Chapitre 1

Introduction

On s'intéresse ici au chiffrement authentifié symétrique (parfois dit « à clef secrète » par opposition à la cryptographie à clef publique). L'objectif de ces séances consiste à montrer comment on peut réaliser une primitive cryptographique particulière : le *chiffrement symétrique authentifié (sémantiquement sûr)*.

Définition 1 : Syntaxe du chiffrement

> Un mécanisme de chiffrement est constitué par trois algorithmes polynomiaux probabilistes (G, E, D) soumis aux conditions suivantes :
>
> 1. G (key Generation) exécuté sur 1^n produit une chaine de bits (la clef).
> 2. *Correction* : pour tout $k \leftarrow G(1^n)$ et pour tout $\alpha \in \{0,1\}^*$, les algorithmes E (encryption) et D (decryption) satisfont $\mathbb{P}\left[D(k, E(k, \alpha)) = \alpha\right] = 1$.
> 3. D est (généralement) déterministe, et peut éventuellement renvoyer une valeur spéciale $\perp$ qui indique une erreur lors du déchiffrement.

L'entier n dans la définition est le *paramètre de sécurité*. Comme G est polynomial, il peut produire une clef dans la taille est poly(n) (si on lui donnait n en binaire, il ne pourrait pas — cette convention vaut ce qu'elle vaut, mais on va la garder). On note que dans la définition de la correction, c'est la *même* clef qui est utilisée pour chiffrer et pour déchiffrer. La chaine de bit $E(k, \alpha)$ est le chiffrement de α sous la clef k. On le notera parfois $E_k(\alpha)$ ou encore $\{\alpha\}_k$.

En cryptographie asymétrique, les algorithmes de génération de clefs sont souvent non-triviaux et les clefs sont parfois des objets structurés. Par contre, la quasi-totalité des constructions habituelles de la cryptographie symétrique fonctionnent avec un algorithme G qui renvoie simplement n bits aléatoires.

Bien sûr, pour que tout ceci ait un intérêt, il ne faut pas qu'il existe d'algorithme polynomial $\mathcal{A}$ tel que $\mathcal{A}(E_k(\alpha)) = