouver un antécédent est obtenu en tirant une valeur aléatoire  $x$  dans l'ensemble de départ avec la probabilité uniforme et en calculant  $y = f(x)$ .

Quelques explications sur les notations de l'écriture de l'équation (1) :

- La notation  $x \in_R \{0, 1\}^n$  signifie que la probabilité est exprimée lorsque  $x$  est un élément choisi aléatoirement avec la probabilité uniforme dans l'ensemble  $\{0, 1\}^n$ .
- Le paramètre  $1^n$  de l'algorithme  $A$  est un artifice technique pour imposer un paramètre de taille  $n$  à l'algorithme  $A$ . L'algorithme  $A$  est supposé efficace, c'est-à-dire qu'il a une complexité polynomiale en la taille des données. Avec ce paramètre, la complexité de  $A$  est bornée par un polynôme en  $n$ . Sans ce paramètre, si par exemple la taille de  $f(x)$  est en  $\log(n)$ , la complexité de  $A$  serait simplement bornée par un polynôme en  $\log(n)$ , ce qui change complètement la complexité supposée de l'algorithme  $A$ .



Comme le calcul de la valeur d'une fonction à sens unique est d'une complexité polynomiale, retrouver un antécédent d'une fonction à sens unique est un problème  $\mathcal{NP}$ . En conséquence, une fonction à sens unique ne peut exister que si  $\mathcal{P} \neq \mathcal{NP}$ .



On ne sait pas si les fonctions à sens unique existent vraiment. Leur existence est vraisemblable, mais on ne peut pas la prouver. La seule chose qu'on puisse dire actuellement est que si une fonction à sens unique existe, alors  $\mathcal{P} \neq \mathcal{NP}$ .

La réciproque n'est pas connue non plus. Un problème est  $\mathcal{NP}$  complet s'il est difficile à résoudre dans le pire des cas. La définition des fonctions à sens unique stipule qu'elles sont difficiles à inverser dans le cas moyen. Même si on savait qu'il existe un problème  $\mathcal{NP}$  qui n'est pas  $\mathcal{P}$ , le problème de l'existence de fonction à sens unique resterait ouvert.

La notion de fonction à sens unique est commode pour traiter des résultats théoriques, mais en pratique, on a plutôt affaire à des familles de fonctions à sens unique, comme par exemple la fonction puissance qui opère sur un groupe  $\mathbb{Z}/p\mathbb{Z}$  avec une taille donnée pour  $p$ .

Soit  $p$  un nombre premier. Pour tout entier  $n \in \{0, \dots, p-1\}$  et un générateur  $g$  de  $\mathbb{Z}/p\mathbb{Z}$ , il existe un algorithme efficace pour calculer  $g^n$ , par exemple par une succession de multiplications et d'élevations au carré.

Inversement, étant donné un élément  $y$  de  $\mathbb{Z}/p\mathbb{Z}$ , on ne connaît actuellement pas d'algorithme de complexité polynomiale en la taille de  $y$  pour trouver l'exposant  $n$  qui vérifie  $y = g^n$ . Ce problème est le problème du *logarithme discret*.

## § 1.5 Famille de fonctions à sens unique

La fonction puissance est en pratique une fonction à sens unique. Comme la fonction est différente pour chaque taille de paramètre, on a affaire à une *famille de fonctions à sens unique*. Nous sommes donc amené à définir cette dernière notion.

De plus, la notion de *fonction à sens unique asymptotique* est plus commode pour démontrer des résultats de sécurité, mais la notion de *famille de fonctions à sens unique* correspond mieux aux fonctions utilisées en pratique.

### Définition 1.6 [ Famille de fonctions à sens unique ]

Soit  $I$  un ensemble d'indices inclus dans  $\{0, 1\}^*$ . Une famille de fonctions  $(f_i)_{i \in I}$  de fonctions  $D_i \rightarrow \{0, 1\}^*$  est une famille de fonctions à sens unique si les deux conditions suivantes sont remplies :

1. La famille est facile à échantillonner, c'est-à-dire :
  - a) Il existe un algorithme probabiliste  $S_1$ , prenant en paramètre un entier  $n$ , de complexité polynomiale en  $n$  qui génère un élément  $i$  aléatoire dans  $I$ , tel que les éléments de l'ensemble  $D_i$  ont une taille de  $n$  symboles binaires.
  - b) Il existe un algorithme probabiliste efficace  $S_2$ , qui à partir d'un indice  $i$  de  $I$ , génère un élément aléatoire  $x$  du domaine  $D_i$ .
2. Les fonctions  $f_i$  sont faciles à calculer, c'est-à-dire il existe un algorithme efficace  $A$  qui, à partir d'un élément  $i \in I$  et d'un élément  $x \in D_i$  calcule  $f_i(x)$ .
3. Les fonctions  $f_i$  sont difficiles à inverser, ce qui signifie que la probabilité de succès de tout algorithme efficace qui prend en entrée un élément  $i$  produit par l'algorithme  $S_1$  et un élément  $y$  égal à  $f_i(x)$ , où  $x$  est un résultat de l'algorithme  $S_2$ , pour trouver un antécédent de  $y$  par  $f_i$ , est négligeable en la taille de l'indice  $i$  et de l'élément  $y$ .



La notion de famille de fonction à sens unique est plus proche des fonctions qu'on utilise en cryptologie réelle, comme le produit d'entiers, la fonction puissance ou la fonction RSA. À l'opposé, la notion de fonction à sens unique asymptotique, donnée par la définition 1.5, est plus théorique et est plus commode pour manipuler les preuves de sécurité. Nous verrons que ces deux notions sont équivalentes.



Dans les applications cryptologiques, en particulier à clé publique, il faut penser l'indice  $i$  comme étant la clé publique qui permet le chiffrement. L'inversion de la fonction est réservée aux détenteurs de la clé privée correspondantes.



Un algorithme probabiliste est un algorithme qui produit un résultat aléatoire dans l'ensemble des valeurs possibles. On peut considérer qu'un tel algorithme est un algorithme déterministe qui a un argument supplémentaire constitué d'une chaîne aléatoire  $r \in \{0, 1\}^*$  en fonction de laquelle l'élément aléatoire est produit.

EXEMPLE : Une famille de fonctions à sens unique construite à partir de la multiplication.

- L'ensemble des indices est l'ensemble  $\mathbb{N}$  des entiers naturels.
- Pour un indice  $i \in \mathbb{N}$ , le domaine  $D_i$  est l'ensemble des couples de nombres premiers de  $i$  chiffres en numération binaire.
- Pour un entier  $n \in \mathbb{N}$ , l'algorithme  $S_1$  renvoie  $i = n$ .
- L'algorithme  $S_2$  génère deux nombres premiers  $p$  et  $q$  de taille  $i$ . Pour générer un nombre premier de taille  $i$ , on génère un entier aléatoire, puis on teste sa primalité à l'aide d'un test probabiliste de type MILLER-RABIN, ou bien à l'aide du test AKS qui est déterministe et de complexité polynomiale.
- Pour  $i \in I$ , la fonction  $f_i$  est  $(p, q) \mapsto p \times q$ .

Trouver un antécédent de  $f_i$  revient à factoriser un entier qui est le produit de nombre premiers de tailles similaires.

Le théorème suivant énonce que la notion de fonction à sens unique asymptotique et la notion de famille de fonctions à sens unique sont équivalentes, ce qui permettra de considérer l'un ou l'autre cas selon ce qui est le plus commode.

### Théorème 1.7

Il existe une fonction à sens unique si et seulement si il existe une famille de fonctions à sens unique.

*Preuve.* La preuve de ce théorème est constructive. Supposons tout d'abord qu'il existe une fonction à sens unique  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  et construisons à partir de  $f$  une famille de fonctions à sens unique.

L'ensemble d'indices  $I$  est  $I = \{0, 1\}^*$  l'ensemble des mots binaires.

Pour un mot binaire  $i$  donné dans  $I$ , on définit le domaine  $D_i$  comme étant l'ensemble des mots binaires de la même longueur que  $i$ , c'est-à-dire  $D_i = \{0, 1\}^{|i|}$ .

L'algorithme  $S_1$  choisit un mot aléatoire binaire de longueur  $n$ .

L'algorithme  $S_2$  choisit un mot aléatoire binaire de longueur  $|i|$ .

Pour tout  $i \in I$ , la fonction  $f_i$  est :

$$f_i : \begin{array}{ccc} D_i & \longrightarrow & \{0, 1\}^* \\ x & \longmapsto & f(x) \end{array}$$

Comme la fonction  $f$  est à sens unique, toutes les fonctions  $f_i$  sont à sens unique, car dans le cas contraire, on en déduirait immédiatement un algorithme qui inverse la fonction  $f$  sur une entrée de longueur  $n$ , en trouvant un antécédent de  $f_i$  avec la même probabilité de succès.

Ceci montre donc l'existence d'une famille de fonctions à sens unique  $(f_i)_{i \in I}$ .

Réciproquement, à partir d'une famille de fonctions à sens unique, construisons une fonction à sens unique  $f$ .

Rappelons qu'un algorithme qui rend un résultat aléatoire peut se modéliser comme un algorithme déterministe qui prend en paramètre un mot binaire aléatoire servant de germe à la génération de l'aléa nécessaire à son calcul.

Pour un élément  $z$  de  $\{0, 1\}^*$ , on définit la valeur de  $f(z)$  ainsi :

On coupe le mot binaire  $z$  en deux mots disjoints de taille convenable pour alimenter respectivement  $S_1$  et  $S_2$  en chaîne aléatoire. Soit  $x = (r, s)$ .

On utilise le premier terme  $r$  comme chaîne aléatoire pour l'algorithme  $S_1$  de la famille de fonctions à sens unique, et on utilise la deuxième moitié  $s$  comme chaîne aléatoire l'algorithme  $S_2$ . Ces algorithmes fournissent de manière déterministe un indice  $i = S_1(r) \in I$ , puis un élément  $x = S_2(s, i) \in D_i$ . On pose finalement :

$$f(r, s) = (f_i(x), i).$$

Il reste à montrer que la fonction  $f$  est à sens unique. Pour cela, en vue d'une contradiction, supposons le contraire. Cela signifie que sur une entrée  $(y, i) \in \{0, 1\}^* \times I$ , il existe un algorithme efficace qui trouve un élément  $(r, s)$  vérifiant  $i = S_1(r)$  et  $f(r, s) = f_i(y)$ . Si on pose  $x = S_2(s, i)$ , qui est calculable en temps polynomial, on a trouvé un antécédent  $x$  de  $y$  par  $f_i$ , ce qui montre que sous cette hypothèse, la famille  $(f_i)$  n'est pas à sens unique.

□

## § 1.6 Exercices

**1.1** Démontrer que la fonction carré modulo un produit de deux nombres premiers est une fonction à sens unique concrète si et seulement si la multiplication de deux nombres premiers est une fonction à sens unique concrète.



Le système de chiffrement à clé publique de RABIN consiste à élever au carré un message modulo un entier égal au produit de deux nombres premiers de même taille. Le déchiffrement consiste à extraire une racine carrée du cryptogramme. Une redondance dans le clair permet de choisir entre les quatre racines carrées possibles. Cette propriété montre que tant que la factorisation des entiers est un problème difficile, alors le chiffrement de RABIN est sûr contre une attaque à clair choisi.

**1.2** Formaliser précisément la famille de fonctions supposée à sens unique reposant sur :

- la fonction puissance modulo un nombre premier.
- la fonction RSA.
- la fonction carré modulo le produit de deux nombres premiers.

**1.3** Montrer que si  $\mathcal{P} \neq \mathcal{NP}$ , alors il peut exister des fonctions qui sont difficiles à inverser dans le pire des cas, mais qui ne sont pas des fonctions à sens unique.

**1.4** Une fonction  $\{0, 1\}^* \rightarrow \{0, 1\}^*$  est dite à *longueur régulière* si

$$\forall x, y \in \{0, 1\}^*, \quad |x| = |y| \implies |f(x)| = |f(y)|$$

Montrer que s'il existe une fonction à sens unique, alors il existe une fonction à sens unique à longueur régulière.

**1.5** On dit qu'une fonction  $\{0, 1\}^* \rightarrow \{0, 1\}^*$  conserve les longueurs si, pour tout  $x \in \{0, 1\}^*$ , on a  $|f(x)| = |x|$ . Montrer que s'il existe une fonction à sens unique, alors il existe une fonction à sens unique qui conserve les longueurs.

**1.6** Soit  $f : E \rightarrow E$  une fonction à sens unique.

- La fonction  $f \circ f$  est-elle à sens unique ?
- La fonction  $g : x \mapsto (f(x), f \circ f(x))$  est-elle à sens unique ?
- On suppose maintenant que  $f$  est bijective. La fonction  $f \circ f$  est-elle à sens unique ?

**1.7** Soit  $f$  une fonction à sens unique  $\{0, 1\}^* \rightarrow \{0, 1\}^*$ .

Soit  $\varphi : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \{0, 1\} \times \mathbb{N}$   
 $(x, i) \mapsto (f(x), x_i, i)$ .

- Montrer que la fonction  $\varphi$  est à sens unique.
- Lorsqu'un algorithme renvoie une valeur binaire, on appelle *avantage* de l'algorithme la quantité  $A = 2p - 1$ , où  $p$  est la probabilité qu'il renvoie la bonne valeur. Trouver un algorithme  $A_j$  qui, à partir d'une valeur  $(y, x_i, i)$  obtenue aléatoirement, trouve la valeur  $x_j$  avec un avantage non négligeable.

TRAVAUX PRATIQUES. Implémenter une famille de fonctions à sens unique (produit de deux nombres premiers, fonction RSA, fonction d'élevation au carré, ou exponentielle), puis la fonction à sens unique correspondante.

## Prédicat difficile

Si une fonction est à sens unique, c'est qu'elle masque de l'information sur l'antécédent d'une valeur donnée. Un *prédicat difficile* (*Hard Core Predicate*) d'une fonction est une information binaire sur l'entrée qu'il est difficile de retrouver à partir de la valeur de la fonction. Le minimum qu'on puisse exiger d'une fonction à sens unique est l'existence d'une information calculatoirement inaccessible à partir de la valeur. Ce chapitre montre l'existence d'un prédicat dur caractérise les fonctions à sens unique. Ce résultat est au cœur de la théorie cryptographique.

### § 2.1 Définition

Comme le montre l'exercice I.8 page 16, il existe des fonctions à sens unique qui ne masquent aucun des symboles binaires de l'antécédent. La condition de masquer les composantes de l'antécédent ne suffit donc pas pour caractériser les fonctions à sens unique. Le prédicat difficile d'une fonction peut être une fonction booléenne plus générale de l'entrée.

#### Définition 2.1 [ Prédicat difficile ]

Soit  $f$  une fonction  $f : \{0,1\}^* \rightarrow \{0,1\}^*$ , une fonction booléenne  $p : \{0,1\}^* \rightarrow \{0,1\}$  est un *prédicat dur* pour  $f$  si

1. La valeur de  $p$  est calculable en temps polynomial.
2. Tout algorithme polynomial  $A$  ne peut calculer  $p(x)$  à partir de  $f(x)$  qu'avec un avantage négligeable, c'est-à-dire, pour tout entier  $n$ ,

$$\text{Av}_n(A) = |2 \Pr_{x \in_R \{0,1\}^n} [A(1^n, f(x)) = b(x)] - 1|$$

est une fonction négligeable en  $n$ .



L'exemple de l'exercice I.8 page 16 montre qu'il existe des fonctions à sens unique telles qu'aucune fonction  $x \mapsto x_i$  ne soit un prédicat dur.

### § 2.2 Théorème de Goldreich-Levin

Le résultat de l'exercice II.3 page 23 montre qu'une fonction injective qui a un prédicat dur est nécessairement à sens unique. Le théorème de GOLDREICH-LEVIN énonce réciproquement qu'à partir d'une fonction à sens unique, on peut construire une autre fonction à sens unique qui lui est similaire et qui a un prédicat dur. Ce résultat caractérise finalement les fonctions à sens unique comme étant celles qui admettent un prédicat dur.



**Théorème 2.2** [ GOLDREICH-LEVIN ]

Soit  $f$  une fonction à sens unique. Soit  $g$  la fonction définie à partir de  $f$  par :

$$g : (x, r) \mapsto (f(x), r),$$

avec  $|x| = |r|$ , alors la fonction booléenne  $(x, r) \mapsto x \cdot r$  est un prédicat dur pour  $g$ , le produit scalaire  $x \cdot r$  étant défini par  $x \cdot r = \sum_{i=1}^n x_i r_i$  (modulo 2)

*Preuve.* Supposons pour une contradiction que la fonction  $g$  n'est pas à sens unique. Il existe alors un algorithme efficace qui permet de trouver un antécédent  $(x, r)$  à partir d'une valeur  $(y, r)$ , et cet algorithme permet en particulier de trouver un antécédent  $x$  de  $y$  par  $f$ . Ceci montre que  $f$  n'est alors pas à sens unique, ce qui contredit l'hypothèse.

On suppose pour une contradiction qu'il existe un algorithme  $B$ , capable de trouver  $x \cdot r$  avec une probabilité non négligeable.

Supposons dans un premier temps que  $B$  trouve  $x \cdot r$  avec certitude, c'est-à-dire avec une probabilité égale à 1 à partir d'une valeur  $(y, r)$ . Cet algorithme, appliqué successivement avec  $r$  égal aux éléments de la base canonique de  $\{0, 1\}^n$ , renvoie les composantes successives de  $x$ .

Si maintenant, l'algorithme  $B$  rend une réponse non certaine, par exemple avec une probabilité non négligeable, égale par exemple à  $\frac{1}{2} + \frac{1}{n}$ , alors la méthode ci-dessus ne permet pas de conclure, car il rend une bonne réponse avec une probabilité  $\left(\frac{1}{2} + \frac{1}{n}\right)^n = \left(\frac{1}{2}\right)^n \left(1 + \frac{1}{2n}\right)^n$  qui devient négligeable. Il faut trouver une autre idée.

Cette idée est de construire un algorithme  $A$  qui trouve  $x$  avec une probabilité non négligeable en faisant une hypothèse sur  $\ell$  bits de  $x$ . Si ces  $\ell$  bits sont choisis au hasard, la probabilité qu'ils soient justes est en  $1/2^\ell$ , mais si  $\ell$  est assez petit devant  $n$ , la probabilité de succès restera non négligeable.

Soit donc  $B(y, r)$  un algorithme qui, pour tout entier  $n$ , à partir de  $y = f(x)$ , pour un élément  $x \in_R \{0, 1\}^n$  aléatoire, et un élément  $r \in_R \{0, 1\}^n$  aléatoire également, trouve  $b(x, r) = x \cdot r$  à avec une complexité bornée par un polynôme en  $n$  et réussit avec une probabilité non négligeable en  $n$ . Notons :

$$(2) \quad \Pr_{x, r \in_R \{0, 1\}^n} [B(f(x), r) = x \cdot r] \geq \frac{1}{2} + \varepsilon(n).$$

Construisons un algorithme  $A$  qui utilise  $B$  comme sous-programme, efficace en  $n$  et qui inverse  $f$ , c'est-à-dire trouve  $x$ , avec une probabilité non négligeable.

L'algorithme  $A$  s'appuiera sur  $\ell$  valeurs supposées connues  $b_1 = x \cdot r_1, \dots, b_\ell = x \cdot r_\ell$ , pour  $\ell$  valeurs aléatoires de l'élément  $r \in \{0, 1\}^n$ .

Pour tout sous-ensemble  $I$  de  $\{1, \dots, \ell\}$ , posons  $r_I = \sum_{i \in I} r_i$  (modulo 2). Par bilinéarité du produit scalaire, on a  $x \cdot r_I = \sum_{i \in I} x \cdot r_i$  (modulo 2). Finalement, la connaissance de  $\ell$  bits permet de connaître  $2^\ell$  valeurs.

La valeur de  $r$  étant connue, pour tout indice  $i \in \{1, \dots, n\}$ , la valeur de  $x_i$  se déduit de celle de  $x \cdot (r + e^i)$ , par :

$$x \cdot r + x \cdot (r + e^i) = x \cdot e^i = x_i,$$

où  $e^i$  est le vecteur de  $\{0, 1\}^n$  dont la seule composante non nulle est la  $i^e$  qui vaut 1.

Considérons l'algorithme  $A$  qui, sur une entrée  $y = f(x)$ , pour  $x$  aléatoire dans  $\{0, 1\}^n$ , fait appel à l'algorithme  $B(y, r_I + e^i)$  pour tous les sous-ensembles non vides de  $\{1, \dots, \ell\}$  pour estimer la valeur de  $x_i$ . La valeur de  $r_I \cdot x_i$  étant incertaine et afin d'amplifier la probabilité de succès, la valeur de

$x_i$  est estimée par un vote majoritaire pour toutes les valeurs  $x \cdot r_I + B(y, r_I + e^i)$ . Précisément, posons  $b_{i,I}$  la réponse de  $B(y, r_I + e^i)$ . Soit  $x_{i,I} = x \cdot r_I + b_{i,I}$  la valeur estimée de  $x_i$  à partir de la réponse  $b_{i,I}$  de l'algorithme  $B$ . Soit  $m = 2^\ell - 1$  le nombre de sous-ensembles  $I$  non vides de  $\{1, \dots, \ell\}$ . Il s'agit du nombre d'appels à l'algorithme  $B$ . L'estimation de  $x_i$  par vote majoritaire est

$$x_i^* = \begin{cases} 0 & \text{si } \sum_I x_{i,I} < m/2; \\ 1 & \text{si } \sum_I x_{i,I} \geq m/2. \end{cases}$$

Pour montrer cet algorithme convient pour inverser  $f$  avec une probabilité non négligeable, il reste à évaluer sa probabilité de succès et sa complexité.

L'algorithme  $A$  fait appel à l'algorithme  $B$  sur des valeurs aléatoires de  $r$ , mais avec toujours la même valeur de  $y = f(x)$  à inverser. La probabilité de succès de  $B$  dans ces conditions ne peut pas se déduire directement de la relation (2), car cette valeur est donnée pour une entrée  $y$  aléatoire. Le lemme suivant majore la probabilité de succès de  $B(y, r)$  lorsque qu'il est appelé avec une valeur  $y$  fixée.

### Lemme 2.3

Soit  $x$  un élément fixé de  $\{0, 1\}^n$ . Considérons l'évènement  $S(x)$  suivant : « L'algorithme  $B(f(x), r)$  trouve le bit difficile  $x \cdot r$  avec une probabilité non négligeable au moins égale à  $1/2 + \varepsilon(n)$ . » Lorsque  $x$  est choisi au hasard dans  $\{0, 1\}^n$ , la probabilité de l'évènement  $S(x)$  est non négligeable et supérieure à  $\varepsilon(n)/2$ .

*Preuve.* Soit  $S_n$  l'ensemble des vecteurs  $x$  pour lesquels l'algorithme  $B(f(x), r)$  réussit avec une probabilité non négligeable supérieure à  $1/2 + \varepsilon(n)/2$ . Montrons que le cardinal de l'ensemble  $S_n$  satisfait  $|S_n| \geq 2^n \times \varepsilon(n)/2$ . Pour cela, supposons le contraire en vue d'une contradiction. Dans ce cas, on a

$$\begin{aligned} \Pr_{x,r}(B(f(x), r) = x \cdot r) &= \underbrace{\Pr_r(B(f(x), r) = x \cdot r \mid x \in S_n)}_{\leq 1} \underbrace{\Pr(x \in S_n)}_{< \varepsilon(n)/2} \\ &\quad + \underbrace{\Pr_r(B(f(x), r) = x \cdot r \mid x \notin S_n)}_{< 1/2 + \varepsilon(n)/2} \underbrace{\Pr(x \notin S_n)}_{\leq 1}, \\ &< \frac{\varepsilon(n)}{2} + \frac{1}{2} + \frac{\varepsilon(n)}{2} = \frac{1}{2} + \varepsilon(n) \end{aligned}$$

ce qui est contradictoire avec l'hypothèse (2). Cette inégalité sur le cardinal de l'ensemble  $S_n$  montre que la probabilité de l'évènement  $S(x)$  vaut au moins  $\varepsilon(n)/2$ .  $\square$

### Lemme 2.4

Si l'entier  $\ell$  satisfait  $2^\ell \leq n/\varepsilon(n)^2$ , et sous l'hypothèse que les  $\ell$  valeurs  $b_1, \dots, b_\ell$  de  $x \cdot r_1, \dots, x \cdot r_\ell$  sont exactes, la probabilité de succès de l'algorithme  $A$  est asymptotiquement supérieure à  $1/\sqrt{e} \approx 0,6$ .

*Preuve.* Soit  $X_{i,I}$  la variable aléatoire qui vaut 1 si l'estimation  $x_{i,I}$  est fautive et qui vaut 0 sinon, c'est-à-dire  $X_{i,I} = x_i + x_{i,I}$  (modulo 2). D'après le lemme 2.3, on a :

$$\Pr(X_{i,I} = 0) \geq \frac{1}{2} + \frac{\varepsilon(n)}{2}.$$

Par conséquent, l'espérance de la variable  $X_{i,I}$  satisfait  $E(X_{i,I}) \geq 1/2 + \varepsilon(n)/2$ . Posons  $X_i$  la variable aléatoire égale au nombre d'estimations fautes faites pour chacun des sous-ensembles non vides  $I$  de  $\{1, \dots, \ell\}$ , c'est-à-dire :

$$X_i = \sum_{\substack{I \subset \{1, \dots, \ell\} \\ I \neq \emptyset}} X_{i,I}.$$

Par linéarité de l'espérance, on a :

$$(3) \quad E(X_i) \geq \frac{m}{2} + m \frac{\varepsilon(n)}{2},$$

donc :

$$X_i - E(X_i) \leq X_i - \frac{m}{2} - m \frac{\varepsilon(n)}{2}.$$

Or, en raison de la façon d'estimer la composante  $x_i$  du vecteur  $x$ , la probabilité que  $x_i^*$  soit différent de  $x_i$  est exactement la probabilité que  $X_i$  soit supérieure à  $m/2$ . En raison de l'inégalité (3), on a :

$$\begin{aligned} \Pr(x_i \neq x_i^*) &= \Pr\left(X_i \geq \frac{m}{2}\right) \leq \Pr\left(X_i - E(X_i) \leq -m \frac{\varepsilon(n)}{2}\right) \\ &\leq \Pr\left(|X_i - E(X_i)| \geq m \frac{\varepsilon(n)}{2}\right) \end{aligned}$$

D'après l'inégalité de CHEBYCHEV, on a :

$$\Pr(x_i \neq x_i^*) \leq \frac{4 \text{Var}(X_i)}{m^2 \varepsilon(n)^2}.$$

Comme pour deux sous-ensembles distincts de  $\{1, \dots, \ell\}$  il existe nécessairement un élément qui appartient à l'un sans appartenir à l'autre, les variables  $X_{i,I}$  sont deux-à-deux indépendantes. La variance de  $X_i$  est donc la somme des variances des  $X_{i,I}$  :

$$\text{Var}(X_i) = \sum_{\substack{I \subset \{1, \dots, \ell\} \\ I \neq \emptyset}} \text{Var}(X_{i,I}).$$

Or :

$$\text{Var}(X_{i,I}) = E(X_{i,I}^2) - E(X_{i,I})^2.$$

Mais comme la variable  $X_{i,I}$  prend ses valeurs dans l'ensemble  $\{0, 1\}$ , elle est égale à son carré, d'où :

$$\text{Var}(X_{i,I}) = E(X_{i,I}) - E(X_{i,I})^2.$$

Pour majorer  $\text{Var}(X_{i,I})$ , étudions la fonction  $x \mapsto x - x^2 = x(1 - x)$ . Elle a son maximum qui vaut  $1/4$  en  $1/2$ , par conséquent :

$$\text{Var}(X_{i,I}) \leq \frac{1}{4},$$

et donc

$$\text{Var}(X_i) \leq \frac{m}{4}$$

Par conséquent,

$$\Pr(x_i \neq x_i^*) \leq \frac{1}{m \varepsilon(n)^2}$$

Lorsque l'entier  $\ell$  satisfait  $2^\ell \leq n/\varepsilon(n)^2$ , on a  $m = 2^\ell - 1$  donc  $2m > 2^\ell$  et donc :

$$\Pr(x_i \neq x_i^*) \leq \frac{1}{2n},$$

ce qui signifie que l'algorithme  $A$  calcule correctement le bit  $x_i$  avec une probabilité supérieure à  $1 - 1/2n$ . L'ensemble des bits de  $x$  est donc calculé avec une probabilité supérieure à  $\left(1 - \frac{1}{2n}\right)^n$ , et cette quantité converge vers  $1/\sqrt{e}$ . L'algorithme  $A$  calcule le vecteur  $x$  avec une probabilité non négligeable, asymptotiquement minorée par une constante.  $\square$

*Fin de la preuve du théorème de GOLDREICH-LEVIN.* Montrons maintenant que l'algorithme  $A$  trouve un antécédent de  $y = f(x)$  avec une probabilité non négligeable. Soit  $C$  l'évènement « les  $\ell$  valeurs estimées  $b_1, \dots, b_\ell$  sont correctes, et  $S(x)$  l'évènement L'algorithme  $B(f(x), r)$  trouve le bit difficile  $x \cdot r$  avec une probabilité non négligeable au moins égale à  $1/2 + \varepsilon(n)$ . ». Ces évènements sont indépendants. Soit  $R$  l'évènement qui énonce le succès de l'algorithme  $A$ . On a :

$$\Pr(R) \geq \Pr(R \mid C, S(x)) \times \Pr(C \cap S(x)).$$

La probabilité de l'évènement  $C$  vaut  $1/2^\ell \geq \frac{\varepsilon(n)}{n}$ . La probabilité de l'évènement  $S(x)$  est supérieure à  $\varepsilon(n)$ , et la probabilité de succès de  $A$  sachant que les évènements  $C$  et  $S(x)$  sont réalisés est supérieure à  $1/\sqrt{e}$ . Il en résulte que :

$$\Pr_{x \in_R \{0,1\}^n}(R) \geq \frac{1}{\sqrt{e}} \times \frac{\varepsilon(n)^3}{n},$$

qui est non négligeable.

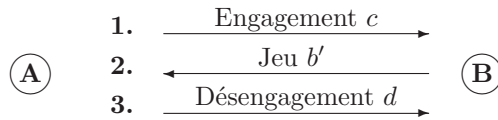
L'algorithme  $A$  fait  $n \times 2^\ell$  appels à l'algorithme  $B$ . Si l'algorithme  $B$  est efficace, alors l'algorithme  $A$  l'est aussi.

□

## § 2.3 Applications

### 2.3.1. Mise en gage

L'illustration typique du protocole de mise en gage d'information est le jeu de pile ou face par téléphone. Si les partenaires ne se font pas confiance, il doivent convenir d'un protocole qui interdira de tricher.



1. Le joueur **A** choisit un symbole binaire  $b$ , égal à 0 ou 1, et l'engage par le calcul de deux valeurs :
  - une valeur d'engagement (*commit*)  $c$ , qu'il transmet à **B** lors de la première passe.
  - une valeur de désengagement  $d$ , qu'il conserve pour plus tard.
2. Le joueur **B** joue une valeur binaire  $b'$  et la transmet à **A** lors de la deuxième passe.
3. Le joueur **A** transmet la valeur du désengagement  $d$  qui va révéler la valeur de son choix  $b$  au joueur **B**.

Pour mettre en œuvre ce jeu, deux fonctions sont requises :

- Une fonction d'engagement *com*, qui à partir d'un bit  $b$ , calcule un couple  $(c, d)$  d'engagement de désengagement.
- Une fonction d'ouverture *open*, qui à partir du couple  $(c, d)$  d'engagement et de désengagement, révèle la valeur du bit  $b$ .

Pour empêcher la triche, deux propriétés de sécurité sont définies :

- La propriété d'enchérissement (*biding*) : la probabilité de trouver deux valeurs de désengagement  $d$  et  $d'$  est négligeable. Cette propriété signifie qu'une fois l'engagement émis, il n'est pas possible de revenir dessus pour changer le résultat. En d'autres termes, il n'existe pas d'algorithme efficace qui calcule un triplet  $(c, d, d')$  tel que  $d \neq d'$  et  $open(c, d) = open(c, d')$ .

- La propriété de discrétion (*hiding*) : la connaissance de  $c$  ne révèle aucune information sur la valeur  $b$  choisie. En d'autres termes, il n'existe pas d'algorithme efficace qui permet de trouver la valeur de  $b$  à partir de celle de  $a$ .

Un bit difficile d'une fonction à sens unique permet de satisfaire la propriété de discrétion. Si de plus, cette fonction à sens unique est choisie injective, la propriété d'enchérissement sera assurée.

### 2.3.2. Preuve sans divulgation

L'existence de fonction à sens unique injective implique l'existence de preuve sans divulgation de la connaissance d'une solution de problème  $\mathcal{NP}$  (ZK=Zero-Knowledge proof).

Un protocole de preuve sans divulgation comprend deux acteurs :

- Un *prouveur* qui prouve sa connaissance d'une donnée sans la révéler.
- Un *vérifieur*, qui vérifie la preuve. à l'issue du protocole, le vérifieur est convaincu que le prouveur détient la donnée en question, mais n'a acquis aucune information sur cette dernière.

Pour montrer cette assertion, il suffit de la montrer pour un problème  $\mathcal{NP}$ -complet.

Illustrons cette propriété sur le problème 3-COL. Il s'agit de convaincre qu'un graphe  $\mathcal{G}$  est coloriable avec trois couleurs, sans révéler aucune information sur la façon de la colorier.

Le prouveur **A** peut prouver au vérifieur **B** qu'il connaît un coloriage, mais cette preuve n'est pas transmissible : **B** ne pourra pas convaincre d'autres personnes que le graphe est coloriable.

Une preuve sans divulgation se construit à partir d'un engagement de bits.

Un graphe  $\mathcal{G}$  est un couple  $(V, E)$  de sommets et d'arrêtes. Les arrêtes sont un sous-ensemble de paires  $\{i, j\}$  de sommets.

Un coloriage à trois couleurs est une application  $\sigma : V \rightarrow \{1, 2, 3\}$  telle que pour toute arrête  $\{i, j\}$ , on a  $\sigma(i) \neq \sigma(j)$ .

La preuve sans divulgation de la connaissance d'une telle application  $\sigma$  est interactive et se déroule sur plusieurs tours. Pour chaque tour :

1. ENGAGEMENT : le prouveur choisit une permutation aléatoire des couleurs  $\pi : \{1, 2, 3\} \rightarrow \{1, 2, 3\}$ , et pour chaque sommet, transmet un engagement de la composition du coloriage avec cette permutation  $\pi$ .
2. DÉFI : Le vérifieur défie le prouveur de révéler le coloriage d'une arrête  $\{i, j\}$  aléatoire.
3. RÉVÉLATION : Le prouveur révèle la valeur de désengagement, ce qui permet de vérifier que les sommets  $i$  et  $j$  sont coloriés différemment.

En raison de la propriété d'enchérissement, le prouveur ne peut plus changer les couleurs  $\sigma(i)$  et  $\sigma(j)$ .

Si le graphe n'est pas coloriable en trois couleurs, alors il existe au moins une arrête dont les sommets ont la même couleur, et le vérifieur pourra l'observer à l'issue du désengagement.

La probabilité de convaincre le vérifieur alors qu'on ne connaît pas le coloriage du graphe vaut  $1 - \frac{1}{|E|}$ . Après un nombre de tours égal à  $n \times |E|$ , la probabilité de fausse preuve vaut  $\left(1 - \frac{1}{|E|}\right)^{|E| \times n}$ .

Si le nombre d'arrête est assez grand, une bonne approximation de cette valeur est  $\left(\frac{1}{e}\right)^n$  qui converge vers 0. La probabilité de prouver la connaissance d'une donnée qu'on ignore est donc négligeable.

## § 2.4 Exercices

**2.8** Montrer que, si on suppose que la fonction RSA est à sens unique sans la connaissance de la factorisation du modulo, alors le bit de parité de la fonction RSA est difficile.

**2.9** Montrer que le bit de parité de la fonction d'exponentiation sur un corps fini n'est pas un bit difficile

**2.10** Montrer que si une fonction injective a un bit difficile, alors elle est à sens unique, et que ce résultat est faux si on ne suppose pas la fonction injective.

**2.11** Construire précisément un schéma de mise en gage de bit à partir d'une fonction à sens unique injective et prouver sa sécurité.



## Générateurs pseudo-aléatoires

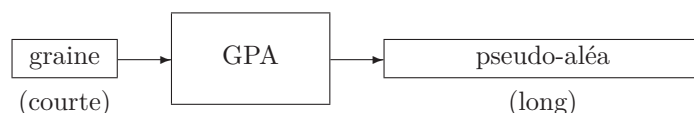
Les générateurs pseudo-aléatoires forment une première primitive utilisable pour chiffrer les messages. Dans un chiffrement à flot (*stream cipher*), le message en clair est constitué d'une suite de symboles binaires. Chaque symbole est combiné avec une suite appelée *suite chiffrante* par un ou exclusif pour former le cryptogramme. Le destinataire combine ce cryptogramme avec la même suite chiffrante et retrouve ainsi le message en clair. La suite chiffrante doit satisfaire les deux propriétés suivantes :

- elle doit être reproductible à l'identique par l'émetteur et le récepteur ;
- elle doit présenter toutes les caractéristiques d'une suite aléatoire.

Une suite binaire qui satisfait ces deux propriétés s'appelle une *suite pseudo-aléatoire*. Un algorithme qui produit de telles suites s'appelle un *générateur pseudo-aléatoire*.

Le principal résultat de ce chapitre est la construction d'un générateur pseudo-aléatoire à partir d'une fonction à sens unique.

Intuitivement, un générateur pseudo-aléatoire (GPA) agit comme un *amplificateur d'aléa*. Il s'agit d'un algorithme déterministe qui, à partir d'une petite source aléatoire, produit une suite plus longue. Ceci est illustré par le schéma suivant :



La propriété attendue d'un tel algorithme est qu'on ne peut pas distinguer les symboles produits d'une véritable source aléatoire.

### § 3.1 Indistinguabilité statistique

Tout d'abord, montrons qu'il est impossible de réaliser un générateur pseudo-aléatoire parfait. Il est impossible de créer du véritable aléa à partir d'une fonction déterministe.

#### Définition 3.1 [ distance statistique ]

Soient  $X$  et  $Y$  deux variables aléatoires à valeur dans un même ensemble fini  $\mathcal{X}$ . On appelle *distance statistique* de  $X$  et  $Y$  la quantité :

$$\Delta^s(X, Y) = \sum_{\alpha \in \mathcal{X}} \left| \Pr[X = \alpha] - \Pr[Y = \alpha] \right|.$$



La distance statistique de  $X$  à  $Y$  ne dépend que des lois des variables  $X$  et  $Y$ . Lorsque ces lois sont exprimées par un vecteur de probabilités, cela correspond à la distance  $L_1$  de ces vecteurs. La distance statistique a donc toutes les propriétés d'une distance :



- $\Delta^s(X, Y) = 0$  si et seulement si  $X$  et  $Y$  ont la même loi ;
- *Symétrie* :  $\Delta^s(X, Y) = \Delta^s(Y, X)$  ;
- *Inégalité triangulaire* :  $\Delta^s(X, Z) \leq \Delta^s(X, Y) + \Delta^s(Y, Z)$ .

Deux variables aléatoires seront d'autant plus proches l'une de l'autre selon cette distance qu'il sera difficile de distinguer une réalisation de l'une d'une réalisation de l'autre par un test statistique.

### Définition 3.2 [ Familles statistiquement indistinguables ]

Soient  $X = (X_n)_{n \in \mathbb{N}}$  et  $Y = (Y_n)_{n \in \mathbb{N}}$  deux familles de variables aléatoires à valeur dans le même ensemble fini  $\mathcal{X}$ . On dit que les deux familles sont statistiquement indistinguables si la distance statistique de  $X_n$  à  $Y_n$  est une fonction négligeable de  $n$ .

Un générateur pseudo-aléatoire parfait serait un algorithme qui produit des symboles statistiquement indistinguishable d'un véritable aléa, c'est-à-dire indistinguishable de la réalisation d'une variable aléatoire uniforme. Montrons qu'un tel générateur n'existe pas. Ce résultat est similaire au principe de non création d'information en théorie de l'information qui énonce qu'il est impossible de créer de l'information à partir d'une manipulation déterministe des données.

### Proposition 3.3 [ Inexistence d'un GPA statistique ]

Pour tout entier  $n$ , soit  $U_n$  la variable aléatoire uniforme sur l'ensemble  $\{0, 1\}^n$ . Soit  $g$  une fonction quelconque  $\{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$ . Alors, les familles de variables aléatoires  $(g(U_n))_{n \in \mathbb{N}}$  et  $(U_{n+1})_{n \in \mathbb{N}}$  ne sont pas statistiquement indistinguables.

*Preuve.* La loi de  $g(U_n)$  est donnée, par :

$$\forall y \in \{0, 1\}^{n+1} \quad \Pr[g(U_n) = y] = \sum_{x, g(x)=y} \Pr(U_n = x) = \frac{1}{2^n} \text{Card}\{x \in \{0, 1\}^n \mid g(x) = y\}.$$

Soit  $n$  un entier. La distance statistique  $d(n) = \Delta^s(g(U_n), U_{n+1})$  entre les variables aléatoires  $g(U_n)$  et  $U_{n+1}$  est donnée par :

$$\begin{aligned} d(n) &= \sum_{y \in \{0, 1\}^{n+1}} \left| \Pr[g(U_n) = y] - \Pr[U_{n+1} = y] \right| \\ &= \frac{1}{2^{n+1}} \sum_{y \in \{0, 1\}^{n+1}} \left| 2 \text{Card}\{x \in \{0, 1\}^n \mid g(x) = y\} - 1 \right| \end{aligned}$$

Dans cette somme, comme  $2 \text{Card}\{x \in \{0, 1\}^n \mid g(x) = y\}$  est un entier pair, chaque terme est supérieur ou égal à 1. D'où  $d(n) \geq 1$  qui n'est pas négligeable.  $\square$

## § 3.2 Indistinguishabilité calculatoire

La qualité d'un générateur pseudo-aléatoire se mesure à la difficulté de distinguer entre une source issue d'un véritable générateur d'aléa (GDA) et une source issue d'un générateur pseudo-aléatoire (GPA) est donc une notion calculatoire et asymptotique. On dira qu'un GPA est sûr si un algorithme de complexité polynomiale, effectuant un nombre polynomial de requêtes ne peut distinguer les réalisations du GPA d'un véritable aléa qu'avec une probabilité négligeable.

**Définition 3.4 [distingueur]**

Soient deux familles de variables aléatoires  $X = (X_n)_{n \in \mathbb{N}}$  et  $Y = (Y_n)_{n \in \mathbb{N}}$ . Un *distingueur* de  $X$  et  $Y$  est un algorithme dont les entrées sont toutes des réalisations de l'une des deux variables aléatoires  $X$  ou  $Y$  et qui renvoie 0 s'il détecte que toutes ses entrées sont des réalisations de  $X$  et qui renvoie 1 s'il détecte que ce sont des réalisations de  $Y$ .



Un distingueur prend en paramètre la réalisation d'une variable aléatoire. La valeur rendue est donc aléatoire.

La qualité d'un distingueur  $D$  se mesure à sa capacité à discerner l'une de l'autre variable aléatoire. Cette capacité est formalisée par un jeu. Les étapes du jeu sont :

1. Le maître du jeu choisit pour toute la suite l'une des deux familles de variables aléatoires  $X$  ou  $Y$  avec une probabilité  $1/2$  pour chacune.
2. L'algorithme  $D$  demande des réalisations de la variables aléatoire.
3. Au bout d'un certain temps, l'algorithme  $D$  fournit sa réponse : 0 s'il détecte que ce sont des réalisations de  $X$  qui lui ont été soumises, et 1 s'il détecte que ce sont des réalisations de  $Y$ .

Il y a une réalisation de jeu pour chaque entier  $n$ . On se limite à des algorithmes de complexité polynomiale en  $n$  et dont le nombre de requêtes à l'étape 2 est polynomiale.

Sa probabilité de succès est la probabilité qu'il réponde 0 sur des réalisations de  $X$  et qu'il réponde 1 sur des réalisations de  $Y$ , soit, pour tout entier  $n$  :

$$\Pr_n^{\text{succes}}(D) = \Pr[D = 0, X_n] + \Pr[D = 1, Y_n]$$



L'événement  $\{D = 0, X_n\}$  est réalisé lorsque l'algorithme  $D$  rend la réponse 0 et que ses entrées sont des réalisations de la variable aléatoire  $X_n$ . De même, l'événement  $\{D = 1, Y_n\}$  est réalisé lorsque l'algorithme  $D$  rend la réponse 1 et que ses entrées sont des réalisations de la variable aléatoire  $Y_n$ .



Il est entendu dans ces relations que l'algorithme  $D$  a une complexité polynomiale en  $n$ . Il réalise donc un nombre de requête borné par un polynôme en  $n$ .

Le choix entre  $X$  et  $Y$  étant équiprobable, on a :

$$\Pr_n^{\text{succes}}(D) = \frac{1}{2} \Pr[D = 0 \mid X_n] + \frac{1}{2} \Pr[D = 1 \mid Y_n]$$



La notation  $\Pr[D = 0 \mid X_n]$  désigne la probabilité conditionnelle que l'algorithme  $D$  rende la réponse 0 sachant que ses entrées sont des réalisations de la variable aléatoire  $Y_n$ .

**Définition 3.5 [avantage d'un distingueur]**

L'avantage d'un distingueur  $D$  de deux familles de variables aléatoires  $X = (X_n)_{n \in \mathbb{N}}$  et  $Y = (Y_n)_{n \in \mathbb{N}}$  est la fonction :

$$\text{Av}_n^{X,Y}(D) : n \mapsto |2 \Pr_n^{\text{succes}}(D) - 1|$$

L'avantage d'un distingueur est nul s'il ne peut pas reconnaître l'une et l'autre famille. Il vaut 1 s'il discerne toujours une réalisation de  $X_n$  d'une réalisation de  $Y_n$ . Une expression de l'avantage est :

$$\forall n \in \mathbb{N}, \text{Av}_n^{X,Y}(D) = |\Pr[D = 0 \mid X_n] - \Pr[D = 0 \mid Y_n]|$$

Pour deux familles de variables aléatoires  $X$  et  $Y$  données, l'ensemble des avantages de tous les distingueurs de  $X$  et  $Y$  est un sous-ensemble de  $\mathbb{R}$  majoré par 1. Il admet donc une borne supérieure, ce qui autorise la définition suivante :

**Définition 3.6 [ distance calculatoire de familles de variables aléatoires ]**

Étant données deux familles de variables aléatoires  $X = (X_n)_{n \in \mathbb{N}}$  et  $Y = (Y_n)_{n \in \mathbb{N}}$  sur un même ensemble, la distance calculatoire entre ces deux familles, notée  $\Delta_n^c(X, Y)$  est la fonction de  $n$  qui donne l'avantage du meilleur distingueur, c'est-à-dire pour tout entier  $n$  :

$$\Delta_n^c(X, Y) = \sup_{D \in \mathcal{D}} (\text{Av}_n^{X, Y}(D)),$$

où  $\mathcal{D}$  est l'ensemble des distingueurs de  $X$  et  $Y$  de complexité polynomiale.

La distance calculatoire satisfait l'inégalité triangulaire.

**Proposition 3.7 [ inégalité triangulaire sur la distance calculatoire ]**

Soient  $X = (X_n)_{n \in \mathbb{N}}$ ,  $Y = (Y_n)_{n \in \mathbb{N}}$  et  $Z = (Z_n)_{n \in \mathbb{N}}$  trois familles de variables aléatoires. La distance calculatoire satisfait pour tout entier  $n$  :

$$(4) \quad \Delta_n^c(X, Z) \leq \Delta_n^c(X, Y) + \Delta_n^c(Y, Z).$$

*Preuve.* Par définition de la borne supérieure, pour tout réel strictement positif  $\varepsilon$ , il existe un distingueur  $D_\varepsilon$  tel que :

$$\begin{aligned} \Delta_{X, Z}(n) &\leq \text{Av}_n^{X, Y}(D_\varepsilon) = \left| \Pr[D_\varepsilon = 0 \mid X_n] - \Pr[D_\varepsilon = 0 \mid Z_n] \right| \\ &\leq \left| \Pr[D_\varepsilon = 0 \mid X_n] - \Pr[D_\varepsilon = 0 \mid Y_n] \right| + \\ &\quad \left| \Pr[D_\varepsilon = 0 \mid Y_n] - \Pr[D_\varepsilon = 0 \mid Z_n] \right| \\ &\leq \text{Av}_n^{X, Y}(D_\varepsilon) + \text{Av}_n^{Y, Z}(D_\varepsilon) \end{aligned}$$

L'inégalité pour la distance calculatoire est obtenue par passage à la borne supérieure sur tous les distingueurs polynomiaux. Pour tout  $\varepsilon > 0$ , on a :

$$\Delta_n^c(X, Z) - \varepsilon \leq \Delta_n^c(X, Y) + \Delta_n^c(Y, Z).$$

Cette inégalité étant satisfaite pour tout  $\varepsilon > 0$ , l'inégalité (4) s'en déduit.  $\square$

L'objectif à atteindre pour un générateur pseudo-aléatoire est l'indistinguabilité calculatoire avec un aléa véritable.

**Définition 3.8 [ indistinguabilité calculatoire ]**

Deux familles de variables aléatoires  $X = (X_n)_{n \in \mathbb{N}}$  et  $Y = (Y_n)_{n \in \mathbb{N}}$  sont calculatoirement indistinguables si pour tout algorithme polynomial  $D$ , l'avantage de  $D$  pour distinguer  $X_n$  de  $Y_n$  est une fonction négligeable en  $n$ .

L'indistinguabilité calculatoire signifie qu'aucun algorithme polynomial n'est capable de discerner l'une de l'autre variable avec une probabilité non négligeable.

**Définition 3.9 [ générateur pseudo-aléatoire ]**

Un générateur pseudo-aléatoire est une fonction  $g : \{0,1\}^* \rightarrow \{0,1\}^*$ , calculable par un algorithme de complexité polynomiale et qui vérifie les deux conditions suivantes :

1. *Expansion* : pour tout élément  $s$  de  $\{0,1\}^*$ , la longueur  $g(s)$  est une fonction strictement croissante  $\ell$  de la longueur de  $x$ .
2. *Pseudo-aléa* : pour tout entier  $n$ , l'image par  $g$  de la distribution uniforme  $U_n$  est calculatoirement indistinguishable de la distribution uniforme  $U_{\ell(n)}$ .



La fonction  $n \mapsto \ell(n)$  est par définition la fonction d'expansion du générateur pseudo-aléatoire. La propriété d'expansion s'exprime par la relation :

$$\forall s \in \{0,1\}^* \quad |g(s)| = \ell(|s|).$$

Pour tout entier  $n$ , notons  $U = (U_n)_{n \in \mathbb{N}}$  la famille des variables aléatoires uniformes sur  $\{0,1\}^n$ . Posons  $X$  la famille de variables aléatoires  $X = (g(U_n))_{n \in \mathbb{N}}$  et posons  $Y$  la famille de variables aléatoires  $Y = (U_{\ell(n)})_{n \in \mathbb{N}}$ . La propriété de pseudo-aléa signifie que la distance calculatoire :

$$\Delta_n^c(X, Y)$$

est une fonction négligeable en  $n$ .

Pour résumer, un distingueur  $D$  du générateur pseudo-aléatoire  $g$  est un algorithme à qui on fournit, soit un véritable aléa, soit des réalisations du générateur pseudo-aléatoire et qui doit reconnaître le type des entrées qui lui sont soumises. Cela se définit formellement par le jeu suivant :

1. Le maître du jeu choisi entre le monde aléatoire  $a$  ou le monde pseudo-aléatoire  $pa$ .
2. À chaque requête du distingueur  $D$ , le maître du jeu fournit un véritable aléa  $U_{\ell(n)}$  ou un pseudo-aléa  $g(U_n)$  selon le monde choisi à l'étape 1.
3. Après un temps polynomial en  $n$ , le distingueur fournit une réponse :  $a$  ou  $pa$ .

Si la réponse du distingueur  $D$  à l'étape 3 est égale au monde choisi par le maître du jeu à l'étape 1, le distingueur a réussi. Dans le cas contraire, il a échoué. La qualité du distingueur se mesure à son avantage qui vaut :

$$\begin{aligned} \text{Av}_n(D) &= |2 \Pr_n^{\text{succes}}(D) - 1| \\ &= |\Pr[D = a \mid a] - \Pr[D = a \mid pa]| \end{aligned}$$

L'expression  $\Pr[D = a \mid a]$  désigne la probabilité que le distingueur  $D$  réponde  $a$  sachant que ses entrées sont vraiment aléatoires, et  $\Pr[D = a \mid pa]$  celle qu'il réponde  $a$  sachant que ses entrées sont pseudo-aléatoires, c'est-à-dire issues du calcul  $g(s)$  pour une graine  $s$  aléatoire.

### § 3.3 Construction de générateurs pseudo-aléatoires

L'existence de générateurs pseudo-aléatoires est équivalente à celle des fonctions à sens unique. L'exercice III.1 page 32 montre l'existence d'une fonction à sens unique à partir de celle de générateurs pseudo-aléatoires.

La suite de ce chapitre est consacrée à la construction de générateurs pseudo-aléatoires à partir d'une fonction à sens unique. La construction se fait en deux temps : tout d'abord la construction de générateur pseudo-aléatoires avec expansion d'un symbole binaire, puis avec une expansion d'un nombre polynomial de symboles binaires.

### 3.3.1. Générateur pseudo aléatoire avec expansion d'un symbole binaire

La première étape est de construire un pseudo-aléatoire dont la fonction d'expansion est :

$$\ell : n \mapsto n + 1.$$

#### Théorème 3.10

Si  $f$  est une fonction à sens unique qui conserve les longueurs et si  $b$  est un prédicat difficile de  $f$ , alors la fonction :

$$g : \begin{array}{ccc} \{0,1\}^* & \longrightarrow & \{0,1\}^* \\ x & \longmapsto & (f(x), b(x)) \end{array}$$

est un générateur pseudo-aléatoire.

*Preuve.* Supposons, pour une contradiction, que la fonction  $g$  ainsi définie n'est pas un générateur pseudo-aléatoire, c'est-à-dire qu'il existe un distingueur  $D$  sachant discerner avec une probabilité non négligeable, entre l'image par  $g$  d'une valeur aléatoire et une valeur vraiment aléatoire.

Utilisons ce distingueur  $D$  pour construire un algorithme  $A$  qui va trouver le prédicat difficile  $b(x)$  à partir d'une image  $f(x)$  avec un avantage non négligeable, ce qui montrera qu'alors, la fonction  $b$  n'est pas un prédicat difficile pour  $f$ , en contradiction avec l'hypothèse.

Sur une entrée  $y = f(x)$ , l'algorithme  $A$  choisit un symbole binaire aléatoire  $r \in \{0,1\}$ , puis soumet le couple  $(y, r)$  à l'algorithme  $D$  supposé discerner entre une valeur aléatoire et une valeur pseudo-aléatoire  $(f(x), b(x))$ , où  $x$  est aléatoire.

Si le distingueur  $D$  répond *pa*, « monde pseudo-aléatoire », c'est probablement que le symbole binaire  $r$  choisi aléatoirement vaut  $b(x)$ , et alors l'algorithme  $A$  renvoie  $r$  comme valeur du prédicat difficile.

Si au contraire, le distingueur  $D$  répond *a*, « monde aléatoire », c'est probablement que le symbole binaire  $r$  n'est pas la valeur de  $b(x)$ , et alors l'algorithme  $A$  renvoie son complémentaire  $\bar{r} = 1 - r$  comme valeur du prédicat difficile.

Montrons que cette stratégie fonctionne en calculant son avantage en fonction de celui de  $D$ . Par hypothèse, l'avantage de  $D$  est non négligeable, ce qui signifie que pour tout entier  $n$  :

$$\Pr_{z \in_R \{0,1\}^{n+1}} [D(z) = a] - \Pr_{x \in_R \{0,1\}^n} [D(f(x), b(x)) = a] = \mu(n)$$

où  $\mu$  est une fonction non négligeable de  $n$ . Maintenant, évaluons l'avantage de  $A$ , qui est égal à

$$\text{Av}_n(A) = 2 \Pr_{x \in_R \{0,1\}^n} [A(f(x)) = b(x)] - 1,$$

et montrons qu'il est non négligeable. Décomposons la probabilité de succès de l'algorithme  $A$  selon la valeur de  $b(x)$ . Si  $b(x) = r$ , la réponse de  $A$  est correcte si  $D$  répond *pa* et si  $b(x) = 1 - r$ , la réponse de  $A$  est correcte si  $D$  répond *a*. Par conséquent, en application de la formule de BAYES :

$$\begin{aligned} \text{Av}_n(A) &= 2 \Pr_{x \in_R \{0,1\}^n} [b(x) = r] \times \Pr_{x \in_R \{0,1\}^n} [D(f(x), r) = pa] \\ &\quad + 2 \Pr_{x \in_R \{0,1\}^n} [b(x) = 1 - r] \times \Pr_{x \in_R \{0,1\}^n} [D(f(x), r) = a] \\ &\quad - 1 \end{aligned}$$

Le bit  $r$  choisi par  $A$  étant aléatoire,  $b(x)$  vaut  $r$  ou  $1 - r$  avec une probabilité  $1/2$ . D'autre part, si  $b(x) = 1 - r$ , l'entrée  $(f(x), r)$  de l'algorithme  $D$  est finalement une entrée aléatoire. De plus, la probabilité que  $D$  réponde *pa* est le complément à 1 de celle qu'il réponde *a*, donc finalement :

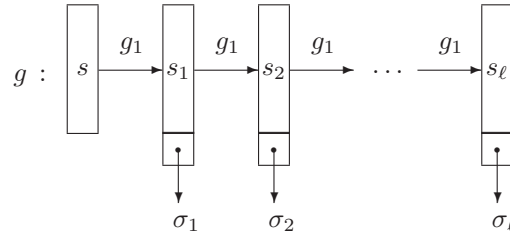
$$\begin{aligned} \text{Av}_n(A) &= \left(1 - \Pr_{x \in_R \{0,1\}^n} [D(f(x), r) = a]\right) + \Pr_{x \in_R \{0,1\}^{n+1}} [D(x) = a] - 1 \\ &= \Pr_{z \in_R \{0,1\}^{n+1}} [D(z) = a] - \Pr_{x \in_R \{0,1\}^n} [D(f(x), b(x)) = a] \\ &= \mu(n) \end{aligned}$$

L'avantage de  $A$  est finalement le même que celui de  $D$ . Il est non négligeable.  $\square$

### 3.3.2. Générateur pseudo-aléatoire avec expansion polynomiale

La deuxième étape est de construire un générateur avec une expansion supérieure à un seul symbole binaire, de manière à produire une longue suite pseudo-aléatoire

On suppose qu'on dispose d'un générateur  $g_1$  d'expansion un symbole binaire, comme par exemple celui construit au paragraphe précédent et qui repose sur une fonction à sens unique qui conserve les longueur et sur un prédicat difficile de cette fonction. Pour produire plusieurs symboles binaires pseudo-aléatoire  $\sigma_1\sigma_2\cdots\sigma_\ell$ , on peut itérer  $g_1$  comme indiqué sur la figure suivante :



#### Théorème 3.11 [ générateur pseudo-aléatoire avec expansion polynomiale ]

Étant donné un générateur pseudo-aléatoire  $g_1$  d'expansion 1 symbole binaire, soit  $p(n)$  un polynôme en  $n$  vérifiant pour tout  $n$  entier  $p(n) > n$ , soit  $n$  un entier et  $\ell = p(n)$ , soit  $g$  la fonction définie sur  $\{0,1\}^*$  dont la valeur sur les vecteurs binaires  $s$  de dimension  $n$  est :

$$g(s) = \sigma_1\sigma_2\cdots\sigma_\ell,$$

définie par  $s_0 = s$ , et, pour  $i = 1$  jusqu'à  $\ell$ , par  $g_1(s_{i-1}) = (s_i, \sigma_i)$ . Alors  $g$  est un générateur pseudo-aléatoire.

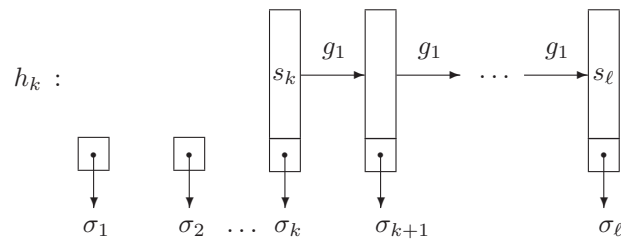
*Preuve.* Soit  $Y$  la variable aléatoire produite par le générateur pseudo-aléatoire  $g$  avec une graine  $S$  aléatoire. Il faut montrer que la distance calculatoire  $\Delta(Y, U)$  est négligeable. Pour montrer cela, on utilise un procédé de démonstration appelé parfois *raisonnement hybride*. Cela consiste à définir une suite de variables aléatoires  $H_0 = Y, H_1, \dots, H_\ell = U$ , et à montrer que la distance calculatoire entre un terme de cette suite et le terme suivant est négligeable.

Par l'inégalité triangulaire(4), on déduira que :

$$\Delta_n^c(Y, U) \leq \sum_{k=0}^{\ell-1} \Delta_n^c(H_k, H_{k+1}),$$

sera négligeable comme somme d'un nombre polynomial de termes négligeables.

Pour  $k = 1$  jusqu'à  $\ell$ , on considère le procédé  $h_k$  qui consiste à produire de l'aléa véritable jusqu'au rang  $k$ , puis d'appliquer le même procédé que  $g$  pour produire la séquence pseudo-aléatoire à partir du rang  $k + 1$ , en utilisant  $s_k$  comme germe initial.



Ainsi, les symboles  $\sigma_1 \cdots \sigma_k$  sont aléatoires, alors que les symboles  $\sigma_{k+1} \cdots \sigma_\ell$  sont pseudo-aléatoires.

Soit  $H_k$  la variable aléatoire égale à la production du générateur pseudo-aléatoire  $h_k$ . Montrons que la distance calculatoire entre  $H_k$  et  $H_{h+1}$  est négligeable, ce qui démontrera le théorème. Pour cela, il suffit de montrer que s'il existe un algorithme capable de discerner la production de  $h_k$  de la production de  $h_{k+1}$ , alors on peut en déduire un algorithme contre le générateur pseudo-aléatoire  $g_1$ . Ceci montrera que si  $g_1$  est un générateur pseudo-aléatoire, alors  $h_k$  et  $h_{k+1}$  sont calculatoirement indistinguables.

Supposons donc l'existence d'un distinguéur  $D_k$ , capable de discerner une réalisation de  $h_k$  d'une réalisation de  $h_{k+1}$  avec un avantage non-négligeable. Ce distinguéur peut être utilisé pour réaliser un algorithme  $A$  qui discerne une réalisation de  $g_1$  (GPA) d'une réalisation d'une variable entièrement aléatoire (GDA) comme suit :

Sur une entrée  $y \in \{0,1\}^{n+1}$ , l'algorithme construit la séquence  $\sigma_1 \dots \sigma_\ell$ , en tirant  $\sigma_1 \cdots \sigma_k$  au hasard, puis en posant  $y = (s_{k+1}, \sigma_{k+1})$ , puis, pour  $i = k+2$  jusqu'à  $\ell$ , en calculant  $(s_i, \sigma_i) = g_1(s_{i-1})$ .

Si le distinguéur  $D_k$  répond  $h_k$ , intuitivement, c'est que la valeur de  $y$  est elle-même issue d'un calcul  $y = g_1(s)$ . L'algorithme  $A$  renvoie  $pa$ .

Si au contraire, le distinguéur  $D_k$  répond  $h_{k+1}$ , c'est que la valeur de  $y$  est aléatoire, et donc l'algorithme  $A$  renvoie  $a$ .

Par construction, l'algorithme  $A$  répond  $pa$  sur une entrée  $y$  avec la même probabilité que  $D_k$  répond  $h_k$  sur une entrée  $\sigma = \sigma_1, \dots, \sigma_\ell$ .

$$\Pr[A(y) = pa] = \Pr[D_k(\sigma) = h_k]$$

Il en résulte que  $A$  et  $D_k$  ont le même avantage. Si celui de  $D_k$  n'est pas négligeable en  $\ell$ , alors celui de  $A$  n'est pas négligeable non plus en  $n$ .  $\square$

## § 3.4 Exercices

**3.12** Démontrer que s'il existe un générateur pseudo-aléatoire avec une expansion  $\ell(n) = 2n$ , alors il existe une fonction à sens unique.

**3.13** Une application de la technique hybride Soit  $g : \{0,1\}^n \rightarrow \{0,1\}^{2n}$  un générateur pseudo-aléatoire d'expansion  $\ell(n) = 2n$ . Démontrer que pour tout polynôme  $t(n)$  en  $n$ , la fonction

$$g_{t(n)} : \begin{array}{ccc} \{0,1\}^{n \times t(n)} & \longrightarrow & \{0,1\}^{2n \times t(n)} \\ x_1 \cdots x_{t(n)} & \longmapsto & g(x_1) \cdots g(x_{t(n)}) \end{array}$$

définit un générateur pseudo-aléatoire. Plus précisément, montrer que si  $G$  est la variable aléatoire  $G = g(X)$  avec  $X$  aléatoire dans  $\{0,1\}^n$ , et  $G^{t(n)} = g_{t(n)}(X_1, \dots, X_{t(n)})$ , avec les  $X_i$  aléatoires dans  $\{0,1\}^n$

$$\Delta(G^{t(n)}, U_{2nt(n)}) \leq t(n) \Delta(G, U_{2n})$$

**3.14** Construire un générateur pseudo-aléatoire avec la fonction RSA.

**3.15** Démontrer la sécurité du générateur de BLUM-BLUM-SHUB défini comme suit :

$$\begin{aligned} N &= p \times q \\ s_{k+1} &= s_k^2 \bmod N \\ \sigma_k &= s_k \bmod 2 \end{aligned}$$

**3.16** (*multiple message RSA avec le même exposant public*) Montrer qu'on peut facilement retrouver un message  $m$  en observant les  $e$  cryptogrammes de  $m$  chiffrés avec le même exposant public  $e$ .

**III.6.** (*chiffrement de PAILLIER*) Soit  $N = p \times q$  le produit de deux nombres premiers distincts. La fonction de chiffrement est :

$$E; \begin{array}{ccc} \mathbb{Z}/(N) \times \mathbb{Z}/(N)^* & \longrightarrow & \mathbb{Z}/(N^2)^* \\ (m, r) & \longmapsto & (1 + N)^m r^N \end{array}$$

1. Démontrer que la fonction  $E$  est une bijection.
2. Étant donné la clé privée  $p \times q$ , décrire la fonction de déchiffrement pour retrouver la valeur de  $m$ .
3. Supposons que l'ensemble  $\{r^N \bmod N^2 \mid r \in \mathbb{Z}/(N)^*\}$  est calculatoirement indiscernable de l'ensemble  $\mathbb{Z}/(N^2)$ . Démontrer que pour tout message  $m$ , l'ensemble des cryptogrammes  $\{E(m, r) \mid r \in \mathbb{Z}/(N)^*\}$  est calculatoirement indiscernable de  $\mathbb{Z}/(N^2)$ . Cela montrera qu'aucune information sur  $m$  ne transparaît dans le cryptogramme.



La fonction de chiffrement de PAILLIER a la propriété intéressante d'homomorphisme suivante :

$$E(m_1, r_1) \times E(m_2, r_2) = E(m_1 + m_2, r_1 r_2)$$

Montrer comment cela peut avantageusement s'appliquer au vote électronique, en association avec une fonction de partage de secret.





# Fonctions pseudo-aléatoires

## § 4.1 Motivation et définition

Les générateurs pseudo-aléatoires présentés au paragraphe précédent permettent déjà de réaliser un chiffrement de messages plus longs que la clé. Dans le chiffrement de VERNAM, chaque symbole binaire du message est additionné modulo 2 au symbole correspondant d'une chaîne aléatoire partagée par les correspondants, comme illustré ci-après :

$$\begin{array}{r} \boxed{s} \\ \oplus \\ \boxed{m} \\ \hline = \boxed{c} \end{array}$$

Pour assurer la sécurité de ce type de chiffrement, il est primordial de n'utiliser la séquence aléatoire qu'une seule fois (*one time pad*). Pour chiffrer un autre message, il faut utiliser une autre graine.

Un générateur pseudo-aléatoire peut permettre d'utiliser la même graine pour chiffrer plusieurs messages.

Une manière de procéder est de produire une très longue chaîne pseudo-aléatoire, et de communiquer un indice à partir duquel considérer la séquence pseudo-aléatoire pour le chiffrement.

$$\begin{array}{c} \boxed{s} \xrightarrow{\text{GPA}} \boxed{\phantom{\sigma_i}} \quad \boxed{\sigma_i} \quad \boxed{\phantom{m}} \\ \oplus \\ \boxed{m} \\ \hline = \boxed{c} \end{array}$$

Comme la production des symboles pseudo-aléatoires est itérative, l'inconvénient est alors d'avoir à recommencer tous les calculs depuis le premier à chaque message.

L'idéal serait d'avoir un accès direct aux termes suivants de la séquence pseudo-aléatoire, sans avoir à calculer les précédents.

La notion famille de fonctions pseudo-aléatoires réponds à cet objectif. Il s'agit de construire une famille de fonctions  $(F_k)_{k \in \mathbb{N}}$ , calculable en temps acceptable, et qu'on ne doit pas pouvoir distinguer d'une fonction aléatoire.

Notons  $\mathcal{H}_n$  l'ensemble de toutes les fonctions  $\{0, 1\}^{\ell(n)} \longrightarrow \{0, 1\}^{\leq \ell(n)}$ . Une famille pseudo-aléatoire est une famille de fonction qu'on ne sait pas distinguer de  $\mathcal{H}_n$ .

**Définition 4.1 [ famille de fonctions pseudo-aléatoires ]**

Une famille de fonctions  $\mathcal{F}_n = \{F_k : \{0,1\}^{\ell(n)} \rightarrow \{0,1\}^{\leq \ell(n)} \mid k \in \{0,1\}^n\}$  est dite pseudo-aléatoire si elle satisfait les deux propriétés suivantes :

1. Elle est efficacement calculable, c'est-à-dire il existe un algorithme en temps polynomial qui sur l'entrée  $k \in \{0,1\}^n$  et  $x \in \{0,1\}^{\ell(n)}$ , retourne la valeur  $F_k(x)$ .
2. Elle est pseudo-aléatoire, c'est-à-dire l'avantage de tout algorithme  $\mathcal{A}$  en temps polynomial, qui distingue un élément de la famille  $\mathcal{F}_n$  d'une fonction aléatoire de  $\mathcal{H}_n$  est négligeable :

$$\left| \Pr_{F \in \mathcal{F}_n} [\mathcal{A}^F(1^n) = 1] - \Pr_{H \in \mathcal{H}_n} [\mathcal{A}^H(1^n) = 1] \right|$$

est une fonction négligeable de  $n$ .



Le nombre des paramètres  $k \in \{0,1\}^n$  des fonctions  $F_k$  vaut  $2^n$  qui est une fonction exponentielle en  $n$ . Un générateur pseudo-aléatoire ne définit donc pas directement une famille de fonctions pseudo-aléatoires, car on ne peut pas calculer efficacement toutes les valeurs.

Un adversaire contre une famille de fonctions aléatoires est un algorithme qui fait des requêtes à un oracle. Il existe deux sortes d'oracles : les oracles qui, sur une entrée  $x$ , rendent la valeur  $F_K(x)$  d'un élément fixé, mais inconnu de la famille  $\mathcal{F}_n$ , et les oracles qui, sur une entrée  $x$ , rendent la valeur  $H(x)$  d'une fonction aléatoire fixée. L'algorithme doit pouvoir efficacement répondre si les réponses de l'oracle sont celles d'une fonction aléatoire ou d'une fonction pseudo-aléatoire.

**§ 4.2 Construction d'une famille de fonctions pseudo-aléatoires**

Ce paragraphe présente la construction d'une famille de fonctions pseudo-aléatoires (FPA) à partir d'un générateur pseudo-aléatoire (GPA). Elle est due à GOLDREICH, GOLDWASSER et MICALI.

On suppose que l'on dispose d'un générateur pseudo-aléatoire d'expansion  $\ell(n) = 2n$  :

$$g : \begin{array}{ccc} \{0,1\}^n & \longrightarrow & \{0,1\}^{2n} \\ k & \longrightarrow & g(k) = (g_0(k), g_1(k)) \end{array}$$

On note  $g_0$  et  $g_1$  les fonctions dont les valeurs sont respectivement les  $n$  premiers et les  $n$  derniers symboles binaires de la valeur du générateur pseudo-aléatoire  $g$ .



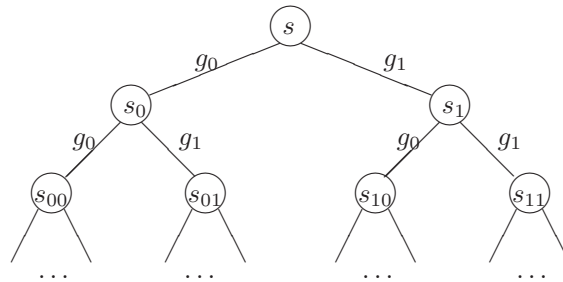
Soit  $s$  la graine du générateur pseudo-aléatoire  $g$ . On étend récursivement, pour tout entier  $\ell$  et tout mot  $x \in \{0,1\}^\ell$ , la fonction  $g^x(s)$  en posant  $x = x'e$  avec  $x' \in \{0,1\}^{\ell-1}$  et  $e \in \{0,1\}$  et :

$$g^x(s) = g^{x'} \circ g_e(s) = s_x$$

La famille  $\mathcal{F}_n$  est alors définie, pour  $x \in \{0,1\}^{\ell(n)}$ , par :

$$f_s(x) = g^x(s).$$

Cette construction peut se représenter par un arbre :



Chaque feuille contient la valeur de  $f_s(x) = s_x$ .

Lorsque le vecteur  $x$  s'écrit  $x = x_1 x_2 \cdots x_n$ , la valeur  $f_s(x)$  vaut  $f_s(x) = g_{x_n} \circ \cdots \circ g_{x_2} \circ g_{x_1}(s)$ .

### Théorème 4.2

La famille  $\mathcal{F}_n = (f_k)_{k \in \{0,1\}^n}$  construite comme ci-dessus avec  $f_k : \{0,1\}^n \mapsto \{0,1\}^n$  est une famille de fonctions pseudo-aléatoires.

*Preuve.* La preuve de ce théorème utilise la technique hybride. Construisons une suite de familles de fonctions  $H^0, H^1, H^2, \dots, H^n$  telle que  $H^0 = \mathcal{F}_n$  et  $H^n = \mathcal{H}_n$ .

Le choix aléatoire d'un élément dans  $H^0$  est dans le choix aléatoire de la valeur  $s$  situé à la racine de l'arbre ci-dessus.

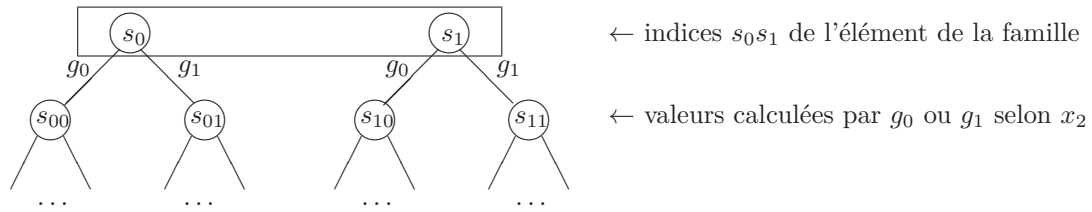
Le choix aléatoire d'un élément dans la famille  $H^n$  est dans le choix aléatoire de chacune des feuilles de l'arbre.

Les familles  $H^i$ , pour  $i = 1$  jusqu'à  $n - 1$  seront définies en choisissant aléatoirement les valeurs situées à la profondeur  $i$  dans l'arbre, puis en calculant les fils avec les deux fonctions  $g_0$  et  $g_1$  du générateur pseudo-aléatoire.

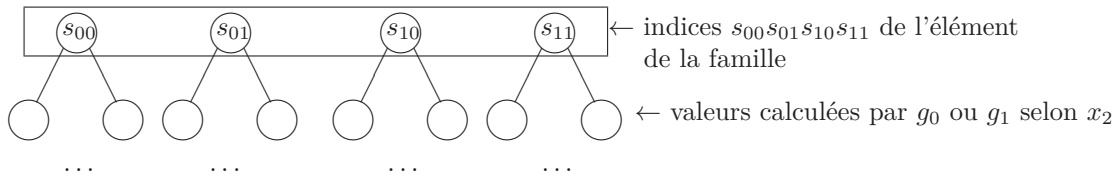
Toutes ces familles sont des familles de fonctions de  $n$  variables binaires.

La famille  $H^0$  est l'ensemble des  $2^n$  fonctions  $\{f_s\}_{s \in \{0,1\}^n}$ , dont la valeur en  $x = x_1, \dots, x_n$  est donnée par  $f_s(x) = g_{x_n} \circ \cdots \circ g_{x_2} \circ g_{x_1}(s)$ . Cette famille est illustrée par l'arbre ci-dessus.

La famille  $H^1$  est l'ensemble des  $2^{2n}$  fonctions  $\{f_{s_0 s_1}\}_{s_0 s_1 \in \{0,1\}^{2n}}$  dont la valeur en  $x = x_1, \dots, x_n$  est donnée par  $f_{s_0 s_1}(x) = g_{x_n} \circ \cdots \circ g_{x_2}(s_{x_0})$ . Une illustration de cette famille est donnée par :



La famille  $H^2$  est l'ensemble des  $2^{2^2 n}$  fonctions  $\{f_{s_{00} s_{01} s_{10} s_{11}}\}_{s_{00} s_{01} s_{10} s_{11} \in \{0,1\}^{2^2 n}}$  dont la valeur en  $x = x_1, \dots, x_n$  est donnée par  $f_{s_{00} s_{01} s_{10} s_{11}}(x) = g_{x_n} \circ \cdots \circ g_{x_3}(s_{x_0 x_1})$ . L'arbre suivant illustre cette famille :



Ainsi de suite...

Pour un entier  $k$  compris entre 1 et  $n - 1$ , la famille  $H^k$  est l'ensemble des  $2^{2^k n}$  fonctions  $\{f_{s_{0 \dots 0} \dots s_{1 \dots 1}}\}_{s_{0 \dots 0} \dots s_{1 \dots 1} \in \{0,1\}^{2^k n}}$  dont la valeur en  $x = x_1, \dots, x_n$  est donnée par  $f_{s_{0 \dots 0} \dots s_{1 \dots 1}}(x) = g_{x_n} \circ \cdots \circ g_{x_{k+1}}(s_{x_0 \dots x_k})$ .

Pour achever la preuve, il reste à montrer que la distance calculatoire entre deux familles consécutives reste négligeable. C'est l'objet du lemme qui suit.

**Lemme 4.3**

On reprend les notations de l'exercice 1 de la section précédente. Soit  $t(n)$  un polynôme en  $n$ . Pour tout distinguéur  $D$  de  $H^k$  et  $H^{k+1}$  qui effectue moins de  $t(n)$  requêtes un oracle répond selon un élément d'une de ces deux familles, alors

$$\text{Av}^D(H^k, H^{k+1}) \leq \Delta(G^{t(n)}, U_{2nt(n)})$$

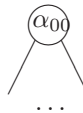
*Preuve.*

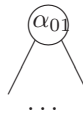
Par contraposée, supposons qu'il existe un distinguéur  $D$ , capable de discerner entre  $H^k$  et  $H^{k+1}$  avec  $t(n)$  requêtes avec un avantage non négligeable, et montrons qu'on peut construire un distinguéur  $B$  capable de discerner entre la variable aléatoire  $G^{t(n)}$ , développement  $t(n)$  fois du générateur pseudo-aléatoire  $G$ , et la variable aléatoire uniforme  $U_{2nt(n)}$  sur  $2nt(n)$  symboles binaires.

L'entrée de l'algorithme  $B$  est un vecteur  $\alpha = \alpha_1, \dots, \alpha_{t(n)}$ , où chaque  $\alpha_i = (\alpha_i^0, \alpha_i^1)$  est constitué de deux composantes dans  $\{0, 1\}^n$ . L'algorithme  $B$  utilise le distinguéur  $D$  pour répondre si l'entrée  $\alpha$  est le résultat du générateur pseudo-aléatoire ou est du véritable aléa.

L'algorithme  $B$  va simuler un oracle auprès du distinguéur  $D$ . Afin de discerner entre  $H^k$  et  $H^{k+1}$ , le distinguéur  $D$  va déposer des requêtes qui consistent à calculer la valeur de l'oracle pour une valeur  $x = x_1, \dots, x_n$ . Lorsque  $D$  dépose la  $i^{\text{e}}$  requête  $x_1 \dots x_n$ , l'algorithme  $B$  vérifie d'abord si le préfixe  $x_1 \dots x_{k+1}$  n'est pas le préfixe d'une requête déjà déposée. Si le préfixe est nouveau, l'algorithme  $B$  calcule la valeur  $f_s(x)$  de l'oracle en simulant un élément de la famille  $H^{k+1}$  en utilisant le vecteur  $\alpha_i$  comme paramètre :

$$f_{s_0 \dots s_1 \dots s_{t(n)-1}}(x) = g_{x_n} \circ \dots \circ \begin{cases} g_{x_{k+2}}(\alpha_{i0}) & \text{si } x_{k+1} = 0 \\ g_{x_{k+2}}(\alpha_{i1}) & \text{si } x_{k+1} = 1 \end{cases}$$





← valeurs initialisées selon  $x_k$

← valeurs calculées par  $g_0$  ou  $g_1$  selon  $x_{k+1}$

Si le préfixe de la requête est le même qu'une requête déjà déposée, alors l'algorithme  $B$  renvoie à  $D$  la valeur antérieurement calculée pour ce préfixe.

Ainsi, si  $\alpha_i$  est le résultat du générateur pseudo-aléatoire  $g$ , alors cet oracle simule exactement  $H^k$  et si  $\alpha_i$  est un aléa véritable, alors cet oracle simule exactement  $H^{k+1}$ .

Par conséquent, si le distinguéur  $D$  sait discerner entre  $G^{t(n)}$  et  $U_{nt(n)}$ , alors, en rendant le même résultat que  $D$ , l'algorithme  $B$  sait discerner entre  $H^k$  et  $H^{k+1}$  avec le même avantage :

$$\text{Av}^D(H^k, H^{k+1}) = \text{Av}^B(G^{t(n)}, U_{2nt(n)})$$

Le résultat du lemme s'obtient par passage à la borne supérieure sur tous les distinguéurs de  $G^{t(n)}$  et  $U_{2nt(n)}$ .  $\square$

*Fin de la preuve du théorème 4.2.* D'après l'exercice 1 de la section précédente, on a

$$\Delta(G^{t(n)}, U_{2nt(n)}) \leq t(n)\Delta(G, U_{2n})$$

Par composition, l'avantage d'un distinguéur de  $\mathcal{F}_n$  et  $\mathcal{H}_n$  est majoré par  $nt(n)\Delta(G, U_{2n})$  qui est négligeable dès que  $\Delta(G, U_{2n})$  est négligeable.

$\square$

### § 4.3 Exercices

1. On reprend la construction de GOLDREICH, GOLDWASSER et MICALI, mais au lieu de définir des fonctions  $\{0, 1\}^n$ , on les définit sur  $\{0, 1\}^{\leq n}$ . C'est-à-dire que  $f(x)$  est défini pour des tailles de  $x$  variables de 1 jusqu'à  $n$  symboles binaires. Cela définit-il encore une famille de fonctions pseudo-aléatoire ?

2. *Construction d'une famille pseudo-aléatoire de fonctions vectorielles à partir d'une famille de fonctions booléennes pseudo-aléatoires.* Étant donnée une famille pseudo-aléatoire  $(f_k)_{k \in \{0, 1\}^n}$  de fonctions booléennes  $f_k : \{0, 1\}^{p(n)} \rightarrow \{0, 1\}$ , construire une famille de fonctions pseudo-aléatoire  $(g_s)_{s \in S}$  de fonctions vectorielles  $g_s : \{0, 1\}^{p(n)} \rightarrow \{0, 1\}^\ell$ .

3. *Une famille de fonctions pseudo-aléatoires est toujours à sens unique*

On dit qu'une famille de fonctions pseudo-aléatoires  $\{0, 1\} \rightarrow \{0, 1\}$  est une *famille pseudo-aléatoire de fonctions à sens unique* si, étant donnée un élément  $y$  de l'image  $\{0, 1\}^n$ , et étant donné un oracle qui réalise un élément aléatoire  $f_k$  de la famille, tout algorithme efficace ne peut trouver un antécédent de  $y$  de  $f_k$  qu'avec une probabilité négligeable.



Cela ne signifie pas que les éléments  $f_k$  de la famille sont des fonctions à sens unique. Connaissant la clé  $k$ , il n'y a aucune raison que ces fonctions soient difficiles à inverser. La définition stipule seulement qu'il n'est pas possible de trouver un antécédent d'un élément  $y$  sans connaissance de la clé  $k$  et lorsqu'on ne dispose que d'un oracle pour déterminer les valeurs  $f_k(x)$ .

Montrer que si la famille  $\mathcal{F}_n$  est une famille de fonctions pseudo-aléatoire, alors elle est une famille pseudo-aléatoire de fonctions à sens unique.



# Chiffrement

L'objet du chiffrement est de réserver le contenu d'un message à son destinataire légitime et de le rendre inaccessible à tout autre. Dans cette section, la sécurité de cette primitive est formalisée.

## § 5.1 Schéma de chiffrement

Définissons tout d'abord ce qu'est un schéma de chiffrement.

### Définition 5.1 [ schéma de chiffrement ]

Un schéma de chiffrement est une famille de quintuplets  $(\mathcal{D}, \mathcal{R}, G, D, E)_{n \in \mathbb{N}}$ , indexés par la valeur du paramètre de sécurité. Pour chaque valeur  $n$  de ce paramètre,

- $\mathcal{D}$  désigne l'ensemble des messages en clair sur lequel opère le chiffrement,
- $\mathcal{R}$  désigne l'ensemble auquel appartiennent les cryptogrammes,
- $G$  est l'algorithme de génération de clé, de complexité polynomiale en  $n$ , qui produit une paire de clés  $(e, d)$  : la clé  $e$  pour le chiffrement, et la clé  $d$  pour le déchiffrement.
- $E$  est l'algorithme de chiffrement, de complexité polynomiale en  $n$ . Pour chaque message  $m$  dans  $\mathcal{D}$ , il invoque la clé de chiffrement  $e$  pour produire un cryptogramme  $c$  dans  $\mathcal{R}$  :

$$c = E(e, m).$$

- $D$  est l'algorithme de déchiffrement, de complexité polynomiale en  $n$ . Pour chaque cryptogramme qui est le résultat d'un chiffrement, il reconstitue le message clair correspondant, grâce à la clé de déchiffrement. Pour tout message  $m \in \mathcal{D}$ , on a :

$$D(d, E(e, m)) = m.$$



L'algorithme de chiffrement  $E$  n'est pas nécessairement déterministe. Lorsqu'on lui soumet un même message  $m$  à deux instants différents, il peut produire des cryptogrammes distincts. Cela est modélisé par un algorithme déterministe à qui l'on passe une chaîne aléatoire comme paramètre supplémentaire.

Par contre, l'algorithme de déchiffrement, lui, est toujours déterministe et reproduit toujours le message en clair d'où est issu le cryptogramme.



Le paramètre de sécurité  $n$  est omis par soucis de simplification des notations, mais cela ne doit pas occulter que les algorithmes  $G$ ,  $E$  et  $D$  sont de complexité polynomiale en  $n$ .



Si  $e = d$ , c'est-à-dire si la clé de chiffrement est la même que la clé de déchiffrement, le schéma de chiffrement est dit *symétrique*. Si la clé de déchiffrement est différente de la clé de chiffrement, mais s'en déduit par un algorithme de complexité polynomiale, il est toujours possible de l'intégrer dans l'algorithme de déchiffrement. On peut donc dans ce cas supposer que  $e = d$ .



Dans le cas contraire, c'est-à-dire si la clé de déchiffrement se calcule pas efficacement à partir de la clé de chiffrement, il n'y a aucune raison de cacher  $e$ . Elle peut être rendue publique. On parle alors de schéma de chiffrement *asymétrique* ou à *clé publique*.



Si l'algorithme de chiffrement n'est pas déterministe, alors un même message peut avoir plusieurs cryptogrammes. Mais pour chacun de ces cryptogrammes, l'algorithme de déchiffrement retourne toujours le même message d'origine.

C'est par exemple le cas du système de chiffrement ELGAMAL.

## § 5.2 Notions de sécurité

La notion de fonction à sens unique est insuffisante pour une fonction de chiffrement satisfaisante. On a vu que si  $f$  est une fonction à sens unique sur  $\{0, 1\}^n$ , alors la fonction :

$$g : \begin{array}{ccc} \{0, 1\}^n & \longrightarrow & \{0, 1\}^n \\ (m_1, m_2) & \longmapsto & (f(m_1), m_2) \end{array}$$

est toujours une fonction à sens unique, puisque pour inverser  $g$ , il est nécessaire de pouvoir inverser  $f$ . Par contre il s'agit d'une piègeuse fonction de chiffrement, car la moitié des symboles binaires du clair ne sont pas chiffrés.

Ce qu'on attend d'une fonction de chiffrement est qu'elle protège *tous* les bits du message clair. On peut imaginer par exemple qu'un long message militaire contienne une information cruciale, comme la décision ou non d'attaquer l'ennemi.

Il existe deux notions de sécurité : la sécurité sémantique et l'indistinguabilité. La première décrit bien ce que les acteurs d'un système cryptographique attendent comme service de confidentialité. La seconde est commode pour établir des preuves de sécurité. Nous montrons dans ce paragraphe que ces deux notions sont finalement équivalentes.

Ces deux notions sont formalisées sous la forme d'un jeu.

### 5.2.1. Sécurité sémantique

La sécurité sémantique est la version calculatoire de la sécurité parfaite. Elle énonce que le cryptogramme n'apporte aucune information que l'adversaire ne puisse calculer sans lui. Tout ce qui est calculable sur le clair peut se calculer sans le cryptogramme. Si l'adversaire sait calculer la moindre information sur message en clair à partir du cryptogramme alors la sécurité sémantique n'est pas atteinte.

La sécurité sémantique est formalisée par un jeu entre un adversaire et le maître du jeu. L'adversaire est mis au défi de trouver une information sur le message clair dont le maître du jeu lui fournit un cryptogramme.

Soit  $p$  une distribution de probabilité sur l'ensemble  $\mathcal{D}$  des messages en clair. Soit  $f$  une fonction d'information sur l'ensemble des messages et à valeur dans un ensemble quelconque  $E$  :

$$f : \begin{array}{ccc} \mathcal{D} & \rightarrow & E \\ m & \mapsto & f(m) \end{array}$$

Le jeu de la sécurité sémantique est défini ci-après :

JEU SEM

1. Le maître du jeu  $M$  choisit au hasard un message  $m$  dans  $\mathcal{D}$  en suivant la loi de probabilité  $p$  ainsi qu'une clé de chiffrement  $e$ . Soit  $y = f(m)$  l'information sur le message  $m$ .
  2.  $M$  chiffre le message  $m$  avec la clé  $e$  et transmet le cryptogramme  $c$  à l'adversaire  $A$ .
  3. Après un temps polynomial, l'adversaire  $A$  renvoie la valeur  $y^*$  estimée pour  $y$ .
- L'adversaire  $A$  a gagné si  $y = y^*$ .

**Définition 5.2 [ Avantage d'un adversaire contre la sécurité sémantique ]**

*L'avantage d'un adversaire  $A$  contre la sécurité sémantique, pour une distribution de probabilités  $p$  et une fonction d'information donnée est :*

$$(5) \quad \text{Av}(A) = \left| \Pr[y^* = f(m)] - \Pr[y^* = f(m')] \right|,$$

*où  $y^*$  est la réponse de l'adversaire lorsque son entrée est un cryptogramme du message  $m$ , et où  $m'$  désigne un message aléatoire.*

Le premier terme  $\Pr[y^* = f(m)]$  est la probabilité de succès de l'adversaire  $A$ , et le deuxième terme  $\Pr[y^* = f(m')]$  est la probabilité que sa réponse  $y^*$  soit égale à l'information d'un message aléatoire sans lien avec le cryptogramme qu'il a reçu.



La probabilité exprimée dans cette définition est pour une clé de chiffrement aléatoire  $e$  fournie par l'algorithme de génération de clé, et pour des messages clairs aléatoires suivant la distribution de probabilité  $p$  définie.

**Définition 5.3 [ Schéma de chiffrement sémantiquement sûr ]**

*Un schéma de chiffrement  $\mathcal{C} = (\mathcal{D}, \mathcal{R}, G, E, D)$  est dit sémantiquement sûr – noté SEM-sûr –, si pour toute distribution de probabilité  $p$  et toute fonction d'information  $f$  sur les messages en clair, l'avantage d'un adversaire polynomial est négligeable.*

L'expression de l'avantage donné par la relation (5) exprime que, lorsque le schéma de chiffrement est sémantiquement sûr, la réponse d'un adversaire ne dépend pas significativement du cryptogramme qui lui est donné.



L'adversaire connaît la distribution de probabilités  $p$  et la fonction  $f$ , mais ignore bien sûr la clé de déchiffrement ainsi que le message clair correspondant au cryptogramme qu'il reçoit en entrée.



La sécurité sémantique est atteinte si l'avantage de tout adversaire est négligeable, pour toute distribution de probabilité sur les clairs. En particulier si deux messages clairs seulement sont également probables.

**5.2.2. Indistinguabilité**

L'indistinguabilité est une autre façon d'évaluer la sécurité d'un schéma de chiffrement, moins intuitive, mais plus facile à appliquer en pratique. Un chiffrement a la propriété d'indistinguabilité si un adversaire à qui l'on présente les cryptogrammes de deux messages qu'il a choisis, est incapable de dire de quel message il provient.

La propriété d'indistinguabilité est formalisée par le jeu suivant, entre le maître du jeu  $M$  et un adversaire  $A$  :

JEU IND.

1. L'adversaire  $A$  choisit deux messages  $m_0$  et  $m_1$  qu'il fournit au maître du jeu.
2. Le maître du jeu  $M$  tire un symbole binaire  $b \in \{0, 1\}$  aléatoire avec probabilité uniforme dans, puis chiffre le message  $m_b$  avec une clé  $e$  aléatoire.
3. L'adversaire  $A$  renvoie  $b^*$  qui est son estimation de  $b$ , c'est-à-dire  $b^* = 0$  s'il estime que le cryptogramme qu'il a reçu provient du message  $m_0$  et  $b^* = 1$  s'il estime qu'il provient du message  $m_1$ .

$A$  a gagné si  $b = b^*$ .



Pour que ce jeu ait un sens, il faut que les messages  $m_0$  et  $m_1$  aient la même taille, et que la taille du cryptogramme ne donne pas d'information sur la nature du message qui a été chiffré.

La qualité d'un adversaire contre la propriété d'indistinguabilité se mesure à son avantage :

#### Définition 5.4 [ Avantage d'un adversaire contre l'indistinguabilité ]

*L'avantage de l'adversaire  $A$  contre la propriété d'indistinguabilité est :*

$$\text{Av}(A) = \left| 2 \Pr[b = b^*] - 1 \right|$$



La quantité  $\Pr[b = b^*]$  désigne la probabilité de succès de l'adversaire  $A$ . Son avantage est aussi :

$$\text{Av}(A) = \left| \Pr[b^* = 0 \mid b = 0] - \Pr[b^* = 0 \mid b = 1] \right|.$$

#### Définition 5.5 [ indistinguabilité d'un schéma de chiffrement ]

*On dit qu'un schéma de chiffrement  $\mathcal{C} = (\mathcal{D}, \mathcal{R}, G, E, D)$  indistinguablement sûr – noté IND-sûr –, si l'avantage de tout adversaire contre la propriété d'indistinguabilité est une fonction négligeable.*



La propriété d'indistinguabilité énonce que tout adversaire échoue pour tout couple  $(m_0, m_1)$  de message. On peut supposer sans restriction que l'adversaire peut choisir les deux messages  $m_0$  et  $m_1$  qui lui donneront les meilleures chances de succès.

EXEMPLE. Soit  $E$  la fonction de chiffrement d'un schéma de chiffrement  $\mathcal{C}$ . Soit  $\mathcal{C}'$  le schéma de chiffrement presque identique à  $\mathcal{C}$ , mais dans lequel la fonction de chiffrement a été remplacé par la fonction  $E'$  donnée par :

$$E'(e, m) = \begin{cases} E(e, m) & \text{si } m \neq 1^n, 0^n \\ 0^n & \text{si } m = 0^n \\ 1^n & \text{si } m = 1^n \end{cases}$$

Le schéma de chiffrement  $\mathcal{C}'$  n'est pas sémantiquement sûr, car avec la distribution de probabilité donné par :

$$\Pr[m] = \begin{cases} 1/2 & \text{si } m = 0^n \text{ ou } 1^n \\ 0 & \text{sinon} \end{cases},$$

il est très facile de construire un adversaire contre la sécurité sémantique de  $\mathcal{C}'$  qui donne toujours la bonne réponse pour tout fonction d'information  $f$ .

Le schéma de chiffrement  $\mathcal{C}'$  n'a pas non plus la propriété d'indistinguabilité, car il est immédiat également de construire un adversaire qui sait distinguer entre le chiffré de  $0^n$  et celui de  $1^n$ .



Un schéma de chiffrement à clé publique déterministe n'a jamais la propriété d'indistinguabilité. La clé de chiffrement étant publique, l'adversaire peut toujours chiffrer les messages  $m_0$  et  $m_1$  qu'il a choisi. Comme le cryptogramme est unique pour un message donné, il peut vérifier de quel message le cryptogramme reçu provient. Pour atteindre la propriété d'indistinguabilité, un schéma de chiffrement à clé publique doit obligatoirement être non déterministe.

### 5.2.3. Équivalence des deux notions

La sécurité sémantique correspond à ce qui est intuitivement requis pour avérer la sécurité d'un schéma de chiffrement. L'indistingabilité est une notion plus commode à manipuler. Mais les deux notions sont finalement équivalentes, comme l'énonce le théorème suivant :

#### Théorème 5.6 [ équivalence de la sécurité sémantique et de l'indistingabilité ]

*Un schéma de chiffrement est sémantiquement sûr si et seulement si il a la propriété d'indistingabilité.*

*Preuve.* Montrons tout d'abord l'implication  $\text{SEM} \Rightarrow \text{IND}$ , ou, ce qui est équivalent, la contraposée  $\neg \text{IND} \Rightarrow \neg \text{SEM}$ .

On suppose qu'il existe un algorithme efficace  $A$  et deux messages  $m_0$  et  $m_1$  tel que  $A$  sait distinguer un cryptogramme de  $m_0$  d'un cryptogramme de  $m_1$  avec un avantage non négligeable. Soit  $b$  le choix du maître du jeu et soit  $b^*$  la réponse de  $A$  pour un cryptogramme du message  $m_b$ . Par définition, l'avantage de  $A$  contre l'indistingabilité est :

$$\text{Av}^{\text{IND}}(A) = \left| \Pr[b^* = 1 \mid b = 1] - \Pr[b^* = 1 \mid b = 0] \right|$$

L'algorithme  $A$  est aussi un adversaire contre la sécurité sémantique pour la distribution sur les messages clairs donnée par :

$$\Pr(m) = \begin{cases} 1/2 & \text{si } m = m_0 \text{ ou } m_1 \\ 0 & \text{sinon} \end{cases}$$

et pour la fonction d'information :

$$f(m) = \begin{cases} 1 & \text{si } m = m_0 \\ 0 & \text{si } m = m_1 \end{cases}$$

Montrons que l'avantage de  $A$ , lorsqu'on le considère comme adversaire contre la sécurité sémantique vaut  $\eta(n)/2$ . Pour cela, évaluons la probabilité que  $A$  donne  $f(m)$  selon que  $m$  vaut  $m_0$  ou  $m_1$ . L'avantage de  $A$  contre la sécurité sémantique vaut :

$$\text{Av}^{\text{SEM}}(A) = \left| \Pr[b^* = b] - \Pr[b^* = b'] \right|,$$

où  $b^*$  est la réponse de  $A$  lorsqu'on lui soumet le cryptogramme du message  $m_b$  et où  $b'$  est un symbole binaire aléatoire désignant un message aléatoire parmi  $m_0$  ou  $m_1$ . Comme  $b'$  est indépendant de tout le reste, le second terme vaut  $1/2$ .

Intéressons nous maintenant au premier terme :

$$\begin{aligned} \Pr[b^* = b] &= \underbrace{\Pr[b^* = b \mid b = 0]}_{= \Pr[b^* = 0 \mid b = 0]} \times \underbrace{\Pr[b = 0]}_{= 1/2} + \underbrace{\Pr[b^* = b \mid b = 1]}_{= \Pr[b^* = 0 \mid b = 1]} \times \underbrace{\Pr[b = 1]}_{= 1/2} \\ &= \frac{1}{2} \left( 1 + \Pr[b^* = 0 \mid b = 0] - \Pr[b^* = 0 \mid b = 1] \right) \end{aligned}$$

Finalement, l'avantage de  $A$  comme adversaire contre la sécurité sémantique est :

$$\begin{aligned} \text{Av}^{\text{SEM}}(A) &= \frac{1}{2} \left| \Pr[b^* = 0 \mid b = 0] - \Pr[b^* = 0 \mid b = 1] \right| \\ &= \frac{1}{2} \text{Av}^{\text{IND}}(A) \end{aligned}$$

Ainsi,  $A$  est un adversaire contre la sécurité sémantique dont l'avantage n'est pas négligeable, ce qui prouve que le schéma n'est pas sémantiquement sûr.

Montrons maintenant la réciproque  $\text{IND} \Rightarrow \text{SEM}$ , ou, ce qui est équivalent, la contraposée  $\neg \text{SEM} \Rightarrow \neg \text{IND}$ .

On suppose qu'il existe une distribution de probabilité  $p$  et une fonction d'information  $f$  sur les messages en clair, ainsi qu'un adversaire  $A$  contre la sécurité sémantique dont l'avantage est non négligeable.

Construisons un algorithme  $B$  contre l'indistinguabilité :

ADVERSAIRE  $B$  CONTRE IND

1.  $B$  choisit au hasard deux messages  $m_0$  et  $m_1$  en suivant la loi de probabilité  $p$  qu'il transmet au maître du jeu.
2. Le maître du jeu choisit  $b$  aléatoire dans  $\{0, 1\}$ , chiffre le message  $m_b$  avec une clé aléatoire et transmet le cryptogramme  $c_b$  à  $B$ . L'algorithme  $B$  doit décider si  $c_b$  provient du chiffrement de  $m_0$  ou de  $m_1$ .
3.  $B$  soumet  $c$  à l'algorithme  $A$ . Soit  $y^*$  la réponse de  $A$  qui est son estimation de la valeur  $y = f(m_b)$  avec un certain avantage non nul. On a trois cas :
  1. Si  $y^* = f(m_0)$  alors  $B$  renvoie  $b^* = 0$  ;
  2. Si  $y^* = f(m_1)$  alors  $B$  renvoie  $b^* = 1$  ;
  3. Si  $y^*$  ne vaut ni  $f(m_0)$  ni  $f(m_1)$ , alors cela signifie un échec de l'algorithme  $A$ , et  $B$  répond aléatoirement  $b^* = 0$  ou  $b^* = 1$  avec une probabilité uniforme.

Rappelons que l'avantage de  $A$  contre la sécurité sémantique est :

$$\text{Av}(A) = \left| \Pr[y^* = f(m)] - \Pr[y^* = f(m')] \right|,$$

où  $y^*$  est la réponse de  $A$  lorsqu'on lui soumet le cryptogramme du message  $m$  et où  $m'$  est un message aléatoire.

L'avantage de  $B$  contre l'indistinguabilité est :

$$\text{Av}(B) = \left| \Pr[b^* = 0 \mid b = 0] - \Pr[b^* = 0 \mid b = 1] \right|.$$

Or l'algorithme  $B$  renvoie 0 si  $y^* = f(m_0)$  ou avec probabilité  $1/2$  lorsque  $y^*$  n'est égal ni à  $f(m_0)$  ni à  $f(m_1)$ . D'où :

$$\Pr[b^* = 0 \mid b = 0] = \Pr[y^* = f(m_0) \mid b = 0] + \frac{1}{2} \Pr[f(m_0), f(m_1) \neq y^* \mid b = 0]$$

De même, l'algorithme  $B$  renvoie 1 si  $y^* = f(m_1)$  ou avec probabilité  $1/2$  lorsque  $y^*$  n'est égal ni à  $f(m_0)$  ni à  $f(m_1)$ . D'où :

$$\Pr[b^* = 0 \mid b = 1] = \Pr[y^* = f(m_0) \mid b = 1] + \frac{1}{2} \Pr[f(m_0), f(m_1) \neq y^* \mid b = 1]$$

Dans l'expression de la probabilité que  $y^*$  soit différent de  $m_0$  et  $m_1$ , les deux messages jouent des rôles symétriques. Cette probabilité est donc indépendante de la valeur de  $b$ . Il en résulte que :

$$\Pr[f(m_0), f(m_1) \neq y^* \mid b = 1] = \Pr[f(m_0), f(m_1) \neq y^* \mid b = 0].$$

Et donc par différence :

$$\text{Av}(B) = \left| \Pr[y^* = f(m_0) \mid b = 0] - \Pr[y^* = f(m_0) \mid b = 1] \right|.$$

Dans l'expression ci-dessus, le premier terme est la probabilité que la réponse de l'algorithme  $A$  soit  $f(m_0)$  lorsque c'est bien le cryptogramme de  $m_0$  qui lui est fourni et le deuxième terme est la probabilité que la réponse de l'algorithme  $A$  soit  $f(m_0)$  alors que c'est le cryptogramme d'un autre message  $m_1$  qui lui est transmis. Comme les messages  $m_0$  et  $m_1$  sont choisis aléatoirement avec la loi de probabilité  $p$ , c'est exactement l'expression de l'avantage de  $A$ . L'avantage de l'algorithme  $B$  contre l'indistinguabilité est donc le même que l'avantage de l'algorithme  $A$  contre la sécurité sémantique :

$$\text{Av}(B) = \text{Av}(A).$$

L'algorithme  $B$  est un adversaire contre l'indistinguabilité qui a un avantage non négligeable, ce qui montre que le schéma de chiffrement n'a pas la propriété d'indistinguabilité.  $\square$

## § 5.3 Modèles d'attaques

On introduit ici des interactions entre l'adversaire d'un schéma de chiffrement et le maître du jeu, afin de l'aider dans sa mission. Pour résoudre le problème qui lui est assigné, contre la sécurité sémantique ou contre l'indistinguabilité, il peut disposer de certaines informations. Selon les informations auquel il a accès, cela classe les attaques en deux catégories : les attaques à clair choisi et les attaques à cryptogramme choisi.

### 5.3.1. Attaques à clair choisi

Au cours de son exécution, l'algorithme contre un schéma de chiffrement peut accéder à un oracle de chiffrement qui lui donnera des cryptogrammes de messages qui lui sont soumis. On parle alors d'attaque à clair choisi, noté CPA (*Chosen Plaintext Attack*).

On notera par exemple CPA-IND la propriété d'indistinguabilité d'un schéma de chiffrement pour tout adversaire qui dispose d'un oracle de chiffrement.



Dans un chiffrement à clé publique, la clé de chiffrement étant publique, l'adversaire peut toujours chiffrer les messages de son choix. La notion d'attaque à clair choisi n'est pertinente que pour le chiffrement symétrique, où l'adversaire, ne disposant pas de la clé de chiffrement, doit avoir recours à un oracle pour obtenir des couples (clair, cryptogramme) de son choix.

### 5.3.2. Attaques à cryptogramme choisi

Dans certaines situations concrètes, le modèle d'attaque à clair choisi peut s'avérer insuffisant, et on peut exiger un niveau supplémentaire de sécurité. Imaginons par exemple qu'un adversaire concret emprunte le dispositif de déchiffrement pendant un certain temps, ou bien puisse accéder temporairement au local où le déchiffrement se réalise.

Il faut alors considérer un modèle où l'adversaire peut accéder à un oracle de déchiffrement. Cet oracle rend le message en clair d'un cryptogramme qui lui est soumis. Il s'agit d'attaque à cryptogramme choisi, notée CCA (*Chosen Ciphertext Attack*).



Pour que cette attaque ait un sens, il faut bien sûr interdire les requêtes qui consistent à demander le déchiffrement du défi qui est soumis à l'adversaire.

On notera par exemple CCA-SEM la sécurité sémantique d'un schéma de chiffrement pour tout adversaire qui dispose d'un oracle de déchiffrement.

## § 5.4 Construction d'un schéma de chiffrement IND-CPA

Dans cette section, on montre que le schéma de chiffrement symétrique construit à partir d'une famille de fonctions pseudo-aléatoires a la propriété d'indistinguabilité dans une attaque à clairs choisis.

Rappelons tout d'abord la construction. On suppose qu'on a une famille  $\mathcal{F}_n$  de fonctions pseudo-aléatoires  $\mathcal{F}_n = (f_k)_{k \in \{0,1\}^n}$ , où les  $f_k$  sont des fonctions  $\{0,1\}^n \rightarrow \{0,1\}^n$ .

Le schéma de chiffrement est défini par :

- L'algorithme  $G(1^n)$  de génération de clé choisit aléatoirement une clé  $k$  dans  $\{0,1\}^n$ .
- L'algorithme  $E_k(r, m)$  de chiffrement est un algorithme probabiliste qui, à un aléa  $r$  dans  $\{0,1\}^n$  et un message  $m$  dans  $\{0,1\}^n$  associe le couple  $(r, f_k(r) \oplus m)$ . Dans le cryptogramme, le second terme  $f_k(r) \oplus m$  contient l'information du message, masqué par une valeur pseudo-aléatoire  $f_k(r)$ . La première composante  $r$  permet au détenteur de la clé secrète  $k$  de retrouver le masque.
- L'algorithme de déchiffrement  $D_k(r, c)$  retrouve le message  $m$  par le calcul  $m = c \oplus f_k(r)$ .



La valeur  $f_k(r)$  est une clé de session utilisée uniquement au cours du chiffrement de ce message. On peut potentiellement utiliser la même clé  $k$  pour produire un nombre exponentiel de clés de session.

Ce schéma de chiffrement opère sur des messages de taille fixe, constitués de  $n$  symboles binaires.

Montrons dans la suite de cette section que ce schéma de chiffrement est IND-CPA, c'est-à-dire qu'il a la propriété d'indistinguabilité au cours d'une attaque à clairs choisis.

La première étape de la preuve est de montrer que si on remplace dans le schéma de chiffrement la fonction pseudo-aléatoire  $f_h$  par une fonction parfaitement aléatoire  $h$ , alors l'avantage d'un adversaire ne dépasse pas l'avantage d'un algorithme qui sait discerner une fonction pseudo-aléatoire d'une fonction aléatoire.

L'interaction entre l'adversaire  $A$  du schéma de chiffrement et l'environnement lors d'une attaque est modélisée par un jeu.

Le jeu, noté *jeu 0*, correspond à la situation réelle. L'adversaire  $A$  joue contre un *maître du jeu* qui correspond à l'environnement  $E$ . Les étapes sont les suivantes :

Une clé  $k$  est choisie aléatoirement  $k = G(1^n)$ .

Au cours du jeu, l'adversaire  $A$  accède à un oracle de chiffrement  $f_k$ . L'adversaire soumet à cet oracle un message  $m$ , et l'oracle lui renvoie  $(r, c) = E_k(r, m)$  pour une valeur  $r$ , aléatoire dans  $\{0,1\}^n$ .

L'adversaire  $A$  choisit deux messages  $m_0$  et  $m_1$ .

L'environnement  $E$  choisit un bit aléatoire  $b$  dans  $\{0,1\}$ , puis lance à l'adversaire  $A$  le défi de retrouver  $b$  en lui indiquant le cryptogramme du message  $m_b$  qui vaut  $(\alpha, f_k(\alpha) \oplus m_b)$ .

Ce cryptogramme est un chiffré de  $m_0$  si le bit  $b$  vaut 0, et est un chiffré de  $m_1$  si le bit  $b$  vaut 1.

L'objectif de l'adversaire  $A$  est de savoir si ce cryptogramme est le chiffré de  $m_0$  ou de  $m_1$ .

Après un certain nombre d'appels à l'oracle, l'algorithme  $A$  donne sa réponse. Soit  $b'$  la réponse de  $A$ .

L'algorithme  $A$  a gagné si  $b = b'$  et il a perdu dans le cas contraire.

Notons  $E_0$  l'évènement «  $A$  a gagné dans le jeu 0 ». Définissons l'avantage de  $A$  dans ce jeu comme étant la valeur  $\text{Av}_0^A = 2\Pr(E_0) - 1$ . Il est nul si  $A$  a répondu au hasard, vaut 1 si  $A$  gagne toujours et  $-1$  s'il perd toujours.

Considérons aussi le jeu noté *jeu 1*, similaire au jeu 0, mais au lieu d'utiliser une fonction pseudo-aléatoire  $f_k$  pour le chiffrement, l'environnement utilise une fonction  $h$  parfaitement aléatoire  $\{0,1\}^n \rightarrow \{0,1\}^n$ .

On note  $E_1$  l'évènement «  $A$  gagne au cours du jeu 1 ». L'avantage de  $A$  dans ce jeu est  $\text{Av}_1^A = 2\Pr(E_1) - 1$ .

Montrons le lemme suivant, qui énonce que la différence des probabilités de gain de  $A$  dans le jeu 0 et dans le jeu 1 ne peut pas dépasser l'avantage d'un adversaire contre la famille pseudo-aléatoire.

**Lemme 5.7**

Soit  $\varepsilon_{\text{fpa}}$  l'avantage maximal d'un adversaire contre la famille de fonctions pseudo aléatoires  $(f_k)_{k \in \{0,1\}^n}$ . Alors :

$$|\Pr(E_0) - \Pr(E_1)| \leq \varepsilon_{\text{fpa}}.$$

*Preuve.* Construisons un distinguer  $D$ , qui utilise l'adversaire  $A$ , et qui sait discerner un élément  $f_k$  de la famille pseudo-aléatoire d'une fonction parfaitement aléatoire avec un avantage au moins égal à  $|\Pr(E_0) - \Pr(E_1)|$ .

Le distinguer  $D$  accède à un oracle  $\mathcal{O}$  qui est soit un élément  $f_k$  de la famille de fonctions pseudo-aléatoires, soit une fonction  $h$  parfaitement aléatoire.

Ce distinguer, pour établir sa réponse, va utiliser l'adversaire  $A$  contre notre schéma de chiffrement. Pour cela, il va simuler son environnement, c'est-à-dire fournir à  $A$  tout ce dont il a besoin pour décider. Il va utiliser l'oracle  $\mathcal{O}$  pour générer les clés de session.

Lorsque  $A$  demande à chiffrer un message  $m$ , le distinguer  $D$  choisit un élément  $r$  aléatoire de  $\{0,1\}^n$  et renvoie à  $A$  la valeur  $(r, \mathcal{O}(r) \oplus m)$ .

Lorsque  $A$  fournit les messages  $m_0$  et  $m_1$  pour son défi, le distinguer  $D$  choisit un bit aléatoire  $b \in \{0,1\}$ , une valeur aléatoire  $\alpha \in \{0,1\}^n$  et lance à  $A$  le défi de trouver  $b$  à partir du cryptogramme  $(\alpha, \mathcal{O}(\alpha) \oplus m_n)$ .

L'algorithme  $A$  donne une réponse  $b'$ .

L'heuristique est de dire que, si  $A$  a donné la bonne réponse, c'est que l'oracle réalise une fonction pseudo-aléatoire, alors que s'il a donné la mauvaise réponse, c'est que l'oracle réalise une fonction parfaitement aléatoire. Ainsi, si  $b = b'$ , alors  $D$  renvoie 1 pour signifier que l'oracle réalise une fonction pseudo-aléatoire, et si  $b \neq b'$ , il renvoie 0 pour signifier que l'oracle réalise une fonction aléatoire.

Il reste à évaluer l'avantage du disgingueur  $D$  ainsi construit.

Si l'oracle réalise une fonction pseudo-aléatoire, tout ce que  $D$  simule correspond au jeu 0, alors que si l'oracle réalise une fonction aléatoire, tout ce que  $D$  simule correspond au jeu 1.

Il en résulte que l'avantage de  $D$  est le même que l'avantage de  $A$ , car :

$$\begin{aligned} \varepsilon_{\text{fpa}}^D &= \left| \Pr(D^{f_k}(1^n) = 1) - \Pr(D^h(1^n) = 1) \right| \\ &= \left| \Pr(b = b' \mid \mathcal{O} = f_k) - \Pr(b = b' \mid \mathcal{O} = h) \right| \\ &= |\Pr(E_0) - \Pr(E_1)| \end{aligned}$$

□

Montrons maintenant que le lemme 5.7 suffit à prouver la propriété d'indistinguabilité du schéma de chiffrement dans une attaque à clairs choisis.

Évaluons la probabilité  $\Pr(E_1)$  de gain au cours du jeu 1.

Notons  $D$  l'évènement « la valeur  $\alpha$  dans le cryptogramme que doit décider  $A$  est différente des valeurs aléa  $r_i$  ayant défini les clés de session au cours des  $q$  requêtes ». Dans ce cas, la valeur des  $h(\alpha)$  est parfaitement aléatoire, et la probabilité de gain de  $A$  ne peut pas dépasser  $1/2$  :

$$\Pr(E_1 \mid D) \leq \frac{1}{2}.$$

Par ailleurs, avec une probabilité  $\frac{q}{2^n}$ , la valeur de  $\alpha$  du cryptogramme tombe dans l'une des valeurs  $r_i$  au cours des  $s$  requêtes. Donc :



$$\begin{aligned}
\Pr(E_1) &= \underbrace{\Pr(E_1 \mid D)}_{\leq \frac{1}{2}} \times \underbrace{\Pr(D)}_{\leq 1} + \underbrace{\Pr(E_1 \mid \neg D)}_{\leq 1} \times \underbrace{\Pr(\neg D)}_{= \frac{q}{2^n}} \\
&\leq \frac{1}{2} + \frac{q}{2^n}
\end{aligned}$$

La preuve de la propriété IND-CPA vient ensuite de :

$$\begin{aligned}
\left| \Pr(E_0) - \frac{1}{2} \right| &= \left| \Pr(E_0) - \Pr(E_1) + \Pr(E_1) - \frac{1}{2} \right| \\
&\leq \underbrace{\left| \Pr(E_0) - \Pr(E_1) \right|}_{\leq \varepsilon_{\text{fpa}}} + \underbrace{\left| \Pr(E_1) - \frac{1}{2} \right|}_{\leq \frac{q}{2^n}},
\end{aligned}$$

qui est une fonction négligeable en  $n$ .



Il est crucial pour la preuve de ce résultat, que l'aléa choisi pour définir la clé de session soit parfaitement aléatoire. Si cela n'est pas le cas, la preuve de l'indistinguabilité n'est pas assurée.

## Codes d'authentification

Les schémas de chiffrement présentés jusqu'à présent ne résistent pas à une attaque à cryptogramme choisi. La raison essentielle est qu'il sont *malléables*, c'est-à-dire qu'est possible, sans connaître la clé de manipuler les cryptogramme de façons à obtenir des relations connues sur le clair.

Par exemple, les deux messages en clair  $m$  et  $m^*$  correspondants respectivement aux cryptogrammes  $(r, c)$  et  $(r, c^*)$  du schéma de chiffrement présenté au paragraphe 5.4 vérifient :

$$c + c^* = m + m^*$$

Un adversaire peut alors demander à l'oracle le déchiffrement d'un cryptogramme modifié pour récupérer des informations sur le message en clair qui lui permettront de répondre avec certitude au jeu de l'indistinguabilité.

*Un schéma de chiffrement malléable n'est jamais sûr contre une attaque à cryptogramme choisi.*

Pour rendre un schéma de chiffrement sûr dans une attaque à cryptogramme choisi, il est nécessaire qu'il ne soit pas malléable. Si l'adversaire change le moindre symbole binaire d'un cryptogramme qui lui est donné, il ne doit pas en déduire de relation sur les messages en clair.

Une façon de procéder est de rendre impossible la modification des cryptogramme, et pour cela, une solution est de lui adjoindre une empreinte, un *code d'authentification*, de telle sorte que tout cryptogramme modifié devienne incohérent et donc non déchiffrable.

### § 6.1 Code d'authentification de message

Un code d'authentification de message (*Message Authentication Code*, MAC) est une primitive cryptographique qui calcule une empreinte du message à l'aide d'une clé secrète.

Il s'agit d'une sorte de signature symétrique qui assure l'authentification des messages. Mais la clé pour générer l'empreinte est la même que la clé pour la vérifier. Les acteurs capables de vérifier l'empreinte sont aussi capable de la produire. Un code d'authentification n'assure pas la propriété de non répudiation. Tous les détenteurs de la clé peuvent avoir signé le message. Les empreintes produites par un code d'authentification sont répudiables. Comme la clé est partagée entre le signataire et le vérificateur, il ne sera pas possible que c'est bien le prétendu signataire qui a produit la signature, et non pas un vérificateur malhonnête. De plus, contrairement à une véritable signature, la clé de vérification n'est pas publique. Seuls les détenteurs de la clé peuvent vérifier l'empreinte.

**Définition 6.1 [ Code d'Authentification de Message ]**

Pour un paramètre de sécurité  $n$  donné, un code d'authentification de message (MAC) est la donnée d'un sextuplet  $(\mathcal{D}, \mathcal{C}, \mathcal{K}, G, C, V)$  où :

- a)  $\mathcal{D}$  est le domaine des messages,
- b)  $\mathcal{E}$  est le domaine auquel appartiennent les empreintes produites,
- c)  $\mathcal{K}$  est l'espace des clés,
- d)  $G$  est un algorithme polynomial de génération de clé qui renvoie un élément aléatoire dans  $\mathcal{K}$ ,
- e)  $C$  est un algorithme polynomial qui génère une empreinte d'un message à l'aide d'une clé :

$$C : \begin{array}{ll} \mathcal{D} \times \mathcal{K} & \rightarrow \mathcal{E} \\ (m, k) & \mapsto C(m, k) \end{array}$$

- f)  $V$  est un algorithme de vérification de la validité d'une empreinte pour un message et une clé donnée :

$$V : \begin{array}{ll} \mathcal{D} \times \mathcal{E} \times \mathcal{K} & \rightarrow \{0, 1\} \\ (m, e, k) & \mapsto \begin{cases} 1 & \text{si } C(m, k) = e \\ 0 & \text{sinon.} \end{cases} \end{array}$$



Si la fonction de génération d'empreinte est déterministe, on peut toujours supposer que l'algorithme de vérification est le suivant :

1. Calculer l'empreinte  $e^* = C(m, k)$  ;
2. Si  $e = e^*$  alors rendre 1  
Si  $e \neq e^*$  alors rendre 0.

**§ 6.2 Adversaire d'un code d'authentification**

La sécurité d'un code d'authentification de message se mesure à la difficulté de forger une empreinte sans connaître la clé secrète lors d'une attaque à messages choisis.

L'objectif de l'adversaire est de construire un message et une empreinte valide qui passera l'algorithme de vérification sans disposer de la clé.

Pour cela, il peut interroger un oracle qui lui indiquera des empreintes valides de messages de son choix. Bien sûr, le message produit par l'adversaire doit être différent de tous les messages qu'il a adressé à l'oracle.

La sécurité d'un code d'authentification se formalise par un jeu entre le maître du jeu  $M$  et un adversaire  $A$ .

**JEU DE L'AUTHENTIFICATION**

1. Le maître du jeu  $M$  choisit une clé  $k$  aléatoire dans  $\mathcal{K}$  produite par l'algorithme de génération  $G$ .
2. L'adversaire  $A$  soumet au maître du jeu des messages  $m_i$  qui lui renvoie les empreintes  $r_i = C(m_i, k)$ .
3. Au bout d'un temps polynomial, l'adversaire renvoie un couple  $(m, e)$  de  $\mathcal{D} \times \mathcal{E}$ .

On dit que l'adversaire a gagné si  $C(m, k) = e$ .

Les performances d'un adversaire contre un code d'authentification se mesure à son avantage.

**Définition 6.2 [Avantage d'un adversaire]**

*L'avantage d'un adversaire  $A$  est sa probabilité de succès. Soit  $(m, e)$  sa réponse et soit  $k$  la clé choisie par le maître du jeu :*

$$\text{Av}(A) = \Pr[V(m, e, k) = 1]$$

L'avantage d'un adversaire appartient à l'intervalle borné  $[0, 1]$ . On peut donc poser :

**Définition 6.3 [Sécurité d'un code d'authentification]**

*La sécurité d'un code d'authentification est la borne supérieure des avantages sur l'ensemble  $\mathcal{A}$  de tous ses adversaires.*

$$S^{\text{MAC}} = \sup_{A \in \mathcal{A}} (\text{Av}(A))$$

*On dit qu'un code d'authentification est sûr si sa sécurité est une fonction négligeable du paramètre de sécurité.*



Lorsqu'on dit qu'une fonction est un code d'authentification, on sous-entend qu'elle satisfait la propriété de sécurité donnée dans la définition 6.3 ci-dessus.

**§ 6.3 Construction à partir d'une fonction pseudo-aléatoire**

Une famille de fonctions pseudo-aléatoires définit un code d'authentification :

**Théorème 6.4**

*Soit  $H = (f_s)_{s \in S}$  une famille de fonctions pseudo-aléatoires  $\{0, 1\}^n \rightarrow \{0, 1\}^n$ , alors la fonction*

$$(m, s) \mapsto f_s(m)$$

*est un code d'authentification sur  $\{0, 1\}^n$ .*

*Preuve.* On suppose que la famille  $H$  est une famille de fonctions pseudo-aléatoires, c'est-à-dire que la distance calculatoire  $\Delta(H, \mathcal{F})$  est négligeable. Soit  $A$  un adversaire contre ce code d'authentification et montrons que son avantage est négligeable. Pour cela, construisons un adversaire  $B$  pour le jeu FPA. L'adversaire  $B$  contre  $H$  utilise l'adversaire  $A$  contre le code pour produire sa réponse. Pour cela, il va simuler l'environnement de  $A$  en lui fournissant des empreintes qu'il demandera à l'oracle.

1. Le maître du jeu choisit un élément aléatoire  $b$  dans  $\{0, 1\}$ . Si  $b = 0$ , il choisit une fonction  $f$  aléatoire dans l'ensemble  $\mathcal{F}$  de toutes les fonctions  $\{0, 1\}^n \rightarrow \{0, 1\}^n$ ; si  $b = 1$  il choisit une fonction  $f = f_s$  aléatoire dans  $H$ .
2. L'adversaire  $A$  soumet des messages  $m_i$  à  $B$ . Ce dernier soumet ces mêmes messages au maître du jeu. Il reçoit une réponse  $y_i = f(m_i)$  qu'il transmet à  $A$ .
3. Au bout d'un temps polynomial,  $A$  propose un couple  $(m, y^*)$ .
4.  $B$  soumet  $m$  au maître du jeu qui lui répond une valeur  $y = f(m)$ . Si  $y = y^*$  alors  $A$  a gagné, et  $B$  répond  $b^* = 1$ , sinon,  $A$  a échoué, et  $B$  répond  $b^* = 0$ .

Si  $b = 1$ , alors  $B$  simule auprès de  $A$  un véritable oracle de production d'empreinte. Il utilise l'avantage de  $A$  pour décider que les réponses de l'oracle sont pseudo-aléatoires. Mais si  $b = 0$ , alors l'adversaire  $A$  ne joue pas un véritable jeu, mais un jeu dans lequel les empreintes fournies sont aléatoires. L'échec de  $A$  signifiera probablement cette situation.

Par définition de l'avantage de  $A$  :

$$\begin{aligned} \text{Av}(A) &= \Pr[f(m) = y^* \mid b = 1] \\ &= \Pr[f(m) = y^* \mid b = 1] - \Pr[f(m) = y^* \mid b = 0] + \Pr[f(m) = y^* \mid b = 0] \\ &\leq \left| \Pr[f(m) = y^* \mid b = 1] - \Pr[f(m) = y^* \mid b = 0] \right| + \Pr[f(m) = y^* \mid b = 0] \end{aligned}$$

Majorons les deux termes de cette somme par des quantités négligeables.

Si  $b = 0$ , la valeur de  $f(m)$ , où  $m$  est la réponse de  $A$ , est aléatoire. La probabilité que  $f(m) = y^*$  vaut donc  $1/2^n$ .

D'autre part, par définition de  $B$ , on a

$$\left| \Pr[f(m) = y^* \mid b = 1] - \Pr[f(m) = y^* \mid b = 0] \right| = \left| \Pr[b^* = 1 \mid b = 1] - \Pr[b^* = 1 \mid b = 0] \right|,$$

ce qui correspond exactement à l'avantage de  $B$  dans son jeu FPA. Or, l'avantage de  $B$  est inférieur à la distance calculatoire  $\Delta(H, \mathcal{F})$ , d'où :

$$\text{Av}(A) \leq \Delta(H, \mathcal{F}) + \frac{1}{2^n}.$$

L'avantage de  $A$  est majoré par une somme de deux termes négligeable. Il est donc négligeable lui-même.  $\square$

## § 6.4 Chiffrement CCA-sûr générique

Ce paragraphe présente la construction d'un schéma de chiffrement IND-CCA à partir d'un schéma de chiffrement IND-CPA et d'un code d'authentification sûr lors d'une attaque à messages choisis. Cette construction est générale et ne repose que sur la donnée de :

- un schéma de chiffrement  $(\mathcal{D}, \mathcal{R}, \mathcal{K}, G, E, D)$  qui est IND-CPA ;
- un code d'authentification défini sur le même domaine des messages :  $(\mathcal{D}, \mathcal{E}, \mathcal{K}', G', C, V)$  qui est sûr lors d'une attaque à messages choisis.

On construit alors un schéma de chiffrement comme suit :

- a) Le domaine des messages est le domaine  $\mathcal{D}$  commun au schéma de chiffrement et au code d'authentification.
- b) Le domaine des cryptogrammes est l'ensemble des couples de  $\mathcal{R} \times \mathcal{E}$ , où le premier terme est un cryptogramme du schéma de chiffrement et le deuxième terme est une empreinte produite par le code d'authentification.
- c) L'espace des clés est l'espace  $\mathcal{K} \times \mathcal{K}'$  des couples constitués d'une clé du schéma de chiffrement d'une clé du code d'authentification.
- d) La fonction de génération de clé  $G^*$  génère un couple de clé, l'une pour le schéma de chiffrement, l'autre pour le code d'authentification. En d'autres termes,  $G^*$  renvoie le couple  $(G, G') = (k, k') \in \mathcal{K} \times \mathcal{K}'$ .
- e) Pour chiffrer un message  $m$ , la fonction de chiffrement  $E^*(m)$  renvoie le couple  $(c, e)$ , où  $c$  est le cryptogramme  $c = E(m, k)$  et  $e$  est l'empreinte du cryptogramme  $e = C(c, k)$ .
- f) Pour déchiffrer le cryptogramme  $(c, e)$ , l'algorithme  $E^*$  vérifie tout d'abord si  $V(c, e, k') = 1$ . Si c'est le cas, alors l'algorithme  $E^*$  renvoie  $D(c, k)$ , et dans le cas contraire, il renvoie un symbole  $\perp$  pour signifier l'échec du déchiffrement.

L'adjonction d'un code d'authentification sur le cryptogramme rend le cryptogramme non malléable.

**Théorème 6.5**

Le schéma de chiffrement  $(\mathcal{D}, \mathcal{R} \times \mathcal{E}, \mathcal{K} \times \mathcal{K}', G^*, E^*, D^*)$  est sûr lors d'une attaque à cryptogrammes choisis.

*Preuve.* Soit  $A$  un adversaire contre ce schéma de chiffrement. On suppose qu'il a accès à un oracle de chiffrement et à un oracle de déchiffrement. Montrons que son avantage est négligeable. Pour cela, construisons un adversaire  $B$  qui n'a accès qu'à un oracle de chiffrement pour le schéma  $(\mathcal{D}, \mathcal{R} \times \mathcal{K}, G, E, D)$  qui va simuler l'environnement de  $A$ .

Lorsque  $A$  transmet des requêtes de chiffrement,  $B$  les soumet à l'oracle pour récupérer le cryptogramme  $c$  et va lui-même générer l'empreinte avec une clé produite par lui-même.

Lorsque  $A$  transmet des requêtes de déchiffrement,  $B$  signifie à  $A$  un échec du déchiffrement, sauf pour les requêtes dont il connaît la réponse, c'est-à-dire pour celles qui sont le résultat des requêtes de chiffrement.

Plus précisément, l'adversaire  $B$  est construit ainsi :

- $B$  produit une clé  $k' \in \mathcal{K}'$  par appel à l'algorithme  $G'$ .
- Lorsque  $B$  reçoit de  $A$  la requête de chiffrer un message  $m_i$ , il transmet ce message à l'oracle de chiffrement qui lui répond le cryptogramme  $c_i$ . Il produit l'empreinte  $C(c_i, k') = e_i$  et renvoie le couple  $(c_i, e_i)$  à  $A$ . L'adversaire  $B$  mémorise la liste de tous les couples  $(m_i, (c_i, e_i))$  ainsi produits.
- Lorsque  $B$  reçoit une requête de déchiffrement pour le cryptogramme  $(c_j, e_j)$ , il vérifie si ce cryptogramme est dans sa liste. Si c'est le cas, il renvoie le message en clair correspondant. Dans le cas contraire, il renvoie  $\perp$  pour signifier l'échec du déchiffrement.
- Lorsque  $A$  transmet les deux messages  $m_0$  et  $m_1$  à distinguer,  $B$  les transfère à l'oracle.
- L'oracle choisit un élément aléatoire  $b \in \{0, 1\}$ . Il transmet à  $B$  le cryptogramme  $c_b$  du message  $m_b$ .
- $B$  calcule l'empreinte  $e_b$  de  $c_b$  et transmet le défi  $(c_b, e_b)$  à  $A$ .
- Au bout d'un temps polynomial,  $A$  renvoie à  $B$  une valeur  $b^*$  estimée de  $b$ .
- La réponse de  $B$  est  $b^*$  également.

On note  $q$  le nombre de requêtes de déchiffrement formulées par  $A$  au cours de son travail. Si aucune de ces  $q$  requêtes de déchiffrement de  $A$  n'est effectivement valide, alors  $B$  simule correctement l'oracle auprès de  $A$ . Par contre, s'il existe des requêtes de déchiffrement valide, alors  $B$  ne simule pas correctement l'oracle. Soit  $E$  l'événement « certaines requêtes de déchiffrement formulées par  $A$  sont valides ». On note  $\overline{E}$  l'événement contraire.

Soit  $p$  la probabilité qu'une requête  $(c_j, e_j)$  formulée par  $A$  soit valide, c'est-à-dire que  $e_j = C(c_j, k')$ . Produire un cryptogramme valide  $(c_j, e_j)$  revient à un succès d'un adversaire contre le code d'authentification. La probabilité de succès de cet adversaire est inférieure à la sécurité  $S^{\text{MAC}}$  du code d'authentification.

La probabilité de  $\overline{E}$  est la probabilité qu'aucune requête de déchiffrement ne soit valide et vaut donc  $\Pr[\overline{E}] = (1 - p)^q \sim 1 - qp$ . Il en résulte que :

$$\Pr[E] \sim qp.$$

Cette valeur est inférieure à  $qS^{\text{MAC}}$ , qui est négligeable par hypothèse.

Par définition, l'avantage de  $A$  est :

$$\text{Av}(A) = |2 \Pr[b = b^*] - 1|.$$

On conditionne cette probabilité selon les événements  $E$  et  $\overline{E}$  :

$$\text{Av}(A) = |2 \Pr[b = b^* | E] \times \Pr[E] + 2 \Pr[b = b^* | \overline{E}] \times \Pr[\overline{E}] - 1|.$$

Lorsque l'événement  $\overline{E}$  survient, l'algorithme  $B$  simule exactement un véritable oracle de déchiffrement auprès de  $A$  et dans ce cas, l'avantage de  $B$  est le même que celui de  $A$ . La quantité

$\left| 2 \Pr[b = b^* \mid \overline{E}] - 1 \right|$  vaut donc exactement l'avantage de  $B$  en tant qu'adversaire IND-CPA. Faisons apparaître ce terme :

$$\begin{aligned} \text{Av}(A) &= \left| 2 \Pr[b = b^* \mid E] \times \Pr[E] + \underbrace{2 \Pr[b = b^* \mid \overline{E}] \times \Pr[\overline{E}] - \Pr[\overline{E}]}_{= -\Pr[E]} + \underbrace{\Pr[\overline{E}] - 1}_{= -\Pr[E]} \right| \\ &= \Pr[E] \times \left( 2 \Pr[b = b^* \mid E] - 1 \right) = -\Pr[E] \end{aligned}$$

Par inégalité triangulaire :

$$\begin{aligned} \text{Av}(A) &\leq \underbrace{\Pr[E]}_{\leq qS^{\text{MAC}}} \underbrace{\left| 2 \Pr[b = b^* \mid E] - 1 \right|}_{\leq 1} + \underbrace{\Pr[\overline{E}]}_{\leq 1} \underbrace{\left| 2 \Pr[b = b^* \mid \overline{E}] - 1 \right|}_{\text{Av}(B) \leq S^{\text{CPA}}} \\ &\leq qS^{\text{MAC}} + S^{\text{CPA}} \end{aligned}$$

Il en résulte finalement que :

$$\text{Av}(A) \leq qS^{\text{MAC}} + S^{\text{CPA}},$$

qui est négligeable par hypothèse sur le schéma de chiffrement IND-CPA et le code d'authentification utilisés.  $\square$

## § 6.5 Exercices

**1.** On définit ci-après des schémas de chiffrement. Dans chaque cas dire s'ils ont la propriété d'indistinguabilité dans une attaque à clair choisi, à cryptogramme choisi :

- $E(k, m) = m \oplus f_k(r)$ , où  $\mathcal{F}_n = (f_k)_{k \in \{0,1\}^n}$  est une famille de fonctions pseudo-aléatoires  $\{0,1\}^n \rightarrow \{0,1\}^n$ , et  $r$  est un élément aléatoire de  $\{0,1\}^n$ .
- Le mode dictionnaire d'un chiffrement par bloc. Le message est découpé en blocs de même taille, et chaque bloc est chiffré avec une même fonction de chiffrement :

$$E(k, (m_1, \dots, m_\ell)) = (p_k(m_1), \dots, p_k(m_\ell)),$$

où  $\mathcal{P}_n = (p_k)_{k \in \{0,1\}^n}$  est une famille de permutations aléatoires.

- La fonction RSA.
- Le chiffrement EL GAMAL.
- Le chiffrement de PAILLIER.

**2.** Le schéma de chiffrement défini dans le paragraphe 5.4 reste-t-il sûr face à une attaque à cryptogramme choisi ?

*Indication :* poser des requêtes avec la même clé de session.

**3.** Montrer que le mode ECB n'a pas la propriété IND-CPA.

Montrer que le mode ECB n'est pas sûr, y compris lorsqu'on impose que tous les blocs du message clair sont différents.

**4.** Montrer que le mode CBC avec une valeur prédictible pour le vecteur initial n'est pas sûr.

**5.** Montrer que le mode CBC avec vecteur initial aléatoire est malléable, et n'est donc pas sûr contre une attaque à cryptogramme choisi.

**6.** Dans la preuve du théorème 8.2 page 65, montrer que si l'évènement  $C$  survient, il est possible de construire un message correctement signé.

# Chiffrement par blocs

## § 7.1 Motivation et définition

Une famille de fonctions pseudo-aléatoires permet de réaliser du chiffrement symétrique. Le paramètre d'indice de la famille est la clé secrète  $s$ . Pour chiffrer un message  $m$  de  $n$  symboles binaires, on masque  $m$  avec une valeur  $f_s(x)$  pour une valeur  $x$  aléatoire qui est transmise avec le cryptogramme :

$$\begin{aligned} (s, x) &\longmapsto \boxed{f_s(x)} \\ &\oplus \boxed{m} \\ &= \boxed{c = m \oplus f_s(x)} \end{aligned}$$

Le cryptogramme est  $\gamma = (x, c)$ .

Un défaut de cette méthode est l'obligation d'adjoindre au cryptogramme un paramètre  $x$ , obligatoire pour le déchiffrement. L'opération de chiffrement ne préserve pas la longueur.

L'objet de cette section est de définir des familles de permutations pseudo-aléatoires, qui sont chacune inversible, pour réaliser le chiffrement par bloc. Le cryptogramme est directement l'image du clair par un élément de cette famille. Le clair se retrouve en inversant la fonction.

### Définition 7.1 [ famille de permutations pseudo-aléatoires ]

Une famille  $\mathcal{P}_n = (p_k)_{k \in \{0,1\}^n}$  de permutations  $p_k : \{0,1\}^{p(n)} \rightarrow \{0,1\}^{p(n)}$  est dite pseudo-aléatoire si les trois conditions suivantes sont satisfaites :

1. chaque élément est efficacement calculable : il existe un algorithme efficace  $V$  qui calcule la valeur de tout élément pour tout paramètre  $k$  et toute entrée  $x$  :

$$\forall k \in \{0,1\}^n, \quad \forall x \in \{0,1\}^{p(n)}, \quad V(k, x) = p_k(x).$$

2. chaque élément est efficacement inversible : il existe un algorithme efficace  $V$  qui calcule l'antécédent de tout élément  $y$  pour tout paramètre  $k$  :

$$\forall k \in \{0,1\}^n, \quad \forall y \in \{0,1\}^{p(n)}, \quad N(k, y) = p_k^{-1}(y).$$

3. elle a le caractère pseudo-aléatoire : tout algorithme efficace  $A$  ne sait distinguer un élément de la famille d'une permutation réellement aléatoire qu'avec une probabilité négligeable, c'est-à-dire l'avantage de  $A$  défini par :

$$\text{Av}^A = \left| \Pr_{p \in \mathcal{R}_n} [A^p(1^n) = 1] - \Pr_{f \in \mathcal{R}_n} [A^f(1^n) = 1] \right|$$

où  $\mathcal{R}_n$  désigne l'ensemble de toutes les bijection sur  $\{0,1\}^{p(n)}$ , est une fonction négligeable de  $n$ .



L'algorithme  $A^h$  fait appel à un oracle qui réalise le calcul  $y = h(x)$ . Il renvoie une valeur qui vaut 1 si  $A$  redonnait en  $h$  un élément pseudo-aléatoire et renvoie 0 dans le cas contraire, c'est-à-dire que  $A$  reconnaît en  $h$  un élément aléatoire. Une famille de permutations est pseudo-aléatoire si tout algorithme  $A^h$  ne donne la bonne réponse qu'avec un avantage négligeable.

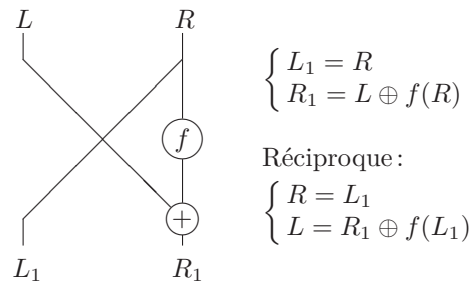


Il existe également une notion plus forte de famille de permutation *super pseudo-aléatoire*, dont les éléments restent indistinguable d'une permutation aléatoire, y compris lorsque l'algorithme  $A^{h,h^{-1}}$  dispose de deux oracles : un pour la permutation, et l'autre pour la permutation inverse.

## § 7.2 Construction à partir d'une famille de FPA

Le schéma de FEISTEL est une construction d'une bijection à partir d'une fonction non nécessairement bijective. C'est cette construction, itérée sur 16 tours, qui est à l'œuvre dans beaucoup d'algorithmes de chiffrement par bloc comme le DES.

Un mot de  $2n$  symboles binaires est partagé en deux parties de  $n$  symboles binaires chacune : la partie gauche notée  $L$  et la partie droite notée  $R$ .



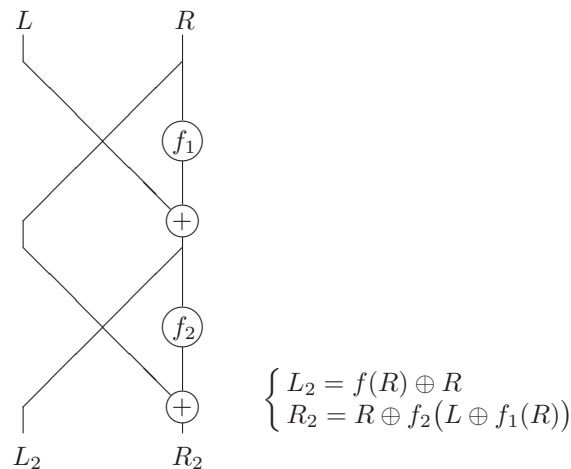
Ce schéma est toujours inversible, y compris lorsque la fonction  $f$  utilisée ne l'est pas.



Même si on suppose que la fonction  $f$  est aléatoire, un tour de FEISTEL n'a pas la propriété d'être pseudo-aléatoire, puisque la partie gauche du résultat est toujours égale à la partie droite.

Il est immédiat de construire un distingueur qui saura presque toujours distinguer entre un tour de FEISTEL et une fonction aléatoire. Il n'échouera que si fortuitement une fonction aléatoire renvoie une valeur gauche égale à son entrée droite, ce qui est négligeable.

Essayons deux tours de FEISTEL :

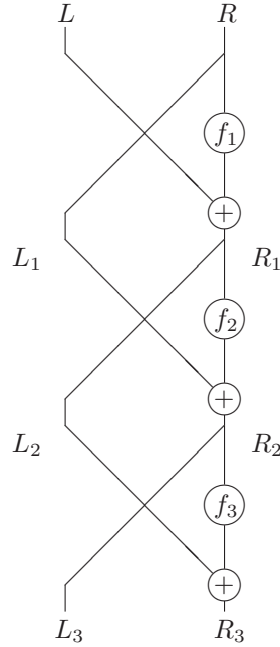


Le schéma à deux tours n'a pas non plus la propriété d'être pseudo-aléatoire. Alors que pour une bijection aléatoire, deux entrées différentes ont des sorties distinctes, aléatoires et indépendantes. Dans le schéma de FEISTEL à deux tours, on a la relation :

$$L \oplus L_2 = f(R).$$

Ainsi en envoyant deux requêtes  $(L, R)$  et  $(L', R)$  à l'oracle avec la même partie droite  $R$ , on peut reconnaître le schéma de FEISTEL à deux tours en calculant la somme modulo 2 des entrées gauche et des sorties gauches  $L \oplus L_2$  et  $L' \oplus L'_2$ . Si on trouve que ces sommes sont égales, alors avec une très forte probabilité, l'oracle est celui d'un schéma de FEISTEL à deux tours. Il est hautement improbable d'avoir cette propriété avec une fonction aléatoire.

Il faut au moins trois tours avec trois fonctions de tour choisie aléatoirement dans une famille pseudo-aléatoire de fonctions pour atteindre la propriété d'être une famille pseudo-aléatoire de permutations, et il faut quatre tours pour atteindre la propriété d'être une famille super pseudo-aléatoire de permutations. Nous allons démontrer la première propriété.



Soit  $\mathcal{F}_n$  une famille de fonctions pseudo-aléatoires de  $\{0, 1\}^n$  vers  $\{0, 1\}^n$ . Pour toute application  $f : \{0, 1\}^n \mapsto \{0, 1\}^n$ , on note  $\text{Feistel}_f$  la bijection :

$$\text{Feistel}_f : \begin{array}{ccc} \{0, 1\}^{2n} & \longrightarrow & \{0, 1\}^n \\ (l, r) & \longmapsto & (r, l \oplus f(r)) \end{array}$$

On note :

$$\text{Feistel}_{f_1 f_2} = \text{Feistel}_{f_2} \circ \text{Feistel}_{f_1}$$

le schéma avec deux itérations utilisant dans cet ordre les fonctions  $f_1$  et  $f_2$  sur  $\{0, 1\}^n$ .

Cela se généralise en posant pour tout entier  $\ell$  :

$$\text{Feistel}_{f_1 \dots f_\ell} = \text{Feistel}_{f_\ell} \circ \text{Feistel}_{f_1 \dots f_{\ell-1}}.$$

Si les fonctions  $f_i$  sont choisies dans la famille pseudo-aléatoire  $\mathcal{F}_n$ , cela définit une famille de bijections. Pour tout entier  $\ell$ , la famille des fonctions de FEISTEL à  $\ell$  tours, construite à partir de la famille  $\mathcal{F}_n$  est notée  $\text{Feistel}_{\mathcal{F}_n}^\ell$  :

$$\text{Feistel}_{\mathcal{F}_n}^\ell = \{ \text{Feistel}_{f_1 \dots f_\ell} \mid f_1, \dots, f_\ell \in \mathcal{F}_n \}.$$

**Théorème 7.2 [ Luby-Rackoff, 1988 ]**

La famille  $\text{Feistel}_{\mathcal{F}_n}^3$  est une famille de permutations pseudo-aléatoires.

*Preuve.* Notons  $\mathcal{R}_n$  la famille de toutes les bijections sur  $\{0, 1\}^n$ . Il faut montrer que les familles  $\text{Feistel}_{\mathcal{F}_n}^3$  et  $\mathcal{R}_{2n}$  sont indistinguables, c'est-à-dire que leur distance calculatoire est négligeable.

La première étape est de montrer que la famille  $\text{Feistel}_{\mathcal{F}_n}^3$  est indistinguable de la famille des schémas de FEISTEL à trois tours où les fonctions de tour sont des fonctions aléatoires. En d'autres termes, si on note  $\mathcal{H}_n$  la famille des fonctions aléatoires de  $\{0, 1\}^n$  vers  $\{0, 1\}^n$ , il s'agit de montrer que les familles  $\text{Feistel}_{\mathcal{F}_n}^3$  et  $\text{Feistel}_{\mathcal{H}_n}^3$  sont indistinguables.

Cela se montre aisément avec la technique hybride. En remplaçant successivement une fonction de tour pseudo-aléatoire par une fonction aléatoire, on peut observer que distinguer ces deux cas revient à distinguer un élément de  $\mathcal{F}_n$  d'un élément de  $\mathcal{H}_n$ , d'où :

$$\Delta(\text{Feistel}_{\mathcal{F}_n}^3, \text{Feistel}_{\mathcal{H}_n}^3) = 3\Delta(\mathcal{F}_n, \mathcal{H}_n)$$

La famille  $\mathcal{F}_n$  étant supposée pseudo-aléatoire, le membre de droite ci-dessus est négligeable.

Montrons maintenant que la famille  $\text{Feistel}_{\mathcal{H}_n}^3$  est indistinguable de la famille  $\mathcal{R}_{2n}$ . Notons pour simplifier  $\mathcal{F}_n^3 = \text{Feistel}_{\mathcal{H}_n}^3$ . On suppose que les trois fonctions  $f_1, f_2$  et  $f_3$  du schéma de FEISTEL sont des fonctions aléatoires.

Si on montre que la valeur n'est pas distinguable d'un aléa, on aura montré qu'un algorithme ne peut pas discerner un schéma de FEISTEL à trois tours avec des fonctions de tour aléatoire d'une permutation aléatoire.

Soit  $R_1^i$  la valeur  $R_1$  interne au schéma de FEISTEL au cours de la  $i^{\text{e}}$  requête. On note  $E_m$  l'évènement :

$E_m$  : « Il existe deux requêtes différentes  $i$  et  $j$ , avec  $i \leq j$  telles que  $R_1^i = R_1^j$  et  $R_2^i = R_2^j$  ».

Évaluons la probabilité de cet évènement.

$$\begin{aligned} \Pr[E_m] &= \Pr[E_{m-1}] + \Pr[E_m \mid \neg E_{m-1}] \times \underbrace{\Pr[E_{m-1}]}_{\leq 1} \\ &\leq \Pr[E_{m-1}] + \Pr[E_m \mid \neg E_{m-1}] \end{aligned}$$

Si l'évènement  $E_m$  a lieu sans que l'évènement  $E_{m-1}$  a lieu, c'est que l'indice  $j$  vaut  $m$ . On a donc  $R_1^i = R_1^m$  pour un indice  $i$  compris entre 1 et  $m-1$  ou bien on a  $R_2^i = R_2^m$  pour un indice  $i$  compris entre 1 et  $m-1$ . Il en résulte :

$$(6) \quad \Pr[E_m \mid \neg E_{m-1}] \leq \sum_{i=1}^m \Pr[R_1^i = R_1^m] + \sum_{i=1}^m \Pr[R_2^i = R_2^m]$$

On est amené à évaluer  $\Pr[R_1^i = R_1^m]$ .

Examinons ce qui se passe au tour précédent au cours des  $i^{\text{e}}$  et  $m^{\text{e}}$  requêtes.

Si  $R_0^i \neq R_0^m$ , comme la fonction  $f_1$  est aléatoire, les valeurs de  $R_1$  au tour  $i$  et  $m$  sont aléatoires avec probabilité uniforme. Elles sont égales avec une probabilité égale à  $1/2^n$ .

Si  $R_0^i = R_0^m$ , alors, comme les requêtes sont distinctes, on a nécessairement  $L_0^i \neq L_0^m$  donc  $R_1^i \neq R_1^m$ .

Pour tout indice  $i$  de requête inférieur à  $m$ , on a  $\Pr[R_1^i = R_1^m] = 1/2^n$ . Donc :

$$\sum_{i=1}^m \Pr[R_1^i = R_1^m] \leq \frac{m-1}{2^n}.$$

On a un résultat identique pour le deuxième terme de (6) en remplaçant l'indice 1 par l'indice 2, donc :

$$\Pr[E_m \mid \neg E_{m-1}] \leq \frac{m-1}{2^{n-1}},$$

et

$$\Pr[E_m] \leq \Pr[E_{m-1}] + \frac{m-1}{2^{n-1}}.$$

En appliquant récursivement cette inégalité pour  $\Pr[E_{m-1}]$ ,  $\Pr[E_{m-2}]$ , ..., on obtient :

$$\Pr[E_m] \leq \frac{(m-1) + (m-2) + \dots + 1}{2^{n-1}} = \frac{n(n-1)}{2 \times 2^{n-1}} \leq \frac{m^2}{2^n}.$$

Cette quantité est négligeable, car  $m$  est borné par un polynôme en  $n$ .

Évaluons maintenant l'avantage d'un adversaire  $A$  supposer distinguer entre  $\mathcal{F}_n^3$  et  $\mathcal{R}_{2n}$ .

$$\text{Av}^A = \left| \Pr_{F \in \mathcal{F}_n^3}[A^F = 1] - \Pr_{F \in \mathcal{R}_{2n}}[A^F = 1] \right|$$

Dans le premier terme, l'algorithme  $A$  reçoit une fonction de  $\mathcal{F}_n^3$ . Pour évaluer la probabilité de succès de  $A$  dans ce cas, conditionnons par l'évènement  $E_m$  :

$$\begin{aligned} \Pr_{F \in \mathcal{F}_n^3}[A^F = 1] &= \underbrace{\Pr[E_m]}_{\leq \frac{m^2}{2^n}} \times \underbrace{\Pr_{F \in \mathcal{F}_n^3}[A^F = 1 \mid E_m]}_{\leq 1} + \\ &\quad \underbrace{\Pr[\neg E_m]}_{\leq 1} \times \Pr_{F \in \mathcal{F}_n^3}[A^F = 1 \mid \neg E_m] \end{aligned}$$

Finalement, on a l'inégalité :

$$\text{Av}^A \leq \frac{m^2}{2^n} + \left| \Pr_{F \in \mathcal{F}_n^3}[A^F = 1 \mid \neg E_m] - \Pr_{F \in \mathcal{R}_{2n}}[A^F = 1] \right|$$

L'évènement  $E_m$  signifie que pour toutes requêtes  $i$  et  $j$ , on a  $R_2^i \neq R_2^j$  et  $R_1^i \neq R_1^j$ . Par conséquent, comme  $f_3$  est une fonction aléatoire, les valeurs  $R_3^i$  sont indépendantes et uniformément distribuées. De même, par passage dans la fonction aléatoire  $f_2$ , les  $R_2^i$  sont indépendants et uniformément distribués sur  $\{0; 1\}^n$ .

Le seul écart à une sortie vraiment aléatoire des couples  $(L_3^i, R_3^i)$  dans le cas d'un schéma de FEISTEL, est qu'on a toujours  $L_3^i \neq L_3^j$  puisque  $L_3 = R_2$ . L'écart à une distribution uniforme est donné par la probabilité de collision de  $m$  valeurs aléatoires parmi  $2^n$  possible, qui, par le paradoxe des anniversaires, est approximativement  $m^2/2^n$ .

Finalement, pour tout adversaire  $A$ , on a  $\text{Av}^A \leq 2 \times \frac{m^2}{2^n}$ . Par passage au maximum, la distance calculatoire  $\Delta(= \mathcal{F}_n^3, \mathcal{R}_n)$  est inférieure à  $2 \times \frac{m^2}{2^n}$  qui est négligeable.

□

## § 7.3 Exercices

1. Considérons la famille des schémas de FEISTEL à trois tours dans laquelle la fonction centrale  $f_2$  est définie par  $f_2 : x \mapsto x \oplus i(x)$ , où la fonction  $i$  est involutive, c'est-à-dire vérifie  $i \circ i(x) = x$  pour tout  $x \in \{0, 1\}^n$ .

a) Montrer que  $f_2 \circ i = f_2$ .

b) En déduire que qu'alors, la famille de bijections obtenue n'est plus pseudo-aléatoire.

*Indication :* chercher deux requêtes différentes qui donneront la même partie gauche  $L_3 = L'_3$  avec ce schéma.

**2.** Soit  $\mathcal{F}_n = (f_k)_{k \in \{0,1\}^n}$  une famille pseudo-aléatoire de bijection de  $\{0,1\}^n$ . Pour tout paramètre  $k$ , on définit la fonction  $g_k$  par :

$$g_k : \begin{array}{ccc} \{0,1\}^{n+1} & \longrightarrow & \{0,1\}^{n+1} \\ (x,b) & \longmapsto & (f_k(x), b \oplus x_0) \end{array}$$

La famille  $\mathcal{G}_n = (g_k)_{k \in \{0,1\}^n}$  est-elle une famille de bijections à est-elle pseudo-aléatoire ?

# Modes d'utilisation

## § 8.1 Chiffrement de messages de tailles variables

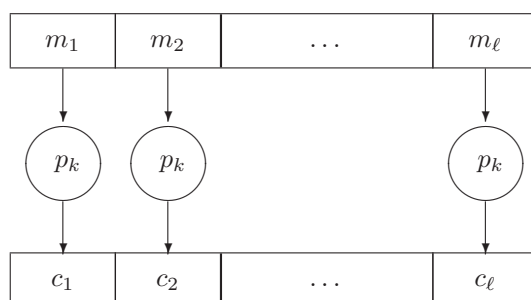
On suppose que l'on dispose d'une famille  $\mathcal{P}_n = (p_k)_{k \in \{0,1\}^n}$  de permutations pseudo-aléatoires  $\{0,1\}^n \rightarrow \{0,1\}^n$ .

La question est : « Comment peut-on chiffrer des messages constitués de blocs de  $n$  symboles binaires ? ».

On suppose pour simplifier que les messages ont une taille qui est multiple de  $n$ . Dans le cas contraire, on peut toujours compléter avec des zéros pour atteindre cette taille.

MODE DICTIONNAIRE, OU MODE ECB (*Electronic Code Book*).

Dans le mode ECB, le message est partagé en blocs de  $n$  symboles binaires. Chaque bloc est chiffré de manière indépendante.

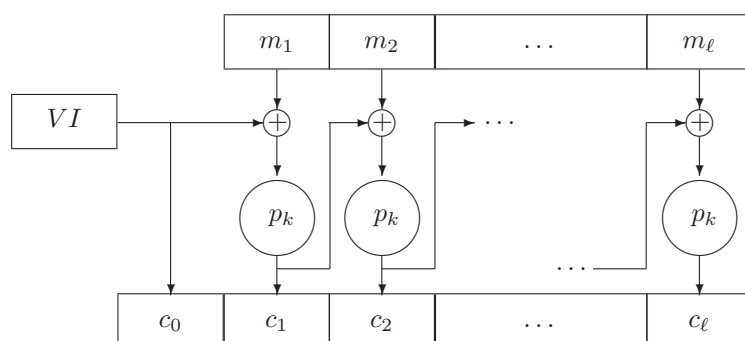


Le mode ECB n'est pas sûr ! Il n'a pas la propriété d'indistinguabilité lors d'une attaque à clairs choisis. Voir l'exercice 3.

MODE CHAÎNÉ, OU MODE CBC (*Cipher Block Chaining*).

Dans le mode CBC, le chiffrement des blocs ne sont plus indépendants les uns des autres. Le chiffrement d'un bloc dépend du cryptogramme du bloc précédent.

Pour cela, il nécessite un vecteur initial (VI).



Les données chiffrées sont rendues aléatoires par l'addition du bloc de cryptogramme précédent.

Le mode CBC est envisageable avec deux possibilités pour le vecteur initial :

- Le vecteur initial peut être un compteur incrémenté à chaque message.
- Le vecteur initial peut être initialisé avec de l'aléa à chaque message.



Lorsque l'adversaire peut prévoir la valeur du vecteur initial, le mode CBC n'est pas sûr. Voir l'exercice 4. En conséquence, il ne peut pas être un compteur.

### Théorème 8.1

|| *Le mode CBC avec vecteur initial aléatoire est IND-CPA.*

*Preuve. (plan)* La méthode de preuve est la même que précédemment. On définit deux jeux pour l'adversaire. L'un, le jeu 0, avec une permutation pseudo aléatoire, l'autre, le jeu 1, avec une permutation parfaitement aléatoire. On montre en deux étapes que les avantages d'un adversaire des jeux 0 et 1 ne diffèrent pas plus que l'avantage d'un distingueur entre une permutations pseudo-aléatoire et une permutation parfaitement aléatoire.

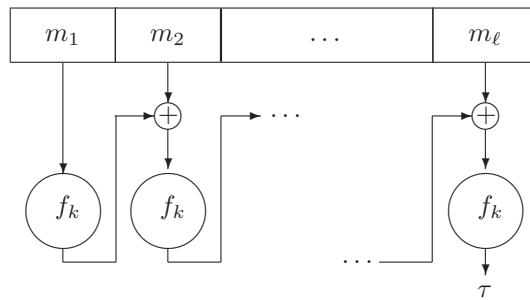
Ensuite, observer les valeurs de  $T_i$  à l'entrée de la permutation à chaque requête de l'adversaire et distinguer le cas où toutes les valeurs de  $T_i$  pour le défi lancé à l'adversaire sont différentes des valeurs atteintes au cours de ses requêtes.

□

#### 8.1.1. Le CBC-MAC

Le code d'authentification de message présenté dans le paragraphe précédent présente peu d'intérêt, car l'empreinte est aussi longue que le message lui-même. Le CBC-MAC permet de proposer des empreintes de taille fixe pour des messages bien plus longs.

Le CBC-MAC est représenté par le schéma ci-dessous :



$$\text{MAC}_k(m_1, \dots, m_\ell) = \tau$$

Ce schéma permet de produire des empreintes pour des messages dont la taille est  $\ell \times n$  bits.



Contrairement au mode CBC pour le chiffrement, il n'est pas nécessaire que la fonction utilisée à chaque tour soit une permutation. Le déchiffrement n'est pas requis. On a seulement besoin d'une fonction pseudo-aléatoire.

**Théorème 8.2 [ Sécurité du CBC-MAC ]**

|| Le CBC-MAC est sûr.

*Preuve.* Comme dans la preuve du théorème 6.4, considérons un adversaire  $A$  contre ce schéma CBC-MAC. On soumet cet algorithme à deux jeux.

Le jeu 0 est le jeu réel, la fonction utilisée à chaque tour du CBC-MAC est une fonction pseudo-aléatoire  $f_k$ . On note  $E_0$  l'évènement « l'algorithme  $A$  a réussi à produire une signature valide au cours du jeu 0 ».

Le jeu 1 est similaire au jeu 0, mais la fonction utilisée à chaque tour du CBC-MAC est une fonction parfaitement aléatoire. On note  $E_1$  l'évènement « l'algorithme  $A$  a réussi à produire une signature valide au cours du jeu 1 ».

On montre, de la même manière que dans la preuve du théorème 6.4 que :

$$(7) \quad |\Pr(E_0) - \Pr(E_1)| \leq \varepsilon_{\text{fpa}},$$

où  $\varepsilon_{\text{fpa}}$  est l'avantage maximal d'un adversaire contre la famille pseudo-aléatoire. Comme dans la preuve du théorème 6.4, on construit un adversaire contre la famille pseudo-aléatoire qui utilise l'adversaire  $A$  en simulant son environnement, et dont l'avantage est par construction donnée par le premier membre de l'inégalité (7).

Évaluons maintenant  $\Pr(E_1)$ . Pendant le déroulement du jeu 1, l'algorithme  $A$  effectue un certain nombre de requêtes de signatures et obtient les couples de messages signés  $(m_1, \tau_1), \dots, (m_q, \tau_q)$ . Considérons l'évènement  $C$  égal à « il existe deux requêtes  $i$  et  $j$  telles que les valeurs de  $f$  à l'avant-dernière itération sont égales ». La probabilité de  $C$  est la probabilité que deux valeurs aléatoires soient égales parmi  $n$ . Elle est approchée par le paradoxe des anniversaires :

$$\Pr(C) \approx \frac{q^2}{2^{n+1}}.$$

En l'absence de collision, la valeur de la fonction  $f$  était parfaitement aléatoires, les requêtes n'apportent aucune information à l'algorithme  $A$  et la probabilité qu'il trouve une signature correcte pour un nouveau message ne dépasse pas le tirage de la valeur au hasard :

$$\Pr(E_1 | \neg C) = \frac{1}{2^n}$$

On a finalement la probabilité

$$\Pr(E_1) = \underbrace{\Pr(E_1 | C)}_{\leq 1} \times \underbrace{\Pr(C)}_{\approx \frac{q^2}{2^{n+1}}} + \underbrace{\Pr(E_1 | \neg C)}_{= \frac{1}{2^n}} \times \underbrace{\Pr(\neg C)}_{\leq 1}$$

qui est une fonction négligeable ainsi que :

$$\Pr(E_0) \leq \Pr(E_1) + \varepsilon_{\text{fpa}}.$$

□



Dans la preuve, on utilise de manière explicite que les messages à signer ont tous la même taille, égale à  $\ell$  blocs de  $n$  bits. Si les messages ont une taille variable, il est possible de forger une fausse empreinte à partir des signatures deux messages bien choisis.

- Soit  $\tau_1$  la signature d'un message d'un seul bloc  $(m_1)$ .
- Soit  $\tau_2$  la signature d'un message d'un seul bloc égal à  $(\tau_1)$ .

Il est immédiat de vérifier que  $\tau_2$  est une empreinte correcte du message constitué de deux blocs  $(m_1, \tau_1)$ .

Pour pallier ce problème, il existe plusieurs remèdes :

- Utiliser pour paramétrer la fonction pseudo-aléatoire, non pas directement la clé  $k$ , mais une clé de session obtenue comme la valeur  $k^* = f_k(\ell)$ , qui est une fonction du nombre de blocs.
- Surchiffrer l'empreinte, qui est obtenue comme  $\tau' = f_{k'}(\tau)$ .





# Chiffrement à clé publique

Un schéma de chiffrement est à clé publique lorsque la clé de déchiffrement  $d$  ne peut pas se calculer de manière efficace à partir de la clé de chiffrement  $e$ . Il n'y a alors aucun inconvénient à publier la clé de chiffrement, ce qui évite d'avoir à communiquer secrètement une clé à son correspondant. Un annuaire public peut contenir la liste des clés de chiffrement de chacun des destinataires. Le secret reste localisé au déchiffrement.

Le chiffrement à clé publique repose sur la notion de *fonction à sens unique avec trappe* ou *porte dérobée*. Ce sont des fonctions difficiles à inverser, sauf pour qui détient une information particulière, appelée la *trappe* qui permet une inversion efficace.

## § 9.1 Problèmes d'appartenance à un sous-groupe

De nombreux schémas de chiffrement à clé publique reposent sur un problème d'appartenance à un sous-groupe. Étant donné sous-groupe  $H$  d'un groupe  $G$ , le problème est de savoir déterminer si un élément  $x$  du groupe  $G$  appartient ou non au sous-groupe  $H$ . Non seulement dans bien des cas, ce problème n'admet pas aujourd'hui de solution efficace, mais il est possible de rendre la solution accessible grâce à une information supplémentaire, conduisant à des fonctions à sens unique avec porte dérobée qui permettent de construire des schémas de chiffrement à clé publique.

### 9.1.1. Le problème de résiduosit  quadratique

Soit  $N = p \times q$  un entier qui est le produit de deux nombres premiers distincts. On considère le groupe  $G$  des entiers  $x$  modulo  $N$  dont la valeur du symbole de JACOBI  $\left(\frac{x}{N}\right)$  vaut 1 :

$$G = \left\{ x \in \mathbb{Z}/N\mathbb{Z} \mid \left(\frac{x}{N}\right) = 1 \right\}$$

Dans  $G$  on considère le sous-groupe  $H$  des éléments qui sont des carrés, ou des résidus quadratiques modulo  $N$  :

$$H = \{ x \in G \mid \exists t \in G, x = t^2 \}$$

Il est difficile de dire si un élément  $x$  de  $G$  appartient ou non à  $H$ . Par contre, si la factorisation  $N = p \times q$  est connue, ce problème devient facile : on calcule les caractères quadratiques  $\left(\frac{x}{p}\right)$  et  $\left(\frac{x}{q}\right)$ . S'ils sont tous deux égaux à  $+1$ , alors l'entier  $x$  appartient à  $H$  et s'ils sont tous deux égaux à  $-1$ , alors il n'appartient pas à  $H$ .

Le système de chiffrement de GOLDWASSER-MICALI repose sur ce problème.

### 9.1.2. Le problème Diffie-Hellmann décisionnel

Soit  $p$  un entier premier et soit  $g$  un générateur d'un sous-groupe de  $\mathbb{Z}/p\mathbb{Z}$  d'ordre premier. On note  $(g)$  le sous-groupe de  $\mathbb{Z}/p\mathbb{Z}$  engendré par  $g$ . On note  $q$  l'ordre de  $(g)$ . L'entier  $q$  est un diviseur de  $p - 1$ .

Soit  $G$  le produit direct  $G = (g) \times (g)$ . Pour un entier  $s$  compris entre 1 et  $q - 1$ , on note  $H_s$  le sous-ensemble de  $G$  défini par :

$$H_s = \{(t, t^s) \mid t \in (g)\}$$

L'ensemble  $H_s$  est un sous-groupe de  $G$ . Ce sous-groupe est engendré par le couple  $(g, g^s)$ .

Si on ne connaît qu'un générateur  $(g, x)$  de ce sous-groupe, sans connaître la valeur de l'entier  $s$ , il est facile de générer les éléments de  $H_s$ , en élevant à une puissance  $r$  aléatoire le générateur  $(g, x)$ , mais il est difficile de décider si un couple  $(y, z)$  appartient ou non à ce sous-groupe.

Le problème de l'appartenance d'un couple  $(y, z)$  au sous-groupe  $H_s$  est connu sous le nom de *problème DIFFIE-HELLMANN décisionnel*. C'est lui qui est sous-jacent dans l'échange de clé éponyme et dans le chiffrement ELGAMAL. La connaissance de l'entier  $s$  agit comme une trappe pour résoudre ce problème. Lorsque  $s$  est connu, on peut décider si le couple  $(y, z)$  appartient ou non au sous-groupe  $H_s$  en calculant  $y^s$  et en vérifiant si  $y^s = z$ .

Un quadruplet  $(g, x, y, z)$  tel que  $x = g^s$  et  $z = y^s$  s'appelle un *quadruplet DIFFIE-HELLMANN*. Il se distingue d'un quadruplet quelconque constitué de deux couples du groupe  $G$  par le fait que  $u = g^r$  et  $z = g^{sr}$ .

### 9.1.3. Le problème des classes de degré composite

Ce problème est celui sur lequel repose le chiffrement de PAILLER. Soit  $N$  un entier qui est le produit de deux entiers premiers distincts :  $N = p \times q$ . On suppose également que  $p$  n'est pas un facteur de  $q - 1$ , ni  $q$  un facteur de  $p - 1$ . Soit  $G$  le groupe des entiers modulo  $N^2$ , c'est-à-dire  $G = \mathbb{Z}/N^2\mathbb{Z}$ . L'ensemble  $H$  des éléments de  $G$  qui sont des puissances  $N$ -ième d'un élément  $r$  de  $\mathbb{Z}/N\mathbb{Z}$  est un sous-groupe de  $G$  :

$$H = \{x \in G \mid \exists r \in \mathbb{Z}/N\mathbb{Z}, x = r^N\}$$

L'ensemble  $H$  est un sous-groupe de  $G$ .

En l'absence de la connaissance de la factorisation de l'entier  $N$ , il est difficile de décider si un élément de  $G$  appartient ou non au sous-groupe  $H$ .

Par contre, la connaissance des facteurs  $p$  et  $q$  de  $N$  permet de résoudre efficacement le problème. Posons :

$$y = a_N + Nb_N$$

l'expression d'un élément  $y$  de  $G$  dans la numération en base  $N$ . Le « chiffre »  $a_N$  est la réduction de  $y$  modulo  $N$ . Si  $y = r^N$ , retrouver  $a_N$  à partir de  $y$  est une instance particulière du RSA. Les hypothèses faites sur  $N$  ont pour conséquence que  $N$  est premier avec  $\lambda(N) = \text{ppcm}(p - 1, q - 1)$ . Posons  $d$  l'inverse de  $N$  modulo  $\lambda(N)$  et  $r^* = y^d \bmod N$ . Une condition nécessaire et suffisante de l'appartenance de l'élément  $y$  au sous-groupe  $H$  est  $(r^*)^N = y$  dans  $\mathbb{Z}/N^2\mathbb{Z}$ .

## § 9.2 Difficulté du problème de l'appartenance

Soit  $(G, \times)_{n \in \mathbb{N}}$  une famille de groupes noté multiplicativement par un entier  $n$  qui est le paramètre de sécurité. Pour simplifier les notations, ce paramètre sera omis dans les notations. Soit  $H$  un sous-groupe de  $G$ . On suppose que :

- les opérations de groupe – multiplication et inverse – sont réalisable par un algorithme efficace, c'est-à-dire de complexité polynomiale en  $n$  /
- les groupes  $G$  et  $H$  sont faciles à échantillonner, c'est-à-dire il existe un algorithme polynomial pour générer un élément aléatoire de  $G$  et de  $H$ .

La difficulté du problème de l'appartenance d'un élément de  $G$  au sous-groupe  $H$  se modélise par un jeu où un adversaire est mis au défi de décider de l'appartenance ou non au sous-groupe  $H$  d'un élément  $x$  qui lui est soumis. Ce jeu fait intervenir deux acteurs : un maître du jeu  $M$  et un adversaire  $A$ .

#### JEU DE L'APPARTENANCE

1. Le maître du jeu choisit un symbole binaire aléatoire  $b \in \{0, 1\}$  avec une probabilité uniforme.
2. Si  $b = 0$ , le maître du jeu choisit un élément  $x$  aléatoire dans  $G$ , et si  $b = 1$  il choisit un élément  $x$  aléatoire dans  $H$ . L'élément  $x$  est transmis à l'adversaire  $A$ .
3. Après un temps polynomial, l'adversaire  $A$  renvoie  $b^*$  qui est son estimation de  $b$ . Si  $b^* = 1$ , il estime que l'élément  $x$  qui lui a été soumis appartient au sous-groupe  $H$  alors que si  $b^* = 0$ , il estime qu'il n'appartient pas à  $H$ .

L'adversaire  $A$  a gagné si  $b = b^*$ . La probabilité de succès de  $A$  est  $\Pr[b = b^*]$ . L'efficacité de l'algorithme  $A$  se mesure à son avantage.

#### Définition 9.1 [ Avantage d'un adversaire ]

*L'avantage de l'adversaire  $A$  pour résoudre le problème de l'appartenance au sous-groupe  $H$  est :*

$$\text{Av}(A) = \left| 2\Pr[b^* = b] - 1 \right|.$$

Une expression de l'avantage de l'adversaire  $A$  est aussi :

$$\text{Av}(A) = \left| \Pr[b^* = 0 \mid b = 0] - \Pr[b^* = 0 \mid b = 1] \right|.$$

Le problème de l'appartenance au sous-groupe  $H$  est ainsi défini :

#### Définition 9.2 [ Difficulté du problème de l'appartenance ]

*On dit que le problème de l'appartenance au sous-groupe  $H$  est difficile si l'avantage de tout adversaire de complexité polynomiale est une fonction négligeable du paramètre de sécurité.*



Il est aujourd'hui admis que les problèmes d'appartenance décrit au paragraphe 9.1 sont tous difficiles sans connaissance de la trappe secrète qui permet de le résoudre.

## § 9.3 Le chiffrement ElGamal est IND-CPA

Dans un schéma de chiffrement à clé publique, la clé de chiffrement est connue de tous. L'adversaire peut donc sans problème chiffrer les cryptogrammes de son choix. Le niveau de sécurité minimal requis pour un tel schéma est la sécurité sémantique – ou l'indistinguabilité – dans une attaque à clair choisi. Dans ce paragraphe, on montre que le schéma de chiffrement ELGAMAL est sûr dans un tel mode.

### 9.3.1. Définition du schéma

Le schéma de chiffrement ELGAMAL est défini comme suit :

Pour un paramètre  $n$  de sécurité :

- a) L'ensemble  $\mathcal{D}$  des messages en clair est l'ensemble  $\mathbb{Z}/p\mathbb{Z}$  des entiers modulo  $p$ , où  $p$  est un nombre premier tel que  $p - 1$  a un facteur premier  $s$  de  $n$  chiffres binaires.
- b) l'ensemble  $\mathcal{R}$  des cryptogrammes est  $\{1, \dots, q - 1\} \times \mathbb{Z}/p\mathbb{Z}$ .
- c) L'algorithme  $G$  de génération de clé génère :

- le nombre premier  $p$  et un facteur premier  $q$  de  $p - 1$ ,
- un entier  $s$  aléatoire compris entre 1 et  $p - 1$ .
- un générateur  $g$  d'ordre  $q$  du sous-groupe  $H$ .

La clé publique de chiffrement est constituée des paramètres  $p$ ,  $g$  et  $x = g^s$ . La clé privée de déchiffrement est  $s$ .

- d) Pour chiffrer un message  $m$  de  $\mathbb{Z}/p\mathbb{Z}$ , il faut :
- Générer un entier  $r$  aléatoire compris entre 1 et  $q - 1$ ,
  - Calculer  $y = g^r$  et  $c = m \times x^r$ .

Le cryptogramme est constitué du couple  $(y, c)$ .

Le deuxième terme contient l'information du message en clair  $m$ , masqué par la valeur  $x^r$ .

Le premier terme  $r$  est un indice qui permet au destinataire dépositaire de la clé privée de calculer le masque.

- e) Pour déchiffrer le cryptogramme  $(y, c)$ , le destinataire :
- calcule le masque  $y^s$  qui est égal à  $x^r$ .
  - reconstitue le message par  $m = c/y^s$ .



Le schéma de chiffrement ELGAMAL n'est pas sûr dans une attaque à cryptogramme choisi. Si l'adversaire demande le déchiffrement du cryptogramme  $(y, 1)$ , il obtient la valeur du masque  $x^r$  qui lui permet de reconstituer le message en clair.

### 9.3.2. INC-CPA sécurité du chiffrement ElGamal

Par contre, le schéma de chiffrement ELGAMAL est sûr dans une attaque à clairs choisis comme l'énonce le théorème suivant :

#### Théorème 9.3 [ IND-CPA sécurité du chiffrement ElGamal ]

*Si le problème DIFFIE-HELLMANN est difficile, alors le chiffrement ELGAMAL est sûr dans une attaque à clairs choisis.*

*Preuve.* On reprend les notations du paragraphe 9.1.2.. On suppose qu'il existe un adversaire  $A$  contre le chiffrement ELGAMAL qui a un avantage non négligeable et on en déduit un adversaire  $B$  contre le problème d'appartenance au sous-groupe  $H_s$ . L'avantage de l'adversaire  $A$  étant supposé non négligeable, sa probabilité de succès est de la forme :

$$\Pr[b^* = b] = \frac{1}{2} + \mu(n),$$

où  $\mu(n)$  est non négligeable.

L'algorithme  $B$  est construit ainsi :

1. Il reçoit du maître du jeu un couple  $(y, z)$  et est mis au défi de savoir si ce couple appartient ou non au sous-groupe  $H_s$ .
2. Il reçoit de  $A$  deux messages  $m_0$  et  $m_1$ .
3. Il choisit un élément  $b$  aléatoire de  $\{0, 1\}$  et transmet à  $A$  le cryptogramme  $(y, z \times m_b)$ .
4. Soit  $b^*$  la réponse de l'adversaire  $A$ . S'il a correctement estimé  $b$ , c'est probablement que le quadruplet  $(g, x, y, z)$  est un véritable quadruplet DIFFIE-HELLMANN vérifiant  $x = g^s$  et  $z = y^s$ . Sa réponse est alors  $\tilde{b} = 1$ . Dans le cas contraire, il renvoie  $\tilde{b} = 0$ .

Estimons maintenant l'avantage de l'algorithme  $B$  ainsi construit. Soit  $\hat{b}$  l'élément de  $\{0, 1\}$  choisi par le maître du jeu pour déterminer si le couple  $(y, z)$  transmis à  $B$  appartient à  $H_s$  ou non. L'avantage de  $B$  est par définition :

$$\text{Av}(B) = \left| \Pr[\tilde{b} = 1 \mid \hat{b} = 1] - \Pr[\tilde{b} = 1 \mid \hat{b} = 0] \right|.$$

Par construction de  $B$ , ceci vaut :

$$(8) \quad \text{Av}(B) = \left| \Pr[b^* = b \mid \widehat{b} = 1] - \Pr[b^* = b \mid \widehat{b} = 0] \right|.$$

Si  $\widehat{b} = 0$ , alors le couple  $(y, z)$  est aléatoire dans  $G$ . Dans ce cas, cryptogramme transmis à  $A$  est parfaitement aléatoire et la probabilité de succès de  $A$  est  $1/2$ . Par contre, si  $\widehat{b} = 1$ , alors le couple  $(y, z)$  appartient à  $H_s$ , et le cryptogramme transmis à  $A$  est celui d'un véritable chiffrement ELGAMAL. Le premier terme du second membre de l'équation (8) est donc la probabilité de succès de  $A$  et le second terme vaut  $1/2$ . Il en résulte que :

$$\text{Av}(B) = \left| \frac{1}{2} + \mu(n) - \frac{1}{2} \right| = |\mu(n)|,$$

qui est non négligeable.

□

## § 9.4 Exercices

1. *Disposer de plusieurs éléments d'un sous-groupe ne change pas la difficulté du problème de l'appartenance.*

Dans le jeu de l'appartenance décrit au paragraphe 9.2 page 68, on suppose qu'à l'étape E, l'adversaire  $A$  reçoit, non pas un seul élément  $x$ , mais un nombre  $m$  d'éléments qui appartiennent soit tous à  $G$  si  $b = 0$ , soit tous à  $H$  si  $b = 1$ .

Montrer que cela ne change pas la difficulté du problème de l'appartenance, c'est-à-dire que s'il n'existe pas d'adversaire polynomial avec avantage non négligeable pour  $m = 1$ , il n'en n'existe pas non plus lorsque  $m$  est un entier borné par un polynôme en  $n$ .

2. *Chiffrement de GOLDWASSER-MICALI (1982).*

Ce chiffrement est le premier chiffrement non déterministe dont la sécurité sémantique a été prouvée. Il repose sur le problème de la résiduosit  quadratique (paragraphe 9.1.1.).

On considère le sch ma de chiffrement d fini comme suit :

a) G n ration de la cl  :

- produire deux nombres premiers distincts  $p$  et  $q$ .
- calculer  $N = p \times q$ .
- g n rer un  l ment  $e$  de  $\mathbb{Z}/N\mathbb{Z}$  tel que les symboles de LEGENDRE  $\left(\frac{e}{p}\right) = \left(\frac{e}{q}\right) = -1$ .

La cl  publique de chiffrement est le couple  $(N, e)$ .

La cl  priv e de d chiffrement est le couple  $(p, q)$ .

b) Chiffrement :

soit  $m = m_1 \cdots m_\ell \in \{0, 1\}^\ell$  le message   chiffrer. Pour chaque indice  $i \in \{1, \dots, \ell\}$  :

- choisir un  l ment  $r_i$  dans  $\mathbb{Z}/N\mathbb{Z}$  tel que le symbole de JACOBI  $\left(\frac{r_i}{N}\right) = 1$ .
- calculer  $c_i = r_i^2 \times e_i^m$ .

Le cryptogramme est l' l ment  $c = (c_1, \dots, c_\ell)$  de  $(\mathbb{Z}/N\mathbb{Z})^\ell$ .

1. Comment d chiffrer le cryptogramme  $c = c_1 \dots c_\ell$  avec la cl  priv e ?
2. V rifier que ce sch ma de chiffrement a la propri t  d'homomorphisme suivante : si  $c$  et  $c'$  sont des cryptogrammes de  $m$  et  $m'$ , alors le produit terme   terme des cryptogrammes  $cc'$  est un cryptogramme de  $m \oplus m'$ .
3. Ce sch ma est-il IND-CCA ?
4. D montrer que si le probl me de r siduosit  quadratique est difficile, alors ce sch ma est IND-CCA.

**3. Chiffrement de PAILLER (1999).**

Ce schéma de chiffrement repose sur la difficulté du problème des classes de degré composite (paragraphe 9.1.3.).

Soit  $N = p \times q$  un entier qui est le produit de deux nombres premiers distincts.

- a) Le domaine des messages en clair est l'ensemble  $\mathbb{Z}/N\mathbb{Z}$ .
- b) La clé publique de chiffrement est l'entier  $N$ , la clé privée de déchiffrement est le couple  $(p, q)$ .
- c) Pour chiffrer le message  $m$  :
  - tirer un entier  $r$  au hasard dans  $\mathbb{Z}/N\mathbb{Z}$ .
  - calculer  $c = (1 + N)^m \times r^N$  dans  $\mathbb{Z}/N^2\mathbb{Z}$ .

Le cryptogramme est le couple  $(r, c)$  du produit direct  $\mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N^2\mathbb{Z}$ .

1. Comment reconstituer le message  $m$  à partir du cryptogramme  $(r, c)$  ?
2. vérifier que ce schéma a la propriété d'homomorphisme suivante: si  $(r, c)$  et  $(r', c')$  sont les cryptogrammes respectifs des messages  $m$  et  $m'$ , alors  $(rr', cc')$  est le cryptogramme de  $m + m'$ .
3. Ce schéma de chiffrement est-il IND-CCA ?
4. Montrer que si le problème des classes de degré composite est difficile, alors ce schéma de chiffrement est IND-CPA.

## Signatures numériques

La signature numérique assure le service d'authentification dans le contexte de la cryptographie à clé publique. Seul le signataire peut signer avec sa clé privée. Tous peuvent vérifier avec la clé publique correspondante.

Le premier schéma de signature asymétrique qui a été inventé est le RSA (1978). On peut transformer le schéma de chiffrement en schéma de signature simplement en inversant le rôle de la clé publique et de la clé privée.

La signature d'un message  $m \in \mathbb{Z}/n\mathbb{Z}$  est la quantité  $\sigma = m^d \bmod n$ , où  $d$  est l'exposant privé, et  $n$  le module public.

Pour vérifier que  $\sigma$  est une signature valide du message  $m$ , on vérifie que  $m = \sigma^e \bmod n$ , où  $e$  est l'exposant public.

Finalement, la signature apparaît ici comme un chiffrement avec la clé privée, la vérification étant un déchiffrement avec la clé publique, ou bien, ce qui revient au même, que la génération de signature est un déchiffrement du message, et la vérification de signature est un chiffrement de la signature.

Ceci donne l'idée que la signature est le dual du chiffrement à clé publique, et que cela nécessite, comme pour le chiffrement d'une fonction à sens unique avec trappe. Cette approche est fautive !

D'une part, il n'est pas toujours possible d'utiliser un schéma de chiffrement pour signer. Par exemple, si le chiffrement est probabiliste, il n'est pas toujours possible d'appliquer l'algorithme de déchiffrement, invoquant la clé privée, à un message arbitraire. Dans ce cas, la signature ne pourra être valide qu'avec une probabilité négligeable.

D'autre part, le chiffrement à clé publique nécessite une fonction à sens unique avec trappe, alors que pour construire un schéma de signature, il suffit de disposer d'une fonction à sens unique. La signature ne nécessite pas de trappe.

Rappelons que l'existence de fonction à sens unique est l'exigence minimale pour réaliser de la cryptographie. La signature, même asymétrique, peut être construite sur cette condition minimale. Même si un jour le chiffrement à clé publique disparaît, il sera peut-être quand même possible de réaliser des signatures à clé publique.

### § 10.1 Les cinq mondes d'Impagliazzo

On ne sait pas aujourd'hui si la famille  $\mathcal{P}$  des problèmes pour lesquels un algorithme efficace existe pour les résoudre est ou non égal à la famille  $\mathcal{NP}$  des problèmes qui sont vérifiables efficacement. Russel IMPAGLIAZZO a imaginé cinq mondes imaginaires possibles selon une hiérarchie de difficulté de résolution des problèmes. Rappelons qu'on dit qu'un problème est facile s'il existe un algorithme efficace pour le résoudre.

1. **Algorithmica.** Ce monde est le monde où  $\mathcal{P} = \mathcal{NP}$ . Dans ce monde, tout les problèmes qui sont faciles à vérifier sont aussi facile à résoudre. La seule cryptographie possible dans Algorithmica est la cryptographie à sécurité inconditionnelle, par exemple reposant sur le masque jetable.
2. **Heuristica.** Dans le monde Heuristica,  $\mathcal{P} \neq \mathcal{NP}$ , c'est-à-dire il existe des problèmes difficiles à résoudre dans le pire des cas qui sont faciles à vérifier, mais tous les problèmes faciles à vérifier sont faciles à résoudre en moyenne.



3. **Pessiland.** Dans le monde Pessiland, il existe des problèmes faciles à vérifier mais difficile à résoudre, y compris dans le pire des cas, mais il n'existe pas de fonction à sens unique. Toutes les fonctions sont inversibles efficacement.
4. **Minicrypt.** Dans le monde Minicrypt, il existe des fonctions à sens unique. Il est possible de trouver des problèmes difficiles qu'on sait résoudre, comme trouver l'antécédent de  $y$  par une fonction à sens unique  $f$ , alors qu'on vient de calculer  $y = f(x)$ . Le monde minicrypt est le monde minimal dans lequel il est possible de faire de la cryptographie symétrique. Dans ce monde, on peut également établir des preuves sans divulgation et des signatures asymétriques.
5. **Cryptomania.** Le monde Cryptomania est le plus proche du monde réel actuel. Dans ce monde, il existe des fonctions à sens unique avec trappe, comme la fonction RSA, qu'on sait inverser lorsque la factorisation du module est connue, mais qui reste difficile pour tous ceux qui ignorent cette factorisation. La cryptographie asymétrique n'est possible que dans le monde cryptomania.

Il se peut que certains de ces mondes disparaissent avec le développement de notre connaissance de la théorie de la complexité. Le premier à disparaître serait Cryptomania. Même si ce dernier venait à disparaître, tant que le monde Minicrypt existe encore, la signature asymétrique restera possible.

Le schéma de signature RSA est construit dans le monde Cryptomania. Il en est de même de la plupart des schémas de signature actuels, comme le DSA, ElGamal, Schnorr, ECDSA, *etc.* La suite de cette section est consacrée à la preuve que la signature numérique appartient au monde Minicrypt. L'objet est de construire une signature asymétrique à partir d'une fonction à sens unique.

## § 10.2 Définition et sécurité des signatures

Définissons tout d'abord ce qu'est un schéma de signature, ainsi que les conditions de sécurité qu'il doit vérifier.

### Définition 10.1 [ Schéma de signature ]

Un schéma de signature numérique  $\pi$ , opérant sur une famille de messages  $\mathcal{M} = (\mathcal{M}_n)_{n \in \mathbb{N}}$ , se compose de trois algorithmes probabilistes : *Gen*, *Sign* et *Verif*.

- L'algorithme de génération des clés  $\text{Gen}(1^n)$  prend un paramètre de sécurité  $1^n$  et retourne le couple  $(k_{\text{pub}}, k_{\text{priv}})$  où  $k_{\text{pub}}$  est la clé publique et  $k_{\text{priv}}$  est la clé privée correspondante.
- Pour le paramètre de sécurité  $1^n$ , l'algorithme de production de signature  $\text{Sign}(k_{\text{pub}}, m)$ , prend comme paramètre une clé privée  $k_{\text{priv}}$  et un message  $m \in \mathcal{M}_n$  et produit une signature  $\sigma$ .
- L'algorithme de vérification  $\text{Verif}(k_{\text{pub}}, m, \sigma)$  est l'algorithme de vérification de la signature  $\sigma$  du message  $m$ . Il invoque la clé publique  $k_{\text{pub}}$ . Il rend une valeur 1 pour signifier que la signature est acceptée, ou bien 0 pour signifier que la signature est rejetée.

Les algorithmes doivent vérifier les conditions suivantes :

- Condition de validité : les signatures qui ont été produites correctement doivent être acceptées. Si  $(k_{\text{pub}}, k_{\text{priv}}) \leftarrow \text{Gen}(1^n)$ , si  $\sigma = \text{Sign}(k_{\text{priv}}, m)$  alors on doit toujours avoir  $\text{Verif}(k_{\text{pub}}, m, \sigma) = 1$ .
- Condition de sécurité : un adversaire ne doit pas pouvoir produire de message correctement signé, y compris lorsqu'il accède à un oracle qui lui signe des documents de son choix. Pour tout algorithme polynomial  $A$  qui accède à l'oracle de signature  $\text{Sign}$ , son avantage, égal par définition à la probabilité que  $A$  produise une signature valide, est négligeable. Pour tout  $(k_{\text{pub}}, k_{\text{priv}}) \leftarrow \text{Gen}(1^n)$ ,

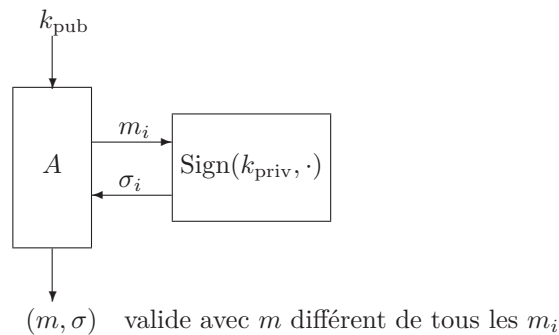
$$\text{Av}(A) = \Pr \left[ A^{\text{Sign}(k_{\text{priv}}, \cdot)}(k_{\text{pub}}) = (m, \sigma) \mid \text{Verif}(k_{\text{pub}}, m, \sigma) = 1 \right],$$

où le message  $m$  n'a pas été demandé à l'oracle de signature  $\text{Sign}(k_{\text{priv}}, \cdot)$ , est négligeable en  $n$ .

On se place ici dans les conditions les plus sévères pour la sécurité. Le faussaire peut faire signer les messages de son choix pour pouvoir produire un message quelconque correctement signé. Il s'agit ici de l'attaque à messages choisis, notée UF-CMA (*Unforgeable under Chosen Messages Attack*).

Il existe des conditions plus faibles de sécurité.

- L'observation passive : l'adversaire peut accéder à un ensemble de messages signés, mais non choisis.
- Pas d'observation : l'adversaire ne peut accéder à aucun message signé. Cette condition est trop faible. En pratique, il est toujours possible d'observer des messages signés.



La signature RSA n'est pas sûre dans une attaque à messages choisis. Par exemple si  $\sigma_1$  est une signature valide d'un message  $m_1$  et si  $\sigma_2$  est une signature valide d'un message  $m_2$ , alors cela signifie que  $\sigma_1^e = m_1$  et  $\sigma_2^e = m_2$ . Il en résulte que le produit  $\sigma_1 \sigma_2$  est une signature valide du message  $m_1 m_2$ , puisque  $(\sigma_1 \sigma_2)^e = \sigma_1^e \sigma_2^e = m_1 m_2$ .

Pour corriger cela, on applique au message une fonction de hachage. Une fonction de hachage idéale est une fonction aléatoire appelée un *oracle aléatoire*. Soit  $H : \mathcal{M} \rightarrow \mathbb{Z}/n\mathbb{Z}$  une fonction

aléatoire, la signature FDH-RSA (*Full Domain Hash RSA*) est calculée par

$$\sigma(m) = H(m)^d \bmod n.$$

Si la fonction  $H$  est aléatoire, c'est-à-dire s'il s'agit d'un oracle aléatoire, alors ce schéma de signature peut être prouvé sûr contre une attaque à messages choisis.

## § 10.3 Le paradigme de Fiat-Shamir

Le paradigme de FIAT-SHAMIR (1986) est une méthode générale qui permet de construire un schéma de signature à partir d'un protocole d'authentification sans divulgation de connaissance (*Zero Knowledge*) et d'un oracle aléatoire.

Rappelons que les protocoles d'authentification sans divulgation sont des protocoles interactifs à trois passes qui permettent de prouver la détention d'une donnée secrète sans révéler aucune information sur cette donnée :

- *Engagement.*
- ← *Défi.*
- *Réponse.*

Une condition de sécurité de ce type de protocole est le caractère aléatoire du défi. Si la fonction  $H$  est un oracle aléatoire, alors l'image d'un message  $m$  par  $H$  est une valeur aléatoire qui peut tenir lieu de défi. On obtient alors un protocole non interactif qui peut faire office de schéma de signature. Il s'agit de prouver que l'on détient bien la clé privée associée à la clé publique  $k_{\text{pub}}$ .

- *Engagement*: la clé publique  $k_{\text{pub}}$ .
- *Défi*:  $H(m)$  le haché du message  $m$  par un oracle aléatoire.
- *Réponse*: la signature  $\sigma$  du message  $m$ .

Considérons par exemple le protocole suivant qui est un protocole sans divulgation, de la connaissance du logarithme discret  $x$  d'une donnée publique  $y$  dans un groupe cyclique  $G$  d'ordre  $q$ , engendré par un générateur  $g$ . Les trois passes de ce protocole sont :

- *Engagement*:  $y = g^x$  et  $u = g^r$  avec  $r$  aléatoire.
- ← *Défi*: une valeur  $k$  aléatoire.
- *Réponse*: la valeur  $t$  telle que  $u = g^t y^k$ .

Ce protocole est sans divulgation, car si on connaît à l'avance la valeur  $k$  du défi, alors il est facile de choisir la réponse  $t$  que l'on donnera, et de s'engager sur le défi  $u = g^t y^k$ . Les trois passes du protocole peuvent par conséquent être simulées et rien ne peut distinguer la simulation d'un protocole réel d'authentification.

Le prouveur qui détient la valeur de  $x$  peut calculer la valeur  $t$  de la réponse en calculant  $t = r - kx \bmod q$ .

Ce protocole peut être transformé en schéma de signature numérique avec un oracle aléatoire  $H(m, u)$ . On obtient alors le schéma de signature de *Schnorr*.

Il est possible de construire une preuve sans divulgation à partir d'une fonction à sens unique, sans nécessairement disposer d'une trappe dans la fonction à sens unique. Cette preuve sans divulgation peut être dérivée en un schéma de signature grâce au paradigme de FIAT-SHAMIR et à un oracle aléatoire qui est prouvée sûre. En conclusion, il est possible de construire un schéma de signature à partir d'une fonction à sens unique et d'un oracle aléatoire. Le problème de cette approche est que la condition de disposer d'un oracle aléatoire est très forte, et les fonctions de hachage réelles sont loin de satisfaire cette hypothèse. Contrairement au modèle de l'oracle aléatoire, dans le modèle standard, on suppose seulement que les fonctions de hachage résistent aux collisions.

La suite de cette section est de construire une signature asymétrique à partir d'une fonction à sens unique dans le modèle standard.

## § 10.4 La signature « une fois » de Lamport

La signature définie dans cette section est une signature jetable, utilisable une fois seulement. Elle ne nécessite qu'une fonction à sens unique  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ .

Pour le paramètre de sécurité  $n$ , l'espace  $\mathcal{M}_n$  des messages est l'ensemble  $\{0, 1\}^n$  des mots binaires de longueur  $n$ . La signature jetable de Lamport signe des messages de longueur fixe égale à  $n$ .

1. L'algorithme Gen de génération de clé procède ainsi :

- Générer aléatoirement  $2n$  éléments de  $\{0, 1\}^n$  qui constituent la clé privée :

$$k_{\text{priv}} = \begin{pmatrix} x_{01} & x_{02} & \cdots & x_{0n} \\ x_{11} & x_{12} & \cdots & x_{1n} \end{pmatrix}$$

- La clé publique est constitué des images des  $x_{ij}$  par la fonction à sens unique  $f$  :

$$k_{\text{pub}} = \begin{pmatrix} y_{01} & y_{02} & \cdots & y_{0n} \\ y_{11} & y_{12} & \cdots & y_{1n} \end{pmatrix}$$

avec, pour  $i \in \{0, 1\}$  et  $j \in \{1, \dots, n\}$ ,  $y_{ij} = f(x_{ij})$ .

2. L'algorithme Sign de signature, pour signer un message  $m$ , fait de  $n$  symboles binaires  $m = m_1, \dots, m_n$ , avec  $m_i \in \{0, 1\}$ , est constitué des éléments de la clé secrète choisis sur la ligne correspondant au symbole  $m_i$ . La signature est :

$$\sigma = (x_{m_1 1}, x_{m_2 2}, \dots, x_{m_n n})$$

3. Pour vérifier la signature  $\sigma = (\sigma_1, \dots, \sigma_n)$  d'un message  $m$ , il suffit alors de vérifier que pour tous les indices  $i$ , on a  $f(\sigma_i) = y_i$ . La signature est considérée comme non valide si pour un des indices  $i$  on a  $f(\sigma_i) \neq y_i$ .



Pour signer un message, on révèle un des secrets. Avec seulement les signatures de deux messages, pourvu qu'ils soient complémentaires l'un de l'autre, toute la clé secrète est révélée. Cette clé n'est utilisable que pour signer un seul message.

En demandant une seule signature d'un message  $m$ , pour produire une signature valide d'un message  $m^* \neq m$ , l'adversaire doit pouvoir inverser  $f$  sur la valeur  $y_i$ , où l'indice  $i$  est l'un de ceux où les messages  $m$  et  $m^*$  diffèrent. La fonction  $f$  étant à sens unique, trouver  $\sigma_i$  tel que  $f(\sigma_i) = y_i$  est supposé difficile.

Montrons la sécurité du schéma de signature une fois de Lamport de manière plus formelle. Pour cela, à partir d'un algorithme  $A$  qui forge une signature, construisons un algorithme  $B$  qui inverse la fonction  $f$  sur une entrée arbitraire  $y$ . En évaluant les avantages de ces algorithmes, on montre la sécurité de ce schéma.

L'algorithme  $B$  va simuler les trois algorithmes du schéma de signature auprès de l'algorithme  $A$ . Le principe est de placer  $y$  à une position dans la clé publique. Lorsque  $A$  demande la signature d'un message  $m$ , il faudra espérer que le symbole  $m_{i^*}$  à la position  $i^*$  permette à  $B$  de proposer une signature valide, et aussi que le message correctement signé que  $A$  fournisse un antécédent de  $y$ .

Tout d'abord,  $B$  choisit une position  $j^*$  et un symbole binaire aléatoire  $i^*$ .

Ensuite,  $B$  impose que  $y$  soit placé en position  $y_{i^* j^*}$  de la clé publique. Les autres entrées des clés publiques et privées sont celles produites par l'algorithme génération de clé standard Gen. C'est-à-dire, pour  $i \neq i^*$  et  $j \neq j^*$ , la valeur de  $x_{ij}$  est aléatoire et  $y_{ij} = f(x_{ij})$ .

Ensuite, l'algorithme  $A$  va demander la signature d'un message  $m = m_1 \cdots m_n$ . Si  $m_{j^*} = i^*$ , ce qui survient avec probabilité  $1/2$ , alors l'algorithme  $B$  échoue. Sinon, il peut révéler les éléments secrets  $(x_{m_1 1}, \dots, x_{m_n n})$ .

L'algorithme  $A$  va proposer un message  $m^+$  et sa signature  $\sigma^+$ . Sur au moins une des positions  $j$ , le message  $m$  diffère du message  $m^+$ . Si  $m_{j^*} = m_{j^*}^+$ , alors l'élément secret révélé pour la signature de  $m^+$  en position  $j^*$  est le même que celui du message  $m$  et l'algorithme  $B$  échoue. Dans le cas contraire, ce qui survient avec une probabilité au moins égale à  $1/n$ , l'algorithme  $A$  propose en position  $j^*$  un antécédent de  $y$  par  $f$  que l'algorithme  $B$  peut renvoyer.

Finalement, l'algorithme  $B$  utilise avec pertinence la réponse de l'algorithme  $A$  avec une probabilité supérieure à  $1/2n$ , et dans ce cas, l'avantage de l'algorithme  $B$  est celui de l'algorithme  $A$ . On a donc :

$$\text{Av}_f(B) \geq \frac{1}{2n} \text{Av}(A),$$

qui reste non négligeable dès que l'avantage de  $A$  est non négligeable, et achève la preuve de la sécurité du schéma de signature jetable de Lamport.

## § 10.5 Une signature « plusieurs fois » avec état

Pour passer à des signatures utilisable plusieurs fois, on utilise une structure d'arbre, similaire à celle utilisée pour construire des fonctions pseudo-aléatoires. La signature dépend d'un état, ce qui n'est pas conforme à la définition, mais constitue une étape avant la construction d'un schéma de signature sans état, ne nécessitant qu'une fonction à sens unique.

La construction de cette signature avec état est possible à partir de n'importe quel schéma de signature une fois  $\pi = (\text{Gen}, \text{Sign}, \text{Verif})$ , et il sera applicable au schéma de signature de Lamport décrit au paragraphe précédent.

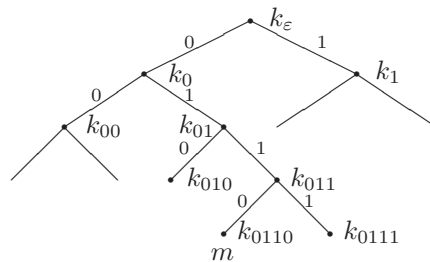
La signature d'un message binaire  $m = m_1 \cdots m_n$  consiste à parcourir un arbre binaire à partir de la racine. Lors de la lecture d'un 0, l'arbre est parcouru à gauche, et lors de la lecture d'un 1, il est parcouru à droite.

Chaque nœud est indexé par un préfixe  $m' = m_1 \cdots m_\ell$  du message. La racine est indexée par le préfixe vide  $\varepsilon$ . Chacun nœud contient également une nouvelle paire de clé  $k_{m'} = (k_{\text{pub}}^{m'}, k_{\text{priv}}^{m'})$ . À la racine, il s'agit de la paire de clé  $k_\varepsilon = (k_{\text{pub}}^\varepsilon, k_{\text{priv}}^\varepsilon)$ . Aux autres nœuds de l'arbre, il s'agit d'une signature qui est générée lorsqu'un nouveau préfixe est parcouru. Pour un nouveau message avec le même préfixe, c'est la même clé qui est utilisée.

Lorsqu'on parcourt l'arbre, la clé publique du nœud courant est ajoutée à la signature produite. Cette clé est authentifiée avec la clé de niveau précédent, constituant un certificat à la manière d'une infrastructure de clés publiques. Au final, la clé  $k_m$  signe le message  $m$ . La signature est constituée de cette signature et des clés publiques certifiées pour tous les préfixes. Le schéma de signature  $\pi$  ne pouvant être utilisé qu'une seule fois, la clé du nœud courant est utilisée pour signer les deux clés publiques du niveau inférieur. La clé racine  $k_\varepsilon$  authentifie le premier préfixe.

De plus, afin de ne pas générer de nouvelles clés inutilement lors de la signature de nouveaux messages, un état mémorisant les préfixes et les clés correspondantes est mis à jour. Une nouvelle clé n'est générée pour un préfixe que si ce préfixe n'appartient pas à l'état. Lorsqu'une nouvelle clé est générée, le préfixe pour lequel elle l'a été est ajouté à l'état.

Voici un exemple pour signer le message  $m = 0110$  :



1. Lors de la lecture du premier symbole 0, deux clés sont générées par appel à la fonction de génération Gen pour générer les clés  $k_0$  et  $k_1$ . La clé privée  $k_{\text{priv}}^\varepsilon$  est utilisée pour produire une signature des deux clés publiques  $k_{\text{pub}}^0$  et  $k_{\text{pub}}^1$ , produisant une signature  $\sigma_\varepsilon$ . Le certificat  $c_\varepsilon$  est la donnée des clés publiques et de leur signature :  $c_\varepsilon = (k_{\text{pub}}^0, k_{\text{pub}}^1, \sigma_\varepsilon)$ .

2. Lors de la lecture du deuxième symbole 1, les clés  $k_{01}$  et  $k_{00}$  sont générées, les clés publiques sont signées avec la clé privée  $k_{\text{priv}}^0$ , produisant une signature  $\sigma_0$ . Le certificat  $c_0$  est  $c_0 = (k_{\text{pub}}^{00}, k_{\text{pub}}^{01}, \sigma_0)$ .

3. Lors de la lecture du troisième symbole 1, les clés  $k_{011}$  et  $k_{010}$  sont générées, les clés publiques sont signées avec la clé privée  $k_{\text{priv}}^{01}$ , produisant une signature  $\sigma_{01}$ . Le certificat  $c_{01}$  est  $c_{01} = (k_{\text{pub}}^{010}, k_{\text{pub}}^{011}, \sigma_{01})$ .

4. Lors de la lecture du dernier symbole 0, les clés  $k_{0110}$  et  $k_{0111}$  sont générées, les clés publiques sont signées avec la clé privée  $k_{\text{priv}}^{011}$ , produisant une signature  $\sigma_{011}$ . Le certificat  $c_{011}$  est  $c_{011} = (k_{\text{pub}}^{0110}, k_{\text{pub}}^{0111}, \sigma_{011})$ .

5. Le message  $m$  est finalement signé avec la clé privée  $k_{0110}$  produisant la signature  $\sigma_{0110}$ . La signature totale du message  $m$  par le schéma  $\pi^*$  est constitué de cette signature et de tous les certificats produits.  $\sigma^* = (c_\varepsilon, c_0, c_{01}, c_{011}, \sigma_{0110})$ .

Pour vérifier cette signature, les signatures des certificats sont vérifiées dans cet ordre, établissant la confiance dans les clés publiques successives, pour finalement vérifier la signature du message  $m$ .

Dans ce schéma, chaque signature n'est utilisée qu'une seule fois, soit pour signer les clés publiques du niveau suivant, soit au final pour signer le message.

Pour un nouveau message, il y a au moins un nœud nouveau. Si un adversaire peut forger une signature pour un nouveau message, alors il peut forger une signature pour le schéma initial  $\pi$  à ce nœud.

Ce schéma de signature ne convient pas encore. Il reste deux problèmes à résoudre :

1. La fonction de génération de signature comprend un état qui est mis à jour au fur et à mesure que des messages sont signés. Ceci sort du cadre de la définition des signatures numériques. Pour se débarrasser de l'état, il faut savoir générer les mêmes signatures pour un préfixe donné. Ceci peut être réalisé en remplaçant l'algorithme probabiliste de génération  $\text{Gen}(1^n)$  par un algorithme déterministe, avec une chaîne aléatoire comme paramètre supplémentaire, alimenté par un générateur pseudo-aléatoire.
2. Les signatures des certificats signent des couples de clés publiques, qui sont deux fois plus longs que la clé servant à signer. Le schéma de Lamport n'est pas directement utilisable. La résolution de ce problème repose sur une fonction de hachage pour que la signature opère sur une donnée de la taille de la clé.

## § 10.6 Fonction de hachage

### 10.6.1. Motivation

Pour signer un message de grande taille, celui-ci est d'abord soumis à une fonction de hachage, afin d'en produire un condensé de taille fixée, et c'est ce condensé qui est soumis à la fonction de signature. Il existe deux notions de sécurité pour les fonctions de hachage :

- la résistance au collision ;
- la notion de fonction de hachage universelle à sens unique.

La seconde notion est moins forte que la première, mais peut être construite à partir d'une fonction à sens unique, alors que la construction de fonctions de hachage à partir de fonctions à sens unique reste un problème ouvert. Il est par ailleurs suffisant de disposer de fonction de hachage universelle à sens unique pour construire un schéma de signature sûr.

### 10.6.2. Définition

L'analyse de la sécurité étant une notion asymptotique, il est nécessaire de définir une famille de fonctions de hachage, selon un paramètre de sécurité.

#### Définition 10.2 [ famille de fonctions de hachage ]

Une famille de fonctions de hachage  $\mathcal{H}$  est constituée de deux algorithmes :

1. un algorithme probabiliste  $\text{Gen}(1^k)$ , où  $k \in \mathbb{N}$  est le paramètre de sécurité, génère un indice  $s$  public, appartenant à un ensemble d'indices  $S$ .
2. Pour indice  $s \in S$ , un algorithme polynomial  $H_s$  calcule, pour tout élément  $x$  de  $\{0,1\}^*$ , une valeur  $H_s(x)$  qui appartient à  $\{0,1\}^{\ell(k)}$ , où  $\ell(x)$  est un polynôme.



En pratique, on utilise toujours une fonction de hachage sans clé, de type SHA 1 ou MD 5, qui correspond à une famille à un seul indice. Ces fonctions produisent un condensé de taille fixe, égale respectivement à 160 et 128 symboles binaires. Toute analyse asymptotique de la sécurité est impossible avec ces fonctions fixées.

### 10.6.3. Notions de sécurité

**1. RÉSISTANCE AUX COLLISIONS.** Une famille de fonctions de hachage est résistante aux collisions (CR, *Collision Resistant*) s'il est difficile de trouver deux éléments  $x$  et  $x'$  dans  $\{0,1\}^*$  qui ont le même condensé.

Un adversaire d'une famille de fonctions de hachage contre les collisions est un algorithme qui produit deux messages différents qui ont le même condensé. L'avantage d'un adversaire est sa probabilité de réussite :

$$\text{Av}_{\mathcal{H}}^{\text{cr}}(A) = \Pr_{s \leftarrow \text{Gen}(1^k)} \left[ A(x) = (x, x') \mid H_s(x) = H_s(x') \text{ et } x \neq x' \right]$$

Cette valeur est une fonction du paramètre de sécurité  $k$ .

#### Définition 10.3 [ résistance aux collisions – CR ]

On dit qu'une famille de fonctions de hachage  $\mathcal{H}$  est résistante aux collisions (CR) si l'avantage  $\text{Av}_{\mathcal{H}}^{\text{cr}}$  de tout adversaire polynomial  $A$  est une fonction négligeable du paramètre de sécurité.

**2. FONCTION DE HACHAGE UNIVERSELLE À SENS UNIQUE (UOWHF *Universal One Way Hash Function*)** Cette propriété de sécurité est une formalisation de la résistance au second antécédent. Étant donné un message  $x$ , il doit être difficile de trouver un second message  $x'$  ayant le même condensé que  $x$ . Un adversaire contre la propriété UOWHF est un algorithme qui produit d'abord un élément  $x$  de  $\{0,1\}^*$  en fonction du paramètre de sécurité, sans connaître l'indice public  $s$ , puis qui doit trouver un second antécédent  $x'$  à  $H_s(x)$ , pour un indice  $s$  aléatoire produit par l'algorithme  $\text{Gen}(1^k)$ .

L'avantage d'un adversaire  $A$  contre la propriété UOWHF est sa probabilité de succès. Soit  $x \leftarrow A(1^k)$  un message choisi par l'adversaire  $A$ .

$$\text{Av}_{\mathcal{H}}^{\text{uowhf}}(A) = \Pr_{s \leftarrow \text{Gen}(1^k)} \left[ A(s) = x' \mid H_s(x) = H_s(x') \text{ et } x \neq x' \right]$$

**Définition 10.4 [ Fonction de hachage universelle à sens unique – UOWHF ]**

On dit qu'une famille de fonctions de hachage  $\mathcal{H}$  est universelle à sens unique (UOWHF) si l'avantage  $\text{Av}_{\mathcal{H}}^{\text{uowhf}}$  de tout adversaire polynomial  $A$  est une fonction négligeable du paramètre de sécurité.

**10.6.4. Une construction de fonction de hachage résistante aux collisions**

Les exercices de ce chapitre proposent des constructions de fonctions de hachage résistantes aux collisions reposant sur le problème du logarithme discret dans un groupe fini, ou sur le problème de la factorisation. Si un ordinateur quantique venait à voir le jour, ces problèmes deviendraient faciles, ce qui rendrait caduques les familles de fonctions de hachage reposant sur ceux-ci. On propose dans cette section la construction d'une famille de fonctions de hachage résistante aux collisions reposant sur le problème du plus court vecteur d'un réseau, qui est un problème qui résisterait à l'existence d'un ordinateur quantique.

Soit  $q$  un entier et  $A$  une matrice de  $m$  lignes et  $n$  colonnes à coefficients dans  $\mathbb{Z}/q\mathbb{Z}$ . Le réseau engendré par  $A$  est l'ensemble des combinaisons linéaires à coefficients entiers des lignes de la matrice  $A$  :

$$\Lambda_q(A) = \{ {}^t s A \bmod q \mid s \in \mathbb{Z}^m \}$$

Il existe aussi le réseau dual qui est l'ensemble des vecteurs à coefficients entiers qui annulent la matrice  $A$  :

$$\Lambda_q^\perp(A) = \{ y \in \mathbb{Z}^m \mid Ay = 0 \bmod q \}$$

AJTAI a montré que pour un choix convenable de  $n$ ,  $m$  et  $q$ , le problème de trouver un élément d'un réseau  $\Lambda_q^\perp(A)$  dont les composantes sont 0, 1 ou  $-1$  modulo  $q$ , est un problème difficile. Ce problème est appelé le problème SIS (*Small Integer Solution*).

En d'autres termes, il est difficile de trouver un vecteur  $z \in \{-1, 0, 1\}^m$  tel que  $Az = 0 \bmod q$ .

La famille de fonctions de hachage définie par :

1.  $\text{Gen}(1^k) \mapsto s = (n, m, q, A)$ , où  $A$  est une matrice de  $n$  lignes et  $m$  colonnes,
2.  $H_s : \begin{array}{ccc} \{0, 1\}^m & \longrightarrow & (\mathbb{Z}/q\mathbb{Z})^n \\ x & \longmapsto & Ax \end{array}$

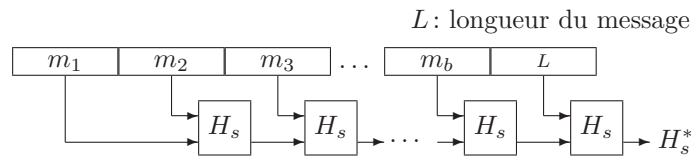
est résistante aux collisions.

En effet, si on peut trouver deux vecteurs  $x$  et  $x'$  qui ont le même condensé, alors on a trouvé un vecteur  $x - x'$  du réseau  $\Lambda_q^\perp(A)$  dont les composantes appartiennent à  $\{-1, 0, 1\}$ , c'est-à-dire une solution du problème SIS.

**10.6.5. La construction de Merkle-Damgård**

La construction du paragraphe précédent opère sur un domaine de taille fixée. L'objet des fonctions de hachage est d'assurer une compression pour produire, à partir d'un long message, un condensé de taille plus réduite. La construction de Merkle-Damgård définit une fonction de hachage sur des messages de longueur quelconque, à partir d'une fonction de facteur de compression égal à 2, c'est-à-dire qui opère sur des messages de  $2\ell$  symboles binaires pour produire des condensés de  $\ell$  symboles binaires. Le principe est similaire à celui du CBC-MAC.

Un long message  $m$  est partagé en  $b$  blocs de  $\ell$  symboles binaires chacun. Chaque bloc entre dans la fonction de hachage  $H_s$ . Le calcul est chaîné selon la figure suivante.





Chaque fonction  $H_s$  est une fonction  $\{0, 1\}^{2\ell} \rightarrow \{0, 1\}^\ell$ .

La longueur  $L$  du message est ajoutée afin de se prémunir contre les collisions construites à partir de la concaténation de messages.

À partir d'une famille de fonctions de hachage  $\mathcal{H} = (\text{Gen}, H)$ , la construction de Merkle-Damgård consiste à produire une famille  $\mathcal{H}^* = (\text{Gen}^*, H^*)$ , où

- l'indice public  $s$  est le même :  $\text{Gen}(1^k) \mapsto s$ , avec  $s \leftarrow \text{Gen}(1^k)$  ;
- la valeur  $H_s^*$  est définie par le schéma ci-dessus.

#### Théorème 10.5 [ sécurité de la construction de Merkle-Damgård ]

*Si la famille  $\mathcal{H}$  est résistante aux collisions, alors la construction de Merkle-Damgård produit une famille de fonctions de hachage  $\mathcal{H}^*$  résistante aux collisions.*

*Preuve.* La preuve est laissée en exercice.

□

## § 10.7 Le paradigme « hache puis signe ».

Ce paradigme correspond à la pratique usuelle de la signature. Pour signer un long document, on commence par lui appliquer une fonction de hachage. On obtient alors un condensé de longueur fixe. Et c'est sur ce condensé que la fonction de signature est appliquée.

### 10.7.1. Avec un hachage résistant aux collisions

À partir de :

- Un schéma de signature  $\pi = (\text{Gen}_\pi, \text{Sign}, \text{Verif})$  et
- Une famille de fonctions de hachage  $\mathcal{H} = (\text{Gen}_\mathcal{H}, H)$ ,

on construit un schéma de signature  $\pi^* = (\text{Gen}_{\pi^*}, \text{Sign}^*, \text{Verif}^*)$  ainsi défini :

- La fonction de génération  $\text{Gen}_{\pi^*}$  appelle la fonction de génération  $\text{Gen}_\pi$  qui produit le couple de clés publique et privée  $(k_{\text{pub}}, k_{\text{priv}})$  et la fonction de génération  $\text{Gen}_\mathcal{H}$  qui produit l'indice public  $s$ . Le résultat de la fonction  $\text{Gen}_{\pi^*}$  est la clé publique  $k_{\text{pub}}^* = (k_{\text{pub}}, s)$  et la clé privée  $k_{\text{priv}}^* = (k_{\text{priv}}, s)$ .
- La génération de signature  $\text{Sign}^*$  applique la génération de signature  $\text{Sign}$  au condensé du message  $m$  par la fonction de hachage  $H_s$ . Elle est définie par :

$$\sigma^* = \text{Sign}^*(k_{\text{priv}}^*, m) = \text{Sign}(k_{\text{priv}}, H_s(m)).$$

- La vérification de signature  $\text{Verif}^*$  applique la fonction de vérification de signature  $\text{Verif}$  au même condensé du message  $m$ , soit :

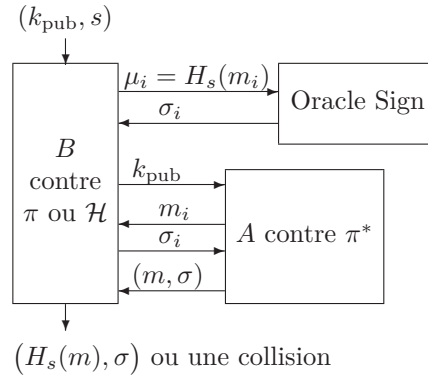
$$\text{Verif}^*(k_{\text{pub}}^*, \sigma^*, m) = \text{Verif}(k_{\text{pub}}, \sigma^*, H_s(m))$$

L'intérêt de cette façon de faire est d'utiliser une signature sur des données courtes pour signer de longs documents. Le théorème suivant montre la sécurité du schéma  $\pi^*$ .

**Théorème 10.6**

Si le schéma de signature  $\pi$  est sûr contre une attaque à messages choisis, et si la famille de fonctions de hachage  $\mathcal{H}$  est résistante aux collisions, alors le schéma de signature  $\pi^*$  est sûr contre une attaque à messages choisis.

*Preuve.* Supposons qu'il existe un adversaire  $A$  contre le schéma  $\pi^*$  et déduisons-en un adversaire  $B$ , soit contre  $\pi$ , soit contre  $\mathcal{H}$ .



L'entrée de l'algorithme  $B$  est une clé publique pour le schéma  $\pi^*$ , c'est-à-dire un couple  $(k_{\text{pub}}, s)$ . La composante  $k_{\text{pub}}$  est directement fournie à  $A$ .

Lorsque l'algorithme  $A$  demande la signature d'un message  $m_i$ , l'algorithme lui applique la fonction de hachage  $H_s$ , transmet le condensé  $\mu_i = H_s(m)$  à l'oracle de signature  $\text{Sign}$ , qui retourne une signature  $\sigma$  directement transmise à l'algorithme  $A$ .

À la fin de son travail, l'algorithme  $A$  fournit un message  $m$  et une signature  $\sigma$ . Le message  $m$  est différent de tous les messages  $m_i$  dont  $A$  a demandé la signature.

Si le haché  $H_s(m)$  est égal à l'un des hachés  $H(m_i)$ , alors l'algorithme  $B$  a trouvé une collision sur  $H_s$ .

Dans le cas contraire, il peut fournir le couple  $(H_s(m), \sigma)$  qui est une signature forgée avec le même avantage que l'algorithme  $A$ .

□

**10.7.2. Avec un hachage universel à sens unique**

L'objet est maintenant d'affaiblir l'hypothèse sur la fonction de hachage de manière à pouvoir se contenter d'une famille de fonctions de hachage universelles à sens unique, dont on sait qu'elles sont dans le monde minicrypt, c'est-à-dire constructible à partir d'une famille de fonctions à sens unique.

Le principe est similaire au précédent. La différence réside dans la génération de l'indice public  $s$  qui n'est pas faite une fois pour toutes, mais qui est faite à chaque signature.

À partir du schéma de signature  $\pi = (\text{Gen}_\pi, \text{Sign}, \text{Verif})$  et de la famille de fonctions de hachage  $\mathcal{H} = (\text{Gen}_\mathcal{H}, H)$ , on construit un schéma de signature  $\pi^* = (\text{Gen}_{\pi^*}, \text{Sign}^*, \text{Verif}^*)$  de la façon suivante :

- Pour le paramètre de sécurité  $k$ , la fonction de génération  $\text{Gen}_{\pi^*}(1^k)$  produit la clé publique  $k_{\text{pub}}^* = k_{\text{pub}}$  et la clé privée  $k_{\text{priv}}^* = k_{\text{priv}}$  par simple appel à la fonction  $\text{Gen}_\pi(1^k)$ .
- La génération de signature  $\text{Sign}^*$  appelle la génération de l'indice public de la famille de fonctions de hachage  $s \leftarrow \text{Gen}_\mathcal{H}(1^k)$ , puis applique la génération de signature  $\text{Sign}$  au condensé du message  $m$  par la fonction de hachage  $H_s$ . Pour que la vérification soit possible, l'indice public  $s$  est fourni avec la signature, et pour s'assurer que cet indice est authentique, il est concaténé au condensé pour être inclus dans la donnée signée. La fonction de génération est donc définie par :

$$\sigma^* = \text{Sign}^*(k_{\text{priv}}^*, m) = (s, \text{Sign}(k_{\text{priv}}, s || H_s(m))).$$

- La vérification de signature  $\text{Verif}^*$ , calcule le haché  $H_s(m)$  avec l'indice public inclus dans la signature, puis applique la fonction de vérification de signature  $\text{Verif}$  à la donnée  $s||H_s(m)$ . Soit :

$$\text{Verif}^*(k_{\text{pub}}^*, \sigma^*, m) = \text{Verif}(k_{\text{pub}}, \sigma^*, s||H_s(m))$$

Cette construction fournit une signature sûre avec seulement la condition que la famille de fonctions de hachage  $\mathcal{H}$  est universelle à sens unique.

### Théorème 10.7

*Si le schéma de signature  $\pi$  est sûr contre une attaque à messages choisis, et si la famille de fonctions de hachage  $\mathcal{H}$  est universelle à sens unique, alors le schéma de signature  $\pi^*$  défini ci-dessus est sûr contre une attaque à messages choisis.*

*Preuve.* La preuve est similaire à celle du théorème 10.6 et est laissée en exercice.

□



L'indice public étant concaténé au condensé  $H_s(m)$ , il ne convient pas si cet indice est trop long. Pour pallier cela, on utilise une seconde famille de fonctions de hachage  $\mathcal{H}'$  avec un indice public  $s'$  choisi à la génération des clés et utilisé pour condenser l'indice  $s$  pour sa signature.

Le schéma est alors le suivant : Étant donné un schéma de signature  $\pi = (\text{Gen}_\pi, \text{Sign}, \text{Verif})$  et deux familles de fonctions de hachage  $\mathcal{H} = (\text{Gen}_\mathcal{H}, H)$  et  $\mathcal{H}' = (\text{Gen}_{\mathcal{H}'}, H')$ , on construit un schéma de signature  $\pi^* = (\text{Gen}_{\pi^*}, \text{Sign}^*, \text{Verif}^*)$  de la façon suivante :

- Génération de clé :

$$\text{Gen}^*(1^k) = (k_{\text{pub}}^*, k_{\text{priv}}^*),$$

où  $k_{\text{pub}}^* = (s', k_{\text{pub}})$  et  $k_{\text{priv}}^* = (s', k_{\text{priv}})$  avec  $s' \leftarrow \text{Gen}_{\mathcal{H}'}(1^k)$  et  $(k_{\text{pub}}, k_{\text{priv}}) \leftarrow \text{Gen}_\pi(1^k)$ .

- Production de la signature : à chaque signature, un indice public  $s$  est produit par  $s \leftarrow \text{Gen}_\mathcal{H}(1^k)$ , puis la signature du message  $m$  est :

$$\sigma^* = \text{Sign}^*(k_{\text{priv}}^*, m) = (s, \text{Sign}(k_{\text{priv}}, H_{s'}(s)||H_s(m))).$$

- Vérification de la signature :

$$\text{Verif}^*(k_{\text{pub}}^*, \sigma^*, m) = \text{Verif}(k_{\text{pub}}, \sigma^*, H_{s'}(s)||H_s(m)),$$

où l'indice  $s$  fait partie de la signature, et l'indice  $s'$  de la clé publique.

## § 10.8 Fonctions de hachage universelles à sens unique

Ce paragraphe présente la construction d'une famille de fonctions de hachage universelle à sens unique à partir d'une famille de permutations à sens unique. Comme cette dernière peut être construite à partir d'une famille de fonctions à sens unique, cela achèvera de montrer que la signature numérique asymétrique appartient au monde Minicrypt.

La construction repose sur une famille universelle deux vers un on suppose l'existence dans un premier temps. On suppose l'existence d'une famille de fonctions de hachage  $\mathcal{H}$  ayant les propriétés suivantes :

- Toutes les fonctions de la famille sont des fonctions *deux vers un*, ce qui signifie que pour toute valeur du paramètre de sécurité  $k$  et tout élément de l'ensemble d'arrivée a exactement deux antécédents. Ceci implique en particulier que l'ensemble de départ  $\mathcal{D}_k$  a un cardinal double de l'ensemble d'arrivée  $\mathcal{U}_k$ .

- Pour tout couple  $(x, x')$  d'éléments distincts de  $\mathcal{D}_k$ , il existe un algorithme  $S(x, y)$  qui produit un indice public  $s$  d'une fonction  $H_s$  vérifiant  $H_s(x) = H_s(x')$  et tel que la production d'un indice  $s$  avec l'algorithme  $S$  pour un élément  $x$  fixé de  $\mathcal{D}_k$  et un élément  $x'$  aléatoire de  $\mathcal{D}_k$  revient exactement à produire l'indice  $s$  avec l'algorithme de génération  $\text{Gen}(1^k)$ .

On suppose aussi l'existence d'une famille  $\mathcal{F}$  de permutations à sens unique  $f_k : \mathcal{D}_k \rightarrow \mathcal{D}_k$ , indexée par le paramètre de sécurité  $k$ .

À partir des familles  $\mathcal{H}$  et  $\mathcal{F}$ , on construit la famille de fonctions de hachage Hash de la façon suivante :

- La fonction de génération est celle de la famille  $\mathcal{H}$  :  $\text{Gen}_{\text{Hash}}(1^k) \mapsto s$  avec  $s \leftarrow \text{Gen}_{\mathcal{H}}$ .
- Les fonctions de la famille calculent la valeur d'un élément du domaine  $\mathcal{D}_k$  en composant préalablement avec la permutation à sens unique :  $\text{Hash}_s : \mathcal{D}_k \rightarrow \mathcal{D}_k$   

$$x \mapsto H_s(f_k(x))$$

Le résultat de cette section est :

### Théorème 10.8

La famille Hash définie ci-dessus est une famille de fonctions de hachage universelles à sens unique (UOWHF)

*Preuve.* Montrons que si la famille Hash n'a pas la propriété d'être une famille de fonctions de hachage universelles à sens unique, alors il est possible d'inverser les éléments de la famille  $\mathcal{F}$ , ce qui contredira l'hypothèse qu'il s'agit d'une famille de permutations à sens unique. Pour cela, supposons l'existence d'un adversaire  $A$  contre la propriété UONHF de la famille Hash et déduisons en un adversaire  $B$  contre la famille  $\mathcal{F}$ .

L'entrée de  $B$  est un élément  $y$  de  $\mathcal{D}_k$  et son objectif est de trouver un antécédent  $\alpha$  par  $f_k$ . Pour atteindre cet objectif, l'algorithme  $B$  va utiliser l'algorithme  $A$  contre la famille Hash.

L'algorithme  $B$  commence par exécuter l'algorithme  $A(1^k)$  qui retourne un élément  $x$  de  $\mathcal{D}_k$ . On pose  $z = f_k(x)$ . On peut supposer que  $z \neq y$ , car dans le cas contraire, un antécédent de  $y$  est trouvé. L'indice public passé en paramètre à l'algorithme  $A$  est l'indice retourné par l'algorithme  $S(z, y)$ . L'hypothèse faite sur la famille  $\mathcal{H}$  indique que cet indice est comme s'il était produit par l'algorithme  $\text{Gen}_{\mathcal{H}}$ . Les éléments  $y$  et  $z$  vérifient par construction  $H_s(y) = H_s(z)$ .

L'algorithme  $A(s)$  retourne un second antécédent de par  $\text{Hash}_s$ , c'est-à-dire un élément  $x'$  de  $\mathcal{D}_k$  tel que  $\text{Hash}_s(x) = H_s(f_k(x)) = H_s(z) = \text{Hash}_s(x') = H_s(f_k(x'))$ . Comme les éléments de  $\mathcal{D}_k$  ont tous exactement deux antécédents par  $H_s$ , l'un d'eux est  $z$  et l'autre est  $y$ , donc  $f(x') = y$ .

□

Il ne reste plus maintenant qu'à exhiber une famille de fonctions de hachage  $\mathcal{H}$  qui vérifie les hypothèses requises. Une telle famille est appelée une *famille universelle* et est utilisée pour limiter les collisions dans les tables de recherches par clé de hachage.

Pour toute valeur  $k$  du paramètre de sécurité, on identifie l'ensemble des vecteurs de  $k$  symboles binaires avec le corps  $\mathbb{F}_{2^k}$ .

- La fonction de génération  $\text{Gen}_{\mathcal{H}}$  produit un couple  $(a, b)$  aléatoire dans  $\mathbb{F}_{2^k} \setminus \{0\} \times \mathbb{F}_{2^k}$ .
- La fonction dont l'indice public est le couple  $(a, b)$  est :

$$H_{ab} : \mathbb{F}_{2^k} \rightarrow (\mathbb{F}_2)^{k-1}$$

$$x \mapsto H_{ab}(x) = p(ax + b),$$

où  $p$  est une projection  $\mathbb{F}_{2^k} \rightarrow (\mathbb{F}_2)^{k-1}$  qui consiste à extraire les  $k - 1$  premières composantes.

Montrons que cette fonction satisfait les deux conditions requises.

Tout d'abord,  $ax + b = (z, \varepsilon)$  où  $z = H_{ab}(x)$  et  $\varepsilon \in \{0, 1\}$ . Comme  $a$  est non nul, la fonction  $x \mapsto ax + b$  est inversible, et tout élément  $z$  a exactement deux antécédents correspondants aux deux valeurs possibles de  $\varepsilon$ .

Ensuite, deux éléments distincts  $x$  et  $x'$  de  $\mathbb{F}_{2^k}$  tels que  $H_{ab}(x) = H_{ab}(x')$  satisfont le système d'inconnues  $a$  et  $b$  suivant :

$$\begin{cases} ax + b &= (z, \varepsilon) \\ ax' + b &= (z, \varepsilon) \end{cases}$$

Pour  $z$  et  $b$  fixé, ce système a pour déterminant  $x - x'$  qui est non nul. La solution est unique. Un choix aléatoire du couple  $(a, b)$ , avec  $a \neq 0$  correspond à un choix  $(x', z, \varepsilon)$  avec  $x' \neq x$ .

## § 10.9 Exercices

1. Montrer que la signature RSA n'est pas sûre, y compris dans le modèle le moins exigeant pour la sécurité, où l'adversaire ne connaît que les paramètres publics, et n'a accès à aucun document signé.

2. Soit  $G$  un groupe cyclique d'ordre  $q$  engendré par  $g$ , où le problème du logarithme discret est un problème difficile. Soit  $x$  un élément de  $\mathbb{Z}/q\mathbb{Z}$  et  $y = g^x$ . On pose l'indice public  $s = (G, q, g, x)$ . Montrer que la famille de fonctions de hachage  $H_s : (\alpha, \beta) \mapsto g^\alpha y^\beta$ , pour  $\alpha, \beta \in \mathbb{Z}/q\mathbb{Z}$ , est résistante aux collisions.

3. Démontrer que si la famille de fonctions de hachage  $\mathcal{H}$  est résistante aux collisions, alors elle est universelle à sens unique.

4. Prouver le théorème 10.5 page 82 établissant la sécurité de la construction de Merkle-Damgård.

5. Rédiger la preuve détaillée du théorème 10.7 page 84.

6. Soit  $\mathcal{H} = (\text{Gen}, H)$  une famille de fonctions de hachage universelle à sens unique  $\mathcal{D}_k \rightarrow \mathcal{D}_k$ . On définit  $\mathcal{H}^*$  par :

- $\text{Gen}^*(1^k) \mapsto (s, s')$  où  $s \leftarrow \text{Gen}(1^k)$  et  $s' \leftarrow \text{Gen}(1^k)$ .
- $H_{ss'} : x \mapsto H_s \circ H_{s'}(x)$ .

Montrer que la famille  $\mathcal{H}^*$  est aussi une famille de fonctions de hachage universelle à sens unique.

7. *Bijections sans pince (claw free permutation).*

### Définition 10.9 [ pince de deux bijections ]

Soient  $f_0$  et  $f_1$  deux bijections sur un même domaine  $\mathcal{D}$ . Un triplet  $(x, y, z)$  d'éléments de  $\mathcal{D}$  est une pince de  $f_0$  et  $f_1$  si  $f_0(x) = f_1(y) = z$ .

Une famille de bijections est dite *sans pince* s'il est difficile de trouver une pince pour deux d'entre elles.

À partir d'une famille de bijections sans pince, construire une famille de fonctions de hachage résistante aux collisions.

*Indication :* considérer  $H_s : \begin{matrix} \{0, 1\}^n & \longrightarrow & \mathcal{D} \\ (x_1, \dots, x_n) & \longmapsto & f_{x_n} \circ \dots \circ f_{x_1}(s) \end{matrix}$ .

8. Soit  $N$  un module RSA. Montrer que la famille constituée des deux fonctions  $f_0 : x \mapsto x^3 \bmod N$  et  $f_1 : x \mapsto x^5 \bmod N$  n'est pas sans pince.

9. Soit  $N$  un module RSA et  $e$  un exposant RSA. Montrer que si le RSA est sûr, alors la famille constituée des deux fonctions  $f_0 : x \mapsto x^e \bmod N$  et  $f_1 : x \mapsto 2x^e \bmod N$  est sans pince.

En déduire une famille de fonctions de hachage résistante aux collisions reposant sur le problème RSA.

**10.** *Signer un symbole binaire avec une famille sans pince à trappe.*

Soit  $(f_0, f_1)$  une famille sans pince, montrez que la signature définie par  $\text{Sign}(k_{\text{pub}}b) = \sigma(b) = f_b^{|-1}(x)$ , où la clé publique  $k_{\text{pub}}$  est constituée de  $x$ ,  $f_0$  et  $f_1$  est sûre.



En composant les signatures d'un symbole binaire, il est possible de définir un schéma de signature sur plusieurs symboles binaires.



# Chiffrement à clé publique

## § 11.1 Sécurité

Le niveau de sécurité standard communément admis pour le chiffrement à clé publique est la sécurité sémantique contre des attaques adaptatives à chiffré choisi. La notion de sécurité sémantique est équivalente à celle d'indistinguabilité.

L'attaque contre l'indistinguabilité se déroule en deux temps comme un jeu entre un adversaire  $A$  et son environnement  $E$ .

1. Dans un premier temps, l'adversaire fournit un couple de messages clairs  $m_0$  et  $m_1$  à l'environnement.
2. L'environnement choisit un bit aléatoire  $b \in \{0, 1\}$  et fournit à l'adversaire le cryptogramme  $c$  du message  $c_b$ . L'objectif de l'adversaire est de fournir la valeur du bit  $b$ . Sa réponse est un bit  $b'$  et il a gagné si  $b = b'$ .

L'avantage de l'adversaire est la quantité :

$$\text{Av}(A) = \left| \Pr(b = b') - \frac{1}{2} \right|.$$

En répondant au hasard, sa probabilité de gagner vaut  $1/2$  et son avantage est nul. Si l'adversaire n'a aucun avantage, c'est qu'il ne peut pas extraire le moindre bit d'information du cryptogramme. C'est ce qui est attendu d'un système sûr de cryptographie à clé publique. L'ennemi ne doit pas savoir extraire du cryptogramme si l'information qu'il contient est l'ordre d'attaquer ou non.

Pendant les deux phases de jeu, l'adversaire peut avoir accès à un oracle de déchiffrement  $\mathcal{D}$  à qui il soumet des cryptogramme et dont il obtient en retour les messages clairs correspondants. On parle alors d'attaque à cryptogramme choisis (CCA *Chosen Ciphertext Attack*).

**CCA1.** Si l'adversaire n'a accès à cet oracle que pendant la première phase du jeu, avant de transmettre le couple de messages  $(m_0, m_1)$ , on parle d'attaque à cryptogramme choisis de niveau 1, notée CCA1.

**CCA2.** Si l'adversaire a accès à l'oracle de déchiffrement  $\mathcal{D}$  pendant les deux phases du jeu, y compris après avoir reçu le cryptogramme de la part de l'environnement, on parle alors d'attaque à cryptogramme choisis de niveau 2, notée CCA2.

Le niveau standard de sécurité admis pour considérer qu'un schéma de chiffrement est sûr, et qui est noté CCA, est l'attaque de niveau 2 : CCA = CCA2.

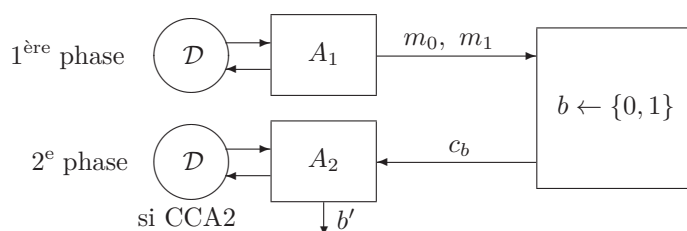


Schéma de l'attaque contre l'indistinguabilité



## § 11.2 Historique

En 1991, Naor et Yung ont proposé une méthode pour construire un schéma de chiffrement résistant aux attaques à cryptogramme choisi à partir de schéma de chiffrement qui sont seulement résistants contre les attaques à clair choisis.

Le principe de la méthode est d'utiliser deux algorithmes de chiffrement  $E_1$  et  $E_2$ , chacun agissant sur les messages respectifs  $m_1$  et  $m_2$ , le tout étant assorti d'une preuve sans divulgation et non interactive que les messages  $m_1$  et  $m_2$  sont égaux au message  $m$  à chiffrer. Le message  $m$  est chiffré deux fois :

$$E(m) = (E_1(m_1), E_2(m_2), \text{NIZK}(m_1 = m_2 = m)),$$

où NISZ désigne un schéma de preuve sans divulgation non interactive (*Non Interactive Zeroknowledge Proof*).

Pour construire un chiffré valide, il faut fournir la preuve que les deux cryptogrammes  $E_1(m_1)$  et  $E_2(m_2)$  proviennent bien du même message  $m$ . L'objectif de la preuve est de neutraliser l'accès à l'oracle de déchiffrement en rendant difficile la construction de chiffrés valides. L'oracle ne rend de réponse correcte que si le chiffré est bien construit. Et dans ce cas, il renvoie à l'adversaire le message qu'il vient de chiffrer, ce qui ne lui apporte aucune information. Dans le cas où l'adversaire tente de tromper l'oracle en fournissant deux cryptogrammes qui ne proviennent pas forcément du chiffrement d'un même message, l'oracle rendra une réponse « *chiffré invalide* », qui n'apporte aucune information non plus. C'est ainsi que ce schéma est prouvé CCA1. Mais les preuves sans divulgation sont malléables. Par exemple, après transmis son défi  $(m_1, m_2)$ , l'adversaire reçoit :

$$c_b = (E_1(m_{b,1}), E_2(m_{b,2}), \text{NIZK}(m_b = m_{b,1} = m_{b,2})),$$

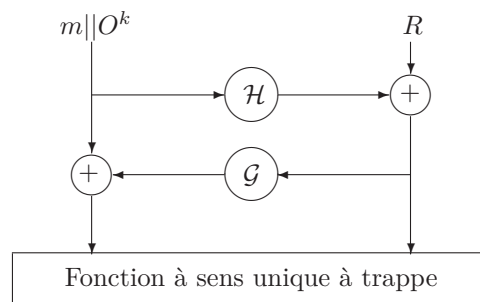
l'adversaire peut laisser identiques les deux cryptogrammes  $E_1(m_{b,1})$  et  $E_2(m_{b,2})$ , et construire une autre preuve correcte  $P'$  que  $m_b = m_{b,1} = m_{b,2}$ . En demandant à l'oracle le déchiffrement de

$$c'_b = (E_1(m_{b,1}), E_2(m_{b,2}), P'),$$

l'oracle lui donne directement  $m'_b$ , ce qui lui permet à coup sûr de rendre la bonne valeur du bit  $b$ . Il est possible de construire des schémas de chiffrement CCA2-surs avec ce principe en utilisant une preuve sans divulgation interactive, et en la rendant non-interactive. Une preuve interactive peut être transformée en preuve non interactive lorsque le prouveur et le vérifieur partagent une même séquence aléatoire (crs *Common Random String*) qui simule l'interaction dans la preuve. Toutefois, les constructions obtenues sont très peu efficaces, nécessitent des permutations à sens unique avec trappe améliorées (ETP *Enhanced Trapdoor Permutation*).

Pour atteindre des schémas CCA2-surs efficaces, BELLARE et ROGAWAY, ont eu recours à la notion d'oracle aléatoire. Il s'agit d'une vision idéale d'une fonction de hachage. Cela correspond à une fonction aléatoire, dont les valeurs seraient comme tirées aléatoirement, mais bien sûr, s'agissant d'une fonction, la même valeur est retournée pour le même paramètre.

Cela a aboutit en 1994 au schéma OAEP (*Optimal Asymmetric Encryption Padding*) , où une quantité aléatoire est ajoutée au message, puis brouillée par un schéma de FEISTEL à deux tours comme schématisé ci-après :



Dans ce schéma, les fonctions  $\mathcal{F}$  et  $\mathcal{G}$  sont supposées être des oracles aléatoires. La partie gauche du schéma de FEISTEL est constituée du message  $m$ , complété par des zéros pour avoir la taille convenable, et la partie droite est constituée d'une quantité aléatoire  $R$  de la taille convenable.

Lorsque la fonction à sens unique avec trappe est le RSA, ce schéma est standardisé pour la sécurité de l'Internet dans les normes ANSI X9.44, IEEE P1363 et SET.

Ce schéma a été présenté comme CCA2-sûr, mais sans donner de preuve jusqu'en 2001, où Victor SHoup a présenté une attaque contre ce schéma avec une fonction à sens unique avec trappe particulière. Une semaine après, il apporte la preuve de la sécurité CCA2, à condition que la fonction de chiffrement utilisée soit à sens unique partielle, c'est-à-dire qu'on ne puisse pas calculer la partie gauche de l'entrée à partir de la sortie. E. FUJISAKI, T. OKAMOTO, D. Pointcheval et J. Stern ont finalement prouvé en 2001 que la fonction RSA satisfait cette propriété d'être à sens unique partielle, pouvant finalement la sécurité CCA2 du standard OAEP-RSA. En 2003, D.H. PHAN et D. POINTCHEVAL ont prouvé qu'avec un schéma de FEISTEL à trois tours au lieu de deux, d'une part la fonction de chiffrement n'a plus besoin d'avoir la propriété d'être à sens unique partielle, toute fonction à sens unique avec trappe convient, mais la redondance  $R$  devient inutile.

Toutefois, cette sécurité n'est assurée que si les fonction  $\mathcal{G}$  et  $\mathcal{H}$  utilisées sont des oracles aléatoires, ce qui est une hypothèse considérée comme trop forte. L'objet de la suite est de présenter une solution au problème du chiffrement à clé publique qui ne repose pas sur le modèle de l'oracle aléatoire. Si les fonctions requises pour réaliser le chiffrement ont seulement des propriétés de résistance aux collisions ou sont à sens unique, on parle alors de *modèle standard*.

## § 11.3 CPA-sécurité sémantique du schéma ElGamal

Rappelons la définition du chiffrement ELGAMAL. Soit un groupe  $G$ , cyclique d'ordre  $q$ , où  $q$  est un nombre premier. Soit  $g$  un générateur de  $G$ , de telle sorte que  $G = \langle g \rangle$ .

- La clé privée est un élément  $x$ , choisi aléatoirement dans  $\mathbb{Z}/q\mathbb{Z}$ .
- La clé publique est l'élément  $p = g^x$ .
- L'espace des messages est le groupe  $G$ . Le chiffrement d'un message  $m \in G$  procède ainsi :
  - Choisir un élément  $r$  aléatoire dans  $\mathbb{Z}/q\mathbb{Z}$ .
  - Le cryptogramme est constitué du couple  $(c_1, c_2)$ , où  $c_1 = g^r$  et  $c_2 = p^r \times m$ .
- à partir du cryptogramme  $(c_1, c_2)$ , le message  $m$  se retrouve par  $m = \frac{c_2}{c_1^x}$ . En effet,  $\frac{c_2}{c_1^x} = \frac{p^r \times m}{g^{rx}} = \frac{g^{xr} \times m}{g^{rx}} = m$ .

Sous l'hypothèse de la difficulté de ce qu'on appelle le *problème* DIFFIE-HELLMANN *décisionnel* (DDH *Decision Diffie-Hellman*), ce schéma de chiffrement est sémantiquement sûr dans une attaque à clairs choisis.

L'hypothèse DDH signifie que le problème de décision DIFFIE-HELLMANN est difficile, c'est-à-dire qu'il est difficile de distinguer les deux quadruplets :

$$(g, g^x, g^y, g^{xy}) \quad \text{et} \quad (g, g^x, g^y, s),$$

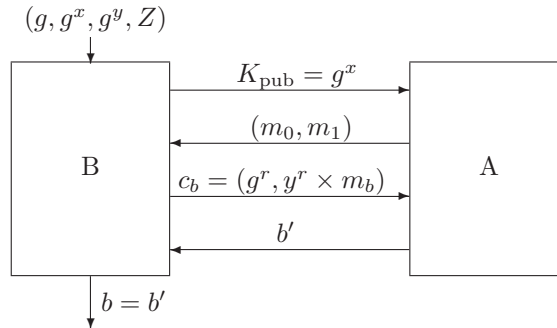
où  $z$  est un élément aléatoire de  $G$ . Cela signifie qu'il est difficile de décider si une donnée  $z$  est ou non égal à  $g^{xy}$  lorsqu'on connaît  $g$ ,  $g^x$  et  $g^y$ . Cette hypothèse équivaut à l'indistinguabilité calculatoire des deux quadruplets :

$$(g_1, g_2, g_1^x g_2^x) \quad \text{et} \quad (g_1, g_2, g_1^x g_2^{x'}),$$

où  $g_1$  et  $g_2$  sont des éléments du groupe  $G$  et les éléments  $x$  et  $x'$  de  $\mathbb{Z}/q\mathbb{Z}$  sont distincts.

La suite de ce paragraphe décrit le principe de la preuve. On construit un algorithme  $B$ , qui résout le problème DDH en utilisant un adversaire  $A$  contre le schéma de chiffrement ELGAMAL.

L'algorithme  $B$  a pour entrée le quadruplet  $(g, g^x, g^y, Z)$ , et il doit décider si oui ou non on a  $Z = g^{xy}$ . Pour cela, il dispose d'un adversaire  $A$  contre l'indistinguabilité du schéma de chiffrement ELGAMAL. L'algorithme  $B$  va exploiter l'adversaire  $A$  en simulant l'environnement qui lui permet de travailler.



La clé publique que  $B$  fournit à l'adversaire  $A$  est  $y = g^x$ . Si la composante  $Z$  vient de  $Z = g^{xy}$ , alors la situation est parfaite pour l'adversaire  $A$  qui sait répondre avec un avantage non négligeable. À l'opposé, si  $Z$  est aléatoire, le chiffré est parfaitement aléatoire et donc  $A$  n'a aucun avantage. Il ne peut répondre correctement qu'avec une probabilité égale à  $1/2$ . Il en résulte que l'avantage de cet algorithme  $B$  pour résoudre le problème DDH est non négligeable.

On n'a pas de preuve que le schéma de chiffrement ELGAMAL est CCA1-sûr, mais on sait qu'il n'est pas CCA2-sûr, car il est homomorphique. Si on sait déchiffrer le cryptogramme  $(R, M)$ , alors on sait déchiffrer aussi le cryptogramme  $R, k \times M$  (voir exercice 2).

## § 11.4 Chiffrement CCA-sûr dans le modèle standard

### 11.4.1. Principe général

Le schéma de Cramer-Shoup a été conçu comme une évolution du chiffrement ELGAMAL de manière à pouvoir établir la preuve de sa sécurité dans une attaque cryptogrammes choisis CCA2 dans le modèle standard, c'est-à-dire sans avoir recours à des oracles aléatoires. En reprenant la preuve du paragraphe précédent, l'objectif, pour l'algorithme  $B$  est de pouvoir résoudre le problème DDH en utilisant un adversaire qui peut avoir accès à un oracle de déchiffrement. Il va simuler la possession de la clé privée de manière à pouvoir répondre aux requêtes de déchiffrement formulées par l'adversaire  $A$  pendant les deux phases de son travail : avant et après la production du défi  $(m_0, m_1)$ .

Selon la méthodologie de Naor et Yung, le cryptogramme ressemblera à quelque chose comme :

$$c = (g_1^r, g_2^r, y^r \times m, \dots),$$

avec une preuve que ce chiffré est valide, c'est-à-dire que les deux premières composantes sont construites à partir du même aléa  $r$ .



Dans la méthodologie de Naor et Yung, la condition de preuve *non interactive sans divulgation* est trop forte. Avec cette condition, tout le monde peut effectuer la vérification. Cela n'est pas nécessaire. On a seulement besoin que le destinataire puisse effectuer cette vérification. Or le destinataire dispose de la clé privée. Il est possible d'affaiblir la condition, et d'avoir un modèle de preuve uniquement pour un vérifieur désigné (*designated verifier*). Pour ce modèle, il est possible de dégager une méthodologie générale.

### 11.4.2. Preuve avec vérifieur désigné

Le problème DDH peut s'exprimer en terme de langage. Le langage DDH est défini par :

$$\mathcal{L}_{\text{DDH}} = \{g_1, g_2, g_1^x, g_2^x \mid x \in \mathbb{Z}/q\mathbb{Z}\}.$$

Il est formé des mots qui sont des quadruplets bien formés, c'est-à-dire dont les deux dernières composantes sont construites avec le même exposant  $x$  de  $\mathbb{Z}/q\mathbb{Z}$ . Ce langage est un sous-langage du langage plus général :

$$\mathcal{L} = \{g_1, g_2, g_1^x, g_2^y \mid x, y \in \mathbb{Z}/q\mathbb{Z}\}.$$

Dans  $\mathcal{L}$ , les deux dernières composantes n'ont pas forcément le même exposant. Le problème décisionnel DIFFIE-HELLMANN consiste à décider si un élément de  $\mathcal{L}$  appartient ou non au langage  $\mathcal{L}_{\text{DDH}}$ .

Une preuve qu'un quadruplet  $(g_1, g_2, X, Y)$  appartient au langage  $\mathcal{L}_{\text{DDH}}$  est l'élément  $x$ . Cet élément s'appelle un *témoin*. Il permet de vérifier que  $X = g_1^x$  et  $Y = g_2^x$ .

L'élément  $x$  est un témoin qui permet à tous de vérifier l'appartenance d'un mot au langage  $\mathcal{L}_{\text{DDH}}$ . Pour rendre la preuve réservée à un vérifieur désigné, il suffit de chiffrer ce témoin pour que sa valeur ne soit accessible qu'au détenteur de la clé privée correspondante.

En remplaçant la preuve non interactive par une preuve pour un vérifieur désigné, l'adversaire ne peut pas jouer avec un cryptogramme qui n'appartient pas au langage  $\mathcal{L}$ . Sur une requête de déchiffrement, soit le cryptogramme n'est pas valide et il n'apprend rien, soit le cryptogramme est bien formé, mais alors c'est qu'il a été chiffré correctement et l'adversaire connaît la réponse, il n'apprend rien non plus.

### 11.4.3. Système universel de preuve

#### Définition 11.1 [ système de preuve ]

Un système de preuve (HPS Hash Proof System) pour un langage  $\mathcal{L}$  consiste en trois algorithmes :

1.  $\text{Gen}(1^k)$  est un algorithme de génération, où  $k$  est un paramètre de sécurité qui produit une chaîne aléatoire  $\text{crs}$  (Common Random String) et une clé de trappe  $\text{tk}$  (Trapdoor Key).
2.  $\text{P}(y, x, \text{crs})$  est le prouveur. Il produit une donnée  $\pi$  à partir du témoin.
3.  $\text{V}(y, \text{tk})$  est le vérifieur. Il produit une donnée  $\pi'$  à partir de la clé de trappe  $\text{tk}$ .



Le prouveur utilise le témoin  $x$  pour établir la preuve, alors que le vérifieur utilise la clé de trappe pour le même usage.

Les propriétés requises pour un système de preuve sont :

1. La correction : si le mot  $y$  appartient au langage  $\mathcal{L}$ , alors le prouveur et le vérifieur produisent la même donnée avec certitude, c'est-à-dire  $\Pr[\pi = \pi'] = 1$ .
2. La  $t$ -universalité : Si  $t$  mots  $y_1, \dots, y_t$  n'appartiennent pas au langage  $\mathcal{L}$ , alors la donnée :

$$(\text{crs}, \text{V}(y_1, \text{tk}_1), \dots, \text{V}(y_t, \text{tk}_t)),$$

est statistiquement indistinguishable d'une donnée :

$$(\text{crs}, r_1, \dots, r_t)$$

où les  $r_i$  sont des aléas véritables.

Ainsi, le prouveur  $\text{P}$  pourra produire une preuve  $\pi$  qu'un mot  $y$  appartient à  $\mathcal{L}$  que seul le vérifieur  $\text{V}$  pourra vérifier avec sa clé privée  $\text{tk}$ . Si l'élément  $y$  n'appartient pas à  $\mathcal{L}$  la vérification échouera.

Donnons maintenant deux exemples de schémas de preuve 1-universel et 2-universel.

### Un schéma de preuve 1-universel reposant sur DDH

Considérons le schéma de preuve suivant : pour un paramètre de sécurité  $k$ , on a un nombre  $q$  premier dont l'écriture binaire comprend  $k$  chiffres. Soit  $G$  un groupe cyclique d'ordre  $q$  et soient  $g_1$  et  $g_2$  deux éléments quelconques de  $G^*$ . Comme  $q$  est premier, ce sont des générateurs du groupe  $G$ . Posons  $g_2 = g_1^\omega$ .

La fonction de génération  $\text{Gen}(-1^k)$  produit la clé de trappe  $\text{tk} = (x_1, x_2)$  où  $x_1$  et  $x_2$  sont des éléments aléatoires de  $\mathbb{Z}/q\mathbb{Z}$ , ainsi que la chaîne aléatoire  $c = g_1^{x_1} g_2^{x_2}$ .

Le langage  $\mathcal{L}_{\text{DDH}}$  est constituée des mots de la forme  $(g_1^r, g_2^r)$ , pour  $r \in \mathbb{Z}/q\mathbb{Z}$ . La valeur de  $r$  est le témoin d'appartenance à  $\mathcal{L}_{\text{DDH}}$ . Ce langage est un sous-ensemble du langage  $\mathcal{L}$  constitué de tous les couples de  $G \times G$ . Pour un élément  $y = (u_1, u_2)$  de  $\mathcal{L}$ , où  $u_1 = g_1^r$  et  $u_2 = g_2^r$ , le témoin  $r$  et la chaîne aléatoire  $c$ , le prouveur  $P$  renvoie la donnée :

$$P(y, r, c) = c^r.$$

à partir de la clé de trappe  $\text{tk}$  et de l'élément  $y = (u_1, u_2)$  de  $\mathcal{L}$ , le vérifieur  $V$  calcule la donnée :

$$V(y, \text{tk}) = u_1^{x_1} u_2^{x_2}$$

Montrons que ce schéma de preuve satisfait les propriétés requises. Ce schéma est correct, car pour un élément du langage  $\mathcal{L}_{\text{DDH}}$ , c'est-à-dire si  $y = (u_1, u_2) = (g_1^r, g_2^r) \in \mathcal{L}_{\text{DDH}}$ , alors on a

$$V(y, \text{tk}) = u_1^{x_1} u_2^{x_2} = (g_1^{x_1} g_2^{x_2})^r = c^r = P(y, r, c).$$

Pour prouver la 1-universalité, il faut montrer que si le couple  $(u_1, u_2)$  n'appartient pas à  $\mathcal{L}_{\text{DDH}}$ , alors, si  $x_1$  et  $x_2$  sont choisis aléatoirement, le couple  $(c, u_1^{x_1} u_2^{x_2})$  n'est pas distinguable d'un couple  $(c, r)$  où  $r$  est un véritable aléa. Comme  $(u_1, u_2)$  n'appartient pas à  $\mathcal{L}_{\text{DDH}}$ , il est le résultat de  $u_1 = g_1^{r_1 x_1}$  et  $u_2 = g_2^{r_2 x_2}$ , avec  $r_1 \neq r_2$ . Pour cela, montrons que la fonction :

$$f : (x_1, x_2) \mapsto (g_1^{x_1} g_2^{x_2}, g_1^{r_1 x_1} g_2^{r_2 x_2})$$

est injective. En passant aux exposants, cela revient à montrer que la fonction :

$$(x_1, x_2) \mapsto (x_1 + \omega x_2, r_1 x_1 + \omega r_2 x_2),$$

ce qui est le cas, car cette fonction est linéaire de matrice  $\begin{pmatrix} 1 & \omega \\ r_1 & \omega r_2 \end{pmatrix}$ , dont le déterminant vaut  $\omega(r_2 - r_1)$  et n'est donc pas nul. La fonction  $f$  étant injective, si  $x_1$  et  $x_2$  sont aléatoires, alors la valeur  $f(x_1, x_2)$  l'est aussi, ce qui achève de démontrer que le schéma de preuve présenté dans ce paragraphe a la propriété de 1-universalité.

### Un schéma de preuve 2-universel

La construction précédente s'applique aussi pour construire un schéma de preuve 2-universel, en doublant les données. Il faut aussi disposer d'une fonction  $h : G \times G \rightarrow \mathbb{Z}/q\mathbb{Z}$  résistante aux collisions. Considérons le schéma de preuve suivant :

La fonction  $\text{Gen}(1^k)$  génère quatre éléments aléatoires  $x_1, x_2, y_1$  et  $y_2$  de  $\mathbb{Z}/q\mathbb{Z}$ . à partir de ces quatre éléments, la chaîne aléatoire est  $\text{crs} = (c, d)$  avec  $c = g_1^{x_1} g_2^{x_2}$  et  $d = g_1^{y_1} g_2^{y_2}$ . La clé de trappe  $\text{tl}$  est  $\text{tk} = (x_1 x_2, y_1 y_2)$ .

Le prouveur  $P$ , à partir d'un élément  $(u_1, u_2) = (g_1^r, g_2^r)$  du langage  $\mathcal{L}_{\text{DDH}}$ , du témoin  $r$  et de la chaîne aléatoire  $\text{crs}$ , calcule la donnée donnée par :

$$P(u_1, u_2, r, \text{crs}) = c^r d^{r^\alpha},$$

où  $\alpha = h(u_1, u_2)$  est l'image par la fonction de hachage  $h$  de l'élément  $(u_1, u_2)$  du langage.

La fonction de vérification est donnée par :

$$V((u_1, u_2, \text{tk}) = (u_1^{x_1} u_2^{x_2})(u_1^{y_1} u_2^{y_2})^\alpha.$$

Vérifier que ce schéma de preuve est correct est immédiat. Il suffit de montrer que si  $(u_1, u_2) \in \mathcal{L}_{\text{DDH}}$  alors  $P(u_1, u_2, r, \text{crs}) = V(u_1, u_2, \text{tk})$ , ce qui découle directement des expressions de  $P$  et  $V$ .

Pour montrer la 2-universalité, considérons deux mots  $y$  et  $y'$  qui appartiennent au langage  $\mathcal{L}$  sans appartenir au langage  $\mathcal{L}_{\text{DDH}}$ , c'est-à-dire qu'on a  $y = (u_1, u_2)$  avec  $u_1 = g_1^{r_1}$ ,  $u_2 = g_2^{r_2}$  et  $r_1 \neq r_2$ . De même, on a  $y' = (u'_1, u'_2)$  avec  $u'_1 = g_1^{r'_1}$ ,  $u'_2 = g_2^{r'_2}$  et  $r'_1 \neq r'_2$ . Il faut alors montrer que le triplet  $(\text{crs}, V(y, \text{tk}), V(y', \text{tk}))$  est indistinguable d'un triplet  $(\text{crs}, r, r')$  où  $r$  et  $r'$  sont des données aléatoires. Pour montrer cela, comme au paragraphe précédent, montrons que la fonction :

$$f; (x_1, x_2, y_1, y_2) \mapsto (g_1^{x_1} g_2^{x_2}, g_1^{y_1} g_2^{y_2}, g_1^{r_1 x_1 + \alpha r_1 y_1} g_2^{r_2 x_2 + \alpha r_2 y_2}, g_1^{r'_1 x_1 + \alpha' r'_1 y_1} g_2^{r'_2 x_2 + \alpha' r'_2 y_2}),$$

est injective, où  $\alpha = h(u_1, u_2)$  et  $\alpha' = h(u'_1, u'_2)$ . En passant aux exposants et en remplaçant  $g_2$  par  $g_1^\omega$ , montrer que  $f$  est injective revient à montrer que la matrice :

$$M = \begin{pmatrix} 1 & \omega & 0 & 0 \\ 0 & 0 & 1 & \omega \\ r_1 & \omega r_2 & \alpha r_1 & \alpha \omega r_2 \\ r'_1 & \omega r'_2 & \alpha' r'_1 & \alpha' \omega r'_2 \end{pmatrix}$$

est inversible. Le déterminant de la matrice  $M$  vaut :

$$\det(M) = \omega^2(r_2 - r_1)(r'_2 - r'_1)(\alpha - \alpha').$$

Comme on a  $r_1 \neq r_2$  et  $r'_1 \neq r'_2$ , le déterminant est nul si et seulement si  $\alpha = \alpha'$ , c'est-à-dire si on a une collision sur la fonction de hachage  $h$ .

#### 11.4.4. Un chiffrement CCA-sûr

Ce paragraphe présente la construction d'un chiffrement CCA-sûr. Le problème difficile sous-jacent est celui de distinguer l'appartenance ou non d'un mot  $y$  à un langage  $\mathcal{L}$ . On suppose également qu'on a les systèmes de preuves suivants avec les propriétés indiquées :

- le système  $(\text{Gen}, P, V)$  a la propriété de 1-universalité pour le langage  $\mathcal{L}$  ;
- le système  $(\text{Gen}', P', V')$  a la propriété de 2-universalité pour le langage  $\mathcal{L}' = L || \{0, 1\}^n$ .

Les mots du langage  $\mathcal{L}'$  sont la concaténation des mots du langage  $\mathcal{L}$  avec n'importe quelle étiquette constituée de  $n$  symboles binaires.

Définissons le système de chiffrement à clé publique  $(\text{Gen}_E, \text{Enc}, \text{Dec})$  :

- Génération des clés  $\text{Gen}_E(1^k)$  pour le paramètre de sécurité  $k$  :
  - Soit  $(\text{crs}, \text{tk}) \leftarrow \text{Gen}(1^k)$ .
  - Soit  $(\text{crs}', \text{tk}') \leftarrow \text{Gen}'(1^k)$ .
  - La clé publique  $K_{\text{pub}}$  est  $(\text{crs}, \text{crs}')$ .
  - La clé privée  $K_{\text{priv}}$  est  $(\text{tk}, \text{tk}')$ .
- Chiffrement  $\text{Enc}_{K_{\text{pub}}}(m)$  :
  - Choisir un élément  $y$  dans le langage  $\mathcal{L}$  avec un témoin  $x$ .
  - La clé de session est  $\pi = P(t, x, \text{crs})$ . Le message  $m$  est masqué avec cette clé de session comme un masque jetable :  $c = m \oplus \pi$ .
  - Soit  $\pi' = P(y || c, x, \text{crs}')$ .
  - Le cryptogramme est  $C = (y, c, \pi')$ .
- Déchiffrement  $\text{Dec}_{K_{\text{priv}}}(y, c, \pi')$  :
  - Calculer  $\tilde{\pi} = V(y, \text{tk})$
  - Calculer  $\tilde{\pi}' = V'(y || c, \text{tk}')$ .
  - Vérifier si  $\pi = \pi'$ . Si ce n'est pas le cas, signifier l'échec du déchiffrement. Si c'est le cas, alors retourner  $m' = \tilde{\pi} \oplus c$ .

*Preuve.* La preuve de la sécurité CCA de ce système de chiffrement considère des jeux.

Le **jeu 0** est le jeu réel. L'adversaire  $A$  peut accéder à l'oracle de déchiffrement, et celui-ci fournit les bonnes réponses conformément à la définition du schéma de chiffrement :

L'adversaire fournit un défi  $(m_0, m_1)$ . L'environnement choisit un bit  $b$  aléatoire dans  $\{0, 1\}$ , un élément  $y$  du langage  $\mathcal{L}$  avec un témoin  $x$ , calcule les données  $P(y, x, \text{crs}) = \pi$  et  $P'(y||c, x, \text{crs}') = \pi'$ . Il fournit le cryptogramme  $C = (y, c, \pi')$ , où  $c = m \oplus \pi$ . L'objectif de l'adversaire  $A$  est de trouver la valeur du bit  $b$ .

Le **jeu 1** est semblable au jeu 0, mais au lieu de calculer les données  $\pi$  et  $\pi'$  par les prouveurs  $P$  et  $P'$ , ces données vont être créées par le vérifieur avec la clé de trappe  $\text{tk}$ . Le calcul des données  $\pi$  et  $\pi'$  se fait ainsi :  $V(y, \text{tk}) = \pi$  et  $V'(y||c, \text{tk}') = \pi'$ .

Les jeux 0 et 1 sont identiques, car comme  $y$  appartient au langage  $\mathcal{L}$ , les valeurs  $\pi$  et  $\pi'$  calculées au cours de ces deux jeux sont les mêmes. La différence réside dans le fait que, dans le jeu 1, le simulateur de l'environnement dispose de la clé privée et n'a plus besoin du témoin de  $y$ .

Dans le **jeu 2**, le cryptogramme est calculé avec un élément  $y$  qui n'appartient pas au langage  $\mathcal{L}$ . Les défis des jeux 1 et 2 sont indistinguables, à cause de la propriété d'indistinguabilité d'appartenance au langage  $\mathcal{L}$ . La question est de savoir si les requêtes ont plus de chance d'être invalides dans le jeu 1 ou dans le jeu 2.

Notons  $\text{DecInv}$  l'évènement « le cryptogramme est invalide mais passe le test de validité ». C'est le cas si l'adversaire  $A$  pose un cryptogramme invalide avec  $y \notin \mathcal{L}$ , mais qui passe la vérification, car  $\tilde{\pi}' = \pi'$ .

Dans le jeu 1, comme la valeur de  $V(y, \text{tk})$  est aléatoire, la probabilité de cet évènement est négligeable, égale à l'inverse du cardinal de l'ensemble d'arrivée.

Dans le jeu 2, c'est différent, car l'adversaire reçoit  $V(y, \text{tk})$  pour  $y \notin \mathcal{L}$ . C'est la 2-universalité qui rend la probabilité de l'évènement  $\text{DecInv}$  négligeable.

Si l'adversaire pose à l'oracle un cryptogramme  $C^* = (y^*, c^*, \pi'^*)$  avec  $y^* = y$  et  $c^* = c$ , alors nécessairement  $\pi'^* \neq \pi'$ , car la question posée à l'oracle doit être différente du défi, et dans ce cas, la vérification rejette le cryptogramme.

Un cryptogramme invalide qui passe la vérification doit nécessairement vérifier  $(y^*, c^*) \neq (y, c)$ . Si le cryptogramme  $C^*$  passe la vérification, c'est que  $\pi'^* = V'(y^*||c^*, \text{tk})$ , mais comme les éléments  $y$  et  $y^*$  n'appartiennent pas au langage  $\mathcal{L}$ , en raison de la 2-universalité, le couple  $((y, c, \pi'), (y^*, c^*, \pi'^*))$  est constitué de composantes aléatoires, et le test passe avec une probabilité négligeable.

Si  $A$  pose des cryptogrammes valides, cela se passe comme dans le jeu 1.

Par conséquent, l'avantage de l'adversaire  $A$  entre le jeu 1 et le jeu 2 change d'une quantité négligeable.

Dans le **jeu 3**, on remplace la composante  $c = \pi \oplus m_b$  dans le défi par un aléa. Statistiquement, le jeu 3 est comme le jeu 1, et dans le jeu 3, l'avantage de l'adversaire est nul.

□

#### 11.4.5. Le chiffrement Cramer-Shoup

Le chiffrement de Cramer-Shoup est la construction de la section précédente avec les schémas de preuve 1-universel et 2-universel du paragraphe 11.4.3. Ce système de chiffrement opère dans un groupe  $G$ , cyclique dont l'ordre  $q$  est un nombre premier. Soient  $g_1$  et  $g_2$  deux générateurs de ce groupe. Soit également une fonction  $h : G \times G \rightarrow \mathbb{Z}/q\mathbb{Z}$  supposée sans collision.

La clé privée est la clé de trappe, constituée de six éléments aléatoires  $x_1, x_2, y_1, y_2, z_1$  et  $z_2$  de  $\mathbb{Z}/q\mathbb{Z}$ .

La clé publique correspondante est le triplet  $(c, d, h)$  de  $G^3$ , où  $c = g_1^{x_1} g_2^{x_2}$ ,  $d = g_1^{y_1} g_2^{y_2}$  ry  $h = g_1^{z_1} g_2^{z_2}$ .



Dans la clé privée, les quatre éléments  $x_1, x_2, y_1$  et  $y_2$  sont la clé de trappe d'un système de preuve, et le couple  $(c, d)$  est la chaîne aléatoire commune.

Pour chiffrer un message  $m$  de  $G$ , on choisit un élément aléatoire  $r$  dans  $\mathbb{Z}/q\mathbb{Z}$ , puis on calcule  $u_1 = g_1^r$  et  $u_2 = g_2^r$ , puis  $e = h^r \times m$ . On pose  $\alpha = h(u_1 u_2, e)$  et  $v = c^r d^{r\alpha}$ . Le cryptogramme est constitué du quadruplet  $C = (u_1, u_2, e, v)$ .



La donnée  $e$  est l'étiquette définissant les mots du langage  $\mathcal{L}'$

Pour déchiffrer ce cryptogramme, on vérifie tout d'abord si  $v$  est ou non égal à  $u_1^{x_1} u_2^{x_2} (u_1^{y_1} u_2^{y_2})^\alpha$ , avec  $\alpha = h(u_1 u_2, e)$ . Si non, on déclare l'échec du déchiffrement, et si oui, on retourne le clair  $e/u_1^{z_1} u_2^{z_2}$ .

## § 11.5 Exercices

1. Écrire en détail la preuve de la sécurité CPA du schéma de chiffrement ELGAMAL sous l'hypothèse de la difficulté du problème DIFFIE-HELLMANN décisionnel.
2. Trouver une attaque contre la sécurité CCA2 du schéma de chiffrement ELGAMAL.  
*Indication :* Si  $c = (R, M)$  demander le déchiffrement de  $(R, 2M)$ .
3. Démontrer que le système de chiffrement de CRAMER-SHOUP est correct, c'est-à-dire que la procédure de déchiffrement retrouve bien le message clair qui a été chiffré, et qu'il est CCA-sûr sous l'hypothèse de la difficulté du problème de décision DIFFIE-HELLMANN.
4. Le système de CRAMER-SHOUP reste-t-il sûr si on détache la composante  $\tilde{v} = (c^r, d^{r^\alpha})$  dans le cryptogramme au lieu de ne transmettre que le produit  $v = c^r d^{r^\alpha}$ .



# Corrigé des exercices

## 1.1.

1.2. a) Pour la fonction puissance modulo un nombre premier :

- L'ensemble d'indices est l'ensemble des couples  $(p, g)$ , où  $p$  est un nombre premier, et  $g$  est un générateur du groupe multiplicatif  $\mathbb{Z}/p\mathbb{Z}^*$ . Pour pouvoir trouver efficacement un générateur du groupe multiplicatif  $\mathbb{Z}/p\mathbb{Z}$ , il faut connaître la factorisation de  $p - 1$ .
- Pour un indice  $i = (p, g)$ , le domaine  $D_i$  est l'ensemble des entiers  $\{1, \dots, p - 1\}$ .
- L'algorithme  $S_1$  génère un nombre premier aléatoire  $p$  dont la factorisation de  $p - 1$  est connue, ainsi qu'un générateur aléatoire  $g$ . Pour tester qu'un élément  $g$  est générateur du groupe multiplicatif  $\mathbb{Z}/p\mathbb{Z}^*$ , on vérifie que  $r^r \neq 1$  pour tout  $r = (p - 1)/r$ , où  $r$  est un diviseur premier de  $p - 1$ .
- L'algorithme  $S_2$  génère un entier aléatoire dans l'ensemble  $\{1, \dots, p - 1\}$ .
- Pour  $i = (p, g)$  et  $n \in \{1, \dots, p - 1\}$ , la fonction  $f_i$  est  $n \mapsto g^n \bmod p$ .

b) Pour la fonction RSA :

- L'ensemble d'indices est  $I$  est l'ensemble des couples  $(n, e)$ , où  $n$  est un entier égal au produit de deux nombres premiers de même taille, et où  $e$  est un exposant public valide pour  $n$ , c'est-à-dire un entier qui est premier avec  $\varphi(n)$ .
- Le domaine des fonctions  $D_i$  est l'ensemble  $\mathbb{Z}/n\mathbb{Z}$ .
- L'algorithme  $S_1$  génère un couple  $(n, e)$ . Pour cela il génère un couple de nombre premiers aléatoires, en tirant au hasard un entier quelconque, puis en testant sa primalité. Cette façon de faire est de complexité polynomiale à l'aide par exemple du test de primalité de MILLER-RABIN, ainsi qu'en raison de la densité des nombres premiers donnée par le théorème des nombres premiers.
- L'algorithme  $S_2$  génère un élément aléatoire de  $\mathbb{Z}/n\mathbb{Z}$ .
- Pour  $i = (n, e) \in I$ , la fonction RSA  $f_i$  est  $x \mapsto x^e \bmod n$ .



Il n'est pas nécessaire de restreindre le domaine seulement aux éléments inversibles modulo  $n$ . En raison du théorème chinois, les formules RSA restent valables pour tout élément de  $\mathbb{Z}/n\mathbb{Z}$ . Certes, la donnée d'un élément non inversible non nul de  $\mathbb{Z}/n\mathbb{Z}$  permet de factoriser  $n$ , mais trouver un tel élément est un problème difficile et la situation d'avoir à chiffrer un tel élément est hautement improbable.

c) Pour la fonction carré modulo le produit de deux nombres premiers :

1.3. Rappelons que si  $\mathcal{P} \neq \mathcal{NP}$ , alors un problème  $\mathcal{NP}$ -complet est un problème difficile dans le pire des cas. L'idée pour cet exercice est de construire une fonction à partir d'un problème  $\mathcal{NP}$ -complet. Elle sera difficile à inverser dans le pire des cas, mais pourra être facile à inverser en moyenne.

Prenons le problème 3COL du coloriage d'un graphe avec trois couleurs.

Un graphe  $\mathcal{G}$  de  $n$  sommet est par définition une matrice  $n \times n$  dont les coefficients appartiennent à l'ensemble  $\{0, 1\}$ . Les coefficients égaux à 1 représentent une arête entre deux sommets.

Un coloriage avec trois couleurs du graphe  $\mathcal{G}(g_{ij})_{i,j \in \{1, \dots, n\}}$  est une application  $c$  de l'ensemble  $\{1, \dots, n\}$  des sommets vers l'ensemble  $\{1, 2, 3\}$  des couleurs, qui à chaque sommet associe une couleur, et tel que deux sommets adjacents ne soient pas coloriés avec la même couleur, c'est-à-dire, pour tout couple  $(i, j)$  de sommets adjacents, c'est-à-dire tels que  $g_{ij} = 1$ , on a  $c(i) \neq c(j)$ .

Rappelons ces résultats :

- Si on sait qu'un graphe est coloriable avec trois couleurs, il est toujours difficile de trouver un coloriage.
- Si on tire au hasard un graphe  $\mathcal{G}$  et une application  $c$ , il est hautement improbable que  $c$  soit un coloriage pour  $\mathcal{G}$ .

Notons  $\mathcal{M}_n$  l'ensemble des graphes de  $n$  sommets. Construisons donc la fonction suivante :

$$f : \begin{array}{ll} \mathcal{M}_n \times \{1, 2, 3\}^n & \longrightarrow \mathcal{M}_n \times \{0, 1\} \\ (\mathcal{G}, c) & \longmapsto \begin{cases} (\mathcal{G}, 0) & \text{si } c \text{ colorie } \mathcal{G} \\ (\mathcal{G}, 1) & \text{sinon} \end{cases} \end{array}$$

Pour un couple  $(\mathcal{G}, c)$  aléatoire, très probablement,  $c$  ne colorie pas  $\mathcal{G}$ , et donc la valeur par  $f$  vaut  $(\mathcal{G}, 1)$ . Dans ce cas, en renvoyant une valeur  $c'$  elle aussi aléatoire, on aura trouvé un antécédent par  $f$ .

Le seul cas où trouver l'antécédent sera un problème difficile sera lorsqu'il faudra chercher un antécédent de  $(\mathcal{G}, 0)$ . Il suffira de répondre au hasard. Ce sera sûrement un échec, mais cela survient très rarement.

La fonction  $f$  n'est pas à sens unique en moyenne, mais est difficile à inverser dans le pire des cas.



Il est crucial comprendre que les défis qu'un algorithme doit chercher à inverser sont des images d'éléments aléatoires de l'ensemble de départ. S'il s'agissait d'éléments aléatoires de l'ensemble d'arrivée, la probabilité de succès serait toujours négligeable.

**1.4.** Soit  $f$  une fonction à sens unique. Construisons à partir de  $f$  une fonction  $F$  à longueur régulière. Pour cela, on complète la valeur  $f(x)$  par des zéros jusqu'à atteindre une longueur constante pour toutes les entrées de taille  $n$ . Comme  $f$  est à sens unique, elle est calculable efficacement et pour toute entrée de taille  $n$ , la taille de  $f(x)$  est bornée par un polynôme en  $n$ . Soit  $P(n)$  ce polynôme connu.

La fonction  $F : x \mapsto (f(x), 0^k)$ , où  $k = P(n) - |f(x)|$ , est à longueur régulière par construction. La fonction  $F$  est aussi à sens unique, car si elle ne l'était pas, cela conduirait directement à un algorithme pour inverser  $f$ .

**1.5.** Soit  $g$  une fonction à sens unique. À partir de  $g$  construisons une fonction  $f$  à sens unique qui conserve les longueurs. Comme  $g$  est calculable, ses valeurs sont bornées par un polynôme de la taille de l'entrée, c'est-à-dire, il existe un polynôme  $p \in \mathbb{R}[x]$  tel que pour tout  $x \in \{0, 1\}^*$ , on a  $|g(x)| \leq p(|x|)$ . On peut déduire de  $g$  une fonction  $h$  à sens unique et à longueur régulière en concaténant des zéros à la valeur  $g(x)$ , comme dans l'exercice précédent.

Maintenant, construisons une fonction  $f$  telle que  $|f(x)| = |x|$ . Cela se fait en posant  $z = x|y$  de la longueur de  $h(x)$  et en ne prenant en compte que  $x$  pour le calcul de la valeur :  $f(x|y) = h(x)$ .

Montrons maintenant que  $f$  est à sens unique. Supposons le contraire pour une contradiction. Il existe donc un algorithme  $A$  qui réussit avec une probabilité non négligeable à trouver un antécédent  $z$  pour un  $y = f(x)$  donné. Déduisons un algorithme  $B$  qui trouve un antécédent de  $g$ . Soit  $y = g(x)$  une entrée de  $B$  et soumettons  $y$  à  $A$ . L'algorithme  $A$  rend une valeur  $z = z_1 \cdots z_n$ . Pour tout entier  $m$  compris entre 1 et  $n$ , tester si  $g(x_1, \dots, x_m) = y$ . Dès que l'égalité est satisfaite renvoyer l'antécédent  $x_1 \cdots x_m$ , sinon, renvoyer une valeur aléatoire. L'algorithme  $B$  reste de complexité polynomiale et réussit avec la même probabilité que  $A$ .

#### 1.6.

1. Non! Pas toujours! Soit  $E = F \times F$  et  $f_1$  une fonction à sens unique  $F \rightarrow F$ . On montre facilement que la fonction :

$$f : \begin{array}{ll} E \times E & \longrightarrow E \times E \\ (x_1, x_2) & \longmapsto (f_1(x_2), 0) \end{array}$$

est à sens unique et vérifie  $f \circ f : (x_1, x_2) \mapsto (f_1(0), 0)$ . Et une fonction constante n'est clairement pas à sens unique.

2. Oui. Montrons que si la fonction  $f$  est à sens unique, alors la fonction  $g$  l'est aussi. Pour cela, montrons la contraposée. Si la fonction  $g$  n'est pas à sens unique, alors la fonction  $f$  ne l'est pas non plus. On suppose qu'il existe un algorithme qui inverse  $g$ , et déduisons en un algorithme  $B$  qui inverse  $f$ .

L'algorithme  $B$  prend en entrée un élément  $y$  de l'image de  $f$ .

- Calculer  $f(y)$ . C'est possible car  $f$  est calculable dans le sens direct.
- Soumettre le couple  $(y, f(y))$  à l'algorithme  $A$ . L'algorithme  $A$  retourne un antécédent  $y$  par  $g$  qui est un antécédent de  $f$ .

La probabilité de succès de l'algorithme  $B$  est la même que la probabilité de succès de l'algorithme  $A$ .

3. Oui. Comme à la question précédente, soit un algorithme  $A$  qui inverse  $f \circ f$  et construisons un algorithme  $B$  qui inverse  $f$ .

L'algorithme  $B$  prend en entrée un élément  $y$  de l'image de  $f$ . Il doit trouver l'unique antécédent de  $y$  par  $f$ .

- Soumettre  $y$  à l'algorithme  $A$ . Soit  $x$  la valeur rendue par l'algorithme  $A$  qui est supposée être la valeur qui vérifie  $f \circ f(x) = y$ .
- Rendre comme résultat  $f(x)$  qui est par construction l'antécédent de  $y$  par  $f$ .

L'algorithme  $B$  donne la bonne réponse avec une probabilité égale à celle de l'algorithme  $A$ .

La différence avec la première question est que, comme  $f$  est une bijection, un élément aléatoire du domaine de  $f$  est aussi un élément aléatoire de l'image de  $f$ , ce qui est faux si  $f$  est

## 1.7.

**2.1.** Soit  $(N, e)$  la clé publique RSA. Supposons pour une contradiction qu'il existe un oracle qui permet, pour tout message chiffré  $x^e \bmod N$  donné, de retourner la parité de  $x$ . Montrons que cette fonction peut être utilisée pour retrouver  $x$  tout entier à partir du cryptogramme  $y = x^e \bmod N$ . L'interrogation de l'oracle informe du bit de parité de  $x$ , c'est à dire de la valeur de son dernier chiffre dans la numération en base 2.

Interrogeons maintenant l'oracle avec la valeur  $2^e y \bmod N$  qui vaut  $(2x)^e \bmod N$ . Notons tout d'abord que le dernier chiffre binaire  $x_0$  de  $x$  est l'avant dernier chiffre binaire de  $2x$ . Il existe deux possibilités pour  $2x$  :

1. soit  $x < N/2$ , donc  $2x < N$  et  $2x \bmod N = 2x$ . Dans ce cas  $2x \bmod N$  est pair.
  2. soit  $x \geq N/2$ , donc  $2x \geq N$  et  $2x \bmod N = 2x - N$ . Et dans ce cas  $2x - N$  est impair.
- Dans les deux cas, on connaît à la fois le dernier chiffre binaire de  $2x \bmod N$ , mais aussi son avant dernier chiffre qui vaut  $x_0$  dans le premier cas et  $x_0 - N_1$  dans le second, où  $N_1$  est l'avant dernier chiffre binaire du module  $N$ .

En interrogeant l'oracle successivement avec les valeurs  $4y \bmod N$ ,  $8y \bmod N, \dots$ , on obtiendra successivement trois derniers bits de  $4x \bmod N$ , les quatre derniers bits de  $8x$ , etc.

Ainsi, si la taille de  $x$  est de  $\ell$  bits, en interrogeant  $\ell$  fois l'oracle, on obtient finalement tous les bits de  $2^{\ell-1}x$ . La valeur de  $x$  est obtenue par simple division par  $2^{\ell-1}$  modulo  $N$ .

**2.2.** Soit  $\alpha$  un générateur du groupe multiplicatif  $(\mathbb{F}_p, \times)$ . Pour tout entier  $n$ , la valeur de  $\alpha^n$  est un carré dans  $\mathbb{F}_p$  si et seulement si  $n$  est pair. à partir d'une valeur  $y = \alpha^n$ , il est facile, avec le symbole de LEGENDRE par exemple, de savoir si  $y$  est un carré, et donc de savoir si  $n$  est pair.

**2.3.** La contraposée est quasi immédiate. Si une fonction  $f$  injective n'est pas à sens unique, alors il existe un algorithme efficace pour trouver un antécédent  $x$  pour une valeur  $y$  donnée. Comme  $x$  est injective, cette valeur est unique. Tout bit de  $b$  de  $f$  se calcule alors efficacement à partir de  $x$ .



L'hypothèse d'injectivité est cruciale. Une fonction constante n'est bien sûr pas à sens unique, et pourtant, toute fonction booléenne non dégénérée de l'entrée, par exemple le bit de poids faible, est un bit difficile, puisque la connaissance de la valeur n'est d'aucune utilité pour trouver ce bit.

**3.1.** Soit  $g$  un GPA d'expansion  $n \mapsto 2n$ . Montrons que la fonction  $g$  est à sens unique. Pour cela, montrons la contraposée, c'est-à-dire que si  $g$  n'est pas à sens unique, alors  $g$  n'est pas un GPA. Si  $g$  n'est pas à sens unique, c'est qu'il existe un algorithme  $A$  qui sait trouver un antécédent avec une probabilité de succès non négligeable. Construisons alors un distinguéur  $D$  sachant discerner la réalisation d'un pseudo-aléa d'un aléa avec un avantage non négligeable. Considérons le distinguéur  $D$  suivant : pour un entier naturel  $n$  et chaque entrée  $X \in \{0, 1\}^{2n}$ , le distinguéur  $D$  soumet  $X$  à l'algorithme  $A$  qui lui renvoie un antécédent possible  $s$ . Si  $g(s) = X$ , alors probablement  $X$  est la réalisation d'un pseudo-aléa. Dans ce cas,  $D$  renvoie  $pa$ . Dans le cas contraire, alors probablement  $X$  est la réalisation d'un véritable aléa. Dans ce cas,  $D$  renvoie  $a$ . Montrons que l'avantage de  $D$  vaut asymptotiquement la probabilité de succès de  $A$ . L'avantage de  $D$  est donné par :

$$\text{Av}_n(D) = \Pr[D = pa \mid pa] - \Pr[D = pa \mid a].$$

Dans le monde  $pa$ , le distinguéur  $D$  rend  $pa$  si et seulement si  $A$  réussit à trouver l'antécédent de  $X$ . Le premier terme est exactement la probabilité de succès de  $A$ . Dans le monde aléatoire, le distinguéur  $D$  ne peut répondre  $pa$  que si  $A$  a trouvé un antécédent, et ceci ne peut survenir que si  $X$  appartient à l'image de  $g$ . Ceci ne peut survenir qu'avec une probabilité égale à  $\frac{2^n}{2^{2n}} = \frac{1}{2^n}$ , d'où :

$$\text{Av}_n(D) = \Pr_n^{\text{succès}}(A) - \frac{1}{2^n}.$$

Par hypothèse, la probabilité de succès de  $A$  est non négligeable. La différence d'une fonction non négligeable et d'une fonction négligeable est non négligeable, ce qui achève la preuve.

**3.2.** Cet exercice est destiné à montrer que, si  $t(n)$  est un polynôme en  $n$ , la concaténation  $t(n)$  fois d'un générateur pseudo-aléatoire reste un générateur pseudo-aléatoire.

Il faut montrer que les deux variables aléatoires suivantes sont calculatoirement indistinguables :

$$\begin{aligned} X &= (g(X_1), \dots, g(X_{t(n)})) \\ Y &= (Y_1, \dots, Y_{t(n)}) \end{aligned}$$

où les  $X_i$  sont des variables aléatoires uniformes sur  $\{0, 1\}^n$  et les  $Y_i$  sont des variables aléatoires uniformes sur  $\{0, 1\}^{2n}$ .

Afin d'appliquer la technique hybride, construisons une suite de variables aléatoires  $Z_i$ , pour  $i = 0$  à  $t(n)$ , telle que  $Z_0 = X$  et  $Z_{t(n)} = Y$ , et telles qu'on puisse montrer que deux termes consécutifs  $Z_i$  et  $Z_{i+1}$  sont calculatoirement indistinguables.

Si on arrive à majorer uniformément  $\Delta(Z_i, Z_{i+1})$  par une quantité négligeable  $\varepsilon(n)$ , cela permettra de conclure avec l'inégalité triangulaire :

$$\Delta(Z_0, Z_{t(n)}) \leq t(n)\varepsilon(n).$$

La construction est la suivante :

$$\begin{aligned} Z_0 &= ( \boxed{g(X_1)} \quad g(X_2) \quad \cdots \quad g(X_{t(n)}) ) \\ Z_1 &= ( \boxed{Y_1} \quad \boxed{g(X_2)} \quad \cdots \quad g(X_{t(n)}) ) \\ Z_2 &= ( Y_1 \quad \boxed{Y_2} \quad \cdots \quad g(X_{t(n)}) ) \\ &\vdots \\ Z_{t(n)-1} &= ( Y_1 \quad Y_2 \quad \cdots \quad \boxed{g(X_{t(n)})} ) \\ Z_{t(n)} &= ( Y_1 \quad Y_2 \quad \cdots \quad \boxed{Y_{t(n)}} ) \end{aligned}$$

La seule différence entre les variables  $Z_i$  et  $Z_{i+1}$  est dans la  $(i+1)^{\text{e}}$  composante qui vaut  $g(X_i)$  pour  $Z_i$  et qui vaut  $Y_i$  pour  $Z_{i+1}$ .

S'il existe un distingueur  $D$  entre  $Z_k$  et  $Z_{k+1}$ , on en déduit donc directement un distingueur de même avantage entre  $g(X_i)$  et  $Y_i$ . Par conséquent  $\Delta(Z_i, Z_{i+1}) \leq \Delta(g(X), Y)$ .

**3.3.** Le chiffre de parité de la fonction RSA est un prédicat difficile. Un générateur pseudo-aléatoire reposant sur la fonction RSA est défini comme suit :

$$\begin{aligned} N &= p \times q, \text{ où } p \text{ et } q \text{ sont deux nombres premiers} \\ e &\text{ est un entier premier avec } \text{ppcm}(p-1, q-1) \\ s_{k+1} &= s_k^e \\ \sigma_k &= s_k \bmod 2 \end{aligned}$$

**3.4.**

**3.5.** Considérons pour simplifier un exposant public  $e$  égal à 3. Rappelons que l'extraction des racine cubique dans  $\mathbb{Z}$  n'est pas un problème difficile. Il existe des algorithmes efficaces pour les extraire, par exemple la méthode de NEWTON. Soient  $n_1, n_2$  et  $n_3$  les trois modules RSA publics. L'exercice revient à montrer que la fonction :

$$f_{n_1 n_2 n_3} : m \longmapsto (m^3 \bmod n_1, m^3 \bmod n_2, m^3 \bmod n_3)$$

n'est pas à sens unique. La connaissance de l'image  $y = (m^3 \bmod n_1, m^3 \bmod n_2, m^3 \bmod n_3)$  permet, à l'aide du théorème des restes chinois, de trouver un élément  $z$  de  $\mathbb{Z}/n_1 n_2 n_3 \mathbb{Z}$  tel que  $z \equiv m^3$  modulo  $n_1 n_2 n_3$ . On suppose bien sûr que  $n_1, n_2$  et  $n_3$  sont deux à deux premiers entre eux, sinon, le calcul de pgcd donne un facteur strict de l'un d'entre eux, ce qui suffit à inverser la fonction RSA. Comme chaque composante  $m^3 \bmod n_i$  de  $y$  est inférieure à  $n_i$ , la valeur  $z$  vaut exactement  $m^3$ . Il suffit d'extraire sa racine cubique pour retrouver  $m$ .

**3.6.**

**4.1** Non ! Si la famille  $\mathcal{F}_n$  contient des fonctions qui admettent des paramètres de taille variable, le caractère pseudo-aléatoire de la famille n'est plus assuré. Pour montrer cela, exhibons un distingueur sachant reconnaître un élément d'une telle famille d'une fonction aléatoire.



Comme la famille constituée de fonction avec une taille fixe de paramètre, le distingueur devra interroger l'oracle avec des paramètres de taille différente.

En interrogeant l'oracle avec  $f(0) = y$  et  $f(00) = z$ , si  $z = g_0(y)$  alors  $f$  est très probablement un élément de la famille  $\mathcal{F}_n$ , sinon, c'est une fonction aléatoire.



Pour définir une famille de fonctions pseudo-aléatoires, il est crucial que les fonctions de cette famille opèrent sur des paramètres de taille fixe.

**4.2** On suppose que pour tout entier  $n$ , la famille  $\mathcal{F} = (f_k)_{k \in \{0,1\}^n}$  est une famille pseudo-aléatoire de fonctions  $\{0,1\}^{p(n)} \rightarrow \{0,1\}$  booléennes.

Pour construire une famille pseudo-aléatoires de fonctions vectorielles à valeurs dans  $\{0,1\}^n$ , le principe est de prendre un ensemble de  $n$  fonctions booléennes qui définiront les  $n$  composantes.

Considérons donc la famille  $\mathcal{G} = (g_s)_{s \in \{0,1\}^{n \times n}}$ , définie par :

$$\begin{aligned} s &= (k_1, \dots, k_n) \\ g_s(x) &= (f_{k_1}(x), \dots, f_{k_n}(x)) \end{aligned}$$

Il reste à montrer que la famille  $\mathcal{G}$  est une famille pseudo aléatoire. Montrons cela par la technique hybride. On construit une suite de familles de fonctions  $\mathcal{H}_0, \dots, \mathcal{H}_n$  telle que  $\mathcal{H}_0 = \mathcal{G}$  et  $\mathcal{H}_n$  est la

famille de fonctions vectorielles  $\{0, 1\}^{p(n)} \rightarrow \{0, 1\}^n$  dont les  $n$  composantes sont des fonctions booléennes aléatoires. La suite de famille est construite en remplaçant à chaque fois une composante pseudo-aléatoire de  $\mathcal{F}$  par une composante aléatoire :

$$\begin{aligned}\mathcal{H}_0 = \mathcal{F} : & \quad x \mapsto (f_{k_1}(x), f_{k_2}(x), \dots, f_{k_n}(x)) \\ \mathcal{H}_1 : & \quad x \mapsto (h_1(x), f_{k_2}(x), \dots, f_{k_n}(x)) \\ \mathcal{H}_2 : & \quad x \mapsto (h_1(x), h_2(x), \dots, f_{k_n}(x)) \\ & \quad \vdots \\ \mathcal{H}_n : & \quad x \mapsto (h_1(x), h_2(x), \dots, h_n(x))\end{aligned}$$

où  $h_1, \dots, h_n$  sont des fonctions booléennes aléatoires  $\{0, 1\}^{p(n)} \rightarrow \{0, 1\}$ .

La distance calculatoire entre deux familles  $\mathcal{H}_i$  et  $\mathcal{H}_{i+1}$  est négligeable, car un distingueur de ces deux familles est exactement comme un distingueur entre les composantes qui diffèrent qui sont  $f_{k_i}$  et  $h_i$ , et on a supposé que la famille  $\mathcal{F}$  des fonctions booléennes est pseudo-aléatoire.

On en déduit que la distance calculatoire  $\Delta(\mathcal{G}, \mathcal{H}_n)$  entre les familles  $\mathcal{G}$  et  $\mathcal{H}_n$  vaut  $n$  fois la distance calculatoire entre  $\mathcal{F}$  et la famille des fonctions booléennes aléatoires, ce qui reste négligeable.

**4.3** Par contraposée, montrons que si la famille n'est pas une famille pseudo-aléatoire de fonctions à sens unique, alors elle n'est pas une famille pseudo-aléatoire.

Par hypothèse, on suppose qu'il existe un algorithme  $A$  qui, étant donnée un élément  $y$  de  $\{0, 1\}^n$ , est capable de trouver un antécédent  $x$  de  $y$  par un élément aléatoire  $f_k$  de la famille, avec une probabilité non négligeable, uniquement en utilisant un oracle pour déterminer les valeurs de  $f_k$  dont il a besoin pour effectuer cette tâche.

Considérons l'algorithme  $B$  qui exécute les instructions suivantes :

- Choisir un élément  $x$  aléatoire de  $\{0, 1\}^n$ .
- Soumettre cet élément à l'oracle qui renvoie  $y = f_k(x)$ .
- Soumettre maintenant  $y$  à l'algorithme  $A$  qui est supposé retourner un antécédent de  $y$ . Soit  $x'$  l'élément de  $\{0, 1\}$  retourné par l'algorithme  $A$ .
- Soumettre à nouveau  $x'$  à l'oracle qui retournera  $y' = f(x')$ .
- Si  $y = y'$  alors l'algorithme  $A$  a réussi. Il y a de fortes chances que l'oracle réalise un élément de la famille pseudo-aléatoire. Dans ce cas, l'algorithme  $B$  rend 1 pour signifier que l'oracle réalise une fonction pseudo-aléatoire.

Sinon, il y a de fortes chances aussi pour que l'oracle réalise une fonction parfaitement aléatoire, l'algorithme rend 0 pour signifier que l'oracle réalise une fonction aléatoire.

Il reste des détails de calcul pour vérifier que si l'avantage de l'algorithme  $A$  n'est pas négligeable, alors l'avantage de l'algorithme  $B$  n'est pas négligeable non plus.



Par un raisonnement similaire, on peut montrer qu'une famille de fonctions pseudo-aléatoire cache tous les bits des entrées. Un adversaire ne peut déterminer un bit  $f(x)$  d'un antécédent  $x$  d'un élément  $y$  qu'avec une probabilité négligeable.

**7.1 a)** La preuve est directe :  $f_2 \circ i(x) = i(x) \oplus i \circ i(x) = i(x) \oplus x = f_2(x)$ .

b) On a  $L_3 = L_1 + f_2(R_1)$ , mais  $L_1 = R_0$ , donc  $L_3 = R_0 + f_2(R_1)$ . On peut utiliser la question a) pour trouver deux requêtes telles que  $R_0 = R'_0$ . Pour avoir  $L_3 = L'_3$ , il faut  $f_2(R_1) = f_2(R'_1)$ . Prendre  $R_1 = R'_1$  ne convient pas, car cela conduirait à  $L_0 = R'_0$ . Par contre, on peut chercher à avoir  $R'_1 = i(R_1)$ . D'après la question a), ces deux valeurs ont la même image par  $f_2$ .

Cela impose comme condition  $R'_1 = i(R_1) = R_1 \oplus f_2(R_1) = R_1 \oplus L_3 \oplus R_0$  d'une part, et  $R'_1 \oplus R_1 = L_0 + \oplus L'_0$  d'autre part. Si on choisit  $L'_0 = R_0 \oplus L_0 \oplus L_3$ , alors on aura  $L_3 = l'_3$ .

L'algorithme  $A$  pour discerner entre un tel schéma  $\mathcal{F}$  et une fonction aléatoire  $\mathcal{R}$  est donc le suivant :

1. Poser une requête  $(L_0, R_0)$ . Soit  $(L_3, R_3)$  la réponse.
2. Poser une seconde requête  $(L'_0, R'_0)$  avec  $R'_0 = R_0$  et  $L'_0 = L_0 \oplus R_0 \oplus L_3$ . Soit  $(L'_3, R'_3)$  la réponse.
3. Si  $L_3 = L'_3$ , alors rendre 1 pour décider que l'oracle est un schéma  $\mathcal{F}$ , sinon, rendre 0 pour décider que l'oracle est un schéma  $\mathcal{R}$ .

Estimons l'avantage d'un tel algorithme.

$$\text{Av}^A = \Pr[A^{\mathcal{F}} = 1] - \Pr[A^{\mathcal{R}} = 1],$$

où  $\Pr[A^{\mathcal{F}} = 1]$  est la probabilité que l'algorithme  $A$  réponde 1 en présence d'un schéma  $\mathcal{F}$  et  $\Pr[A^{\mathcal{R}} = 1]$  est la probabilité que l'algorithme  $A$  réponde 1 en présence d'une fonction aléatoire  $\mathcal{R}$ .

Si l'oracle est un schéma  $\mathcal{F}$ , l'algorithme répond toujours correctement, car on a toujours  $L_3 = L'_3$  dans ce cas là, donc  $\Pr[A^{\mathcal{F}} = 1] = 1$ .

L'algorithme échoue en face d'un oracle qui est une fonction aléatoire si  $R_0 = L_3$ , car dans ce cas, la deuxième requête sera identique à la première, ou si fortuitement,  $L_3 = L'_3$ . Partageons l'événement «  $A^{\mathcal{R}} = 1$  » selon que  $L_3$  est ou non égal à  $R_0$ .

$$\begin{aligned} \Pr[A^{\mathcal{R}} = 1] &= \underbrace{\Pr[A^{\mathcal{R}} = 1 \mid R_0 = L_3]}_{\leq 1} \times \underbrace{\Pr[R_0 = L_3]}_{= \frac{1}{2^n}} - \\ &\quad \underbrace{\Pr[A^{\mathcal{R}} = 1 \mid R_0 \neq L_3]}_{= \frac{1}{2^n}} \times \underbrace{\Pr[R_0 \neq L_3]}_{\leq 1} \end{aligned}$$

En effet, si la fonction est aléatoire, la probabilité d'avoir  $R_0 = L_3$  vaut  $1/2^n$ , et si l'oracle est une fonction aléatoire, la probabilité que l'algorithme réponde 1 est la probabilité que fortuitement, on ait  $L_3 = L'_3$  qui vaut aussi  $1/2^n$ . L'avantage de l'algorithme  $A$  vaut donc  $1 - 1/2^{n-1}$  qui est proche de 1, et donc non négligeable.

**7.2** La famille  $(\mathcal{G}_n)$  est une famille de bijections, car l'image réciproque par  $g_k$  du vecteur  $(y_0, \dots, y_n)$  est le vecteur  $(x_0, \dots, x_{n-1}, x_0 \oplus y_n)$ , où  $(x_0, \dots, x_{n-1})$  est l'image réciproque par  $f_k$  du vecteur  $(y_0, \dots, y_{n-1})$ .

Si une famille de fonctions ou de permutations n'est pas pseudo-aléatoire, c'est qu'il existe une relation prévisible entre les entrées et les sorties. Une fois cette relation trouvée, il est facile d'élaborer un algorithme qui distinguera cette famille d'une famille de fonctions aléatoires. Dans le cas de cet exercice, la première composante de la valeur vaut toujours  $x_0 \oplus b$ , ce qui suggère immédiatement l'algorithme suivant :

1. Soumettre une requête quelconque  $(x_0, \dots, x_{n-1}, b)$ . Soit  $(y_0, \dots, y_n)$  la réponse.
2. Si  $y_n = x_0 \oplus b$  alors retourner 1 pour signifier que l'oracle réalise la famille  $\mathcal{G}$  de l'énoncé. Sinon, retourner 0 pour signifier que l'oracle réalise une permutation aléatoire.

L'avantage de cet algorithme est

$$\text{Av}^A = \left| \underbrace{\Pr_{F \in_R \mathcal{R}}[A^F = 1]}_{= 1/2} - \underbrace{\Pr_{F \in_R \mathcal{G}}[A^F = 1]}_{= 1} \right| = \frac{1}{2}$$

En effet, cet algorithme donne une réponse toujours juste si l'oracle réalise un élément de la famille  $\mathcal{G}$ , et si l'oracle est une permutation aléatoire, il se peut que fortuitement on ait  $y_n = x_{n-1} \oplus b$ , auquel cas l'algorithme donne une fausse réponse, et cela survient avec une probabilité égale à  $1/2$ .

# Index alphabétique

- **A** —
- Algorithmica . . . . . 73.
- **C** —
- CBC
  - mode — . . . . . 63.
  - MAC . . . . . 64.
- CCA . . . . . 89.
- Cryptomania . . . . . 74.
- **D** —
- distance calculatoire . . . . . 28.
- **E** —
- ECB
  - mode — . . . . . 63.
- **F** —
- famille
  - de fonctions pseudo-aléatoire . . . . 36.
  - de permutations pseudo-aléatoire . . 57.
- fonction
  - négligeable . . . . . 12.
  - à sens unique asymptotique . . . . . 12.
- FEISTEL
  - schéma de — . . . . . 58.
- **G** —
- générateur pseudo-aléatoire (GPA) . . . . 25.
- **H** —
- Heuristica . . . . . 73.
- hybride
  - procédé — de démonstration . . . . . 31.
- **I** —
- indistinguabilité . . . . . 44.
- **L** —
- LUBY – RACKOFF . . . . . 60.
- **M** —
- MAC . . . . . 52.
- malléabilité . . . . . 51.
- Minicrypt . . . . . 74, 84.
- mode
  - CBC . . . . . 63.
  - ECB . . . . . 63.
- monoïde libre . . . . . 12.
- **O** —
- oracle aléatoire . . . . . 90.
- **P** —
- PAILLIER
  - chiffrement de — . . . . . 33.
- Pessiland . . . . . 74.
- pince (de bijection) . . . . . 86.
- preuve sans divulgation . . . . . 22.
- problème
  - 3COL . . . . . 98.
- **S** —
- sémantique
  - sécurité — . . . . . 43.
- **Z** —
- zero knowledge . . . . . 22.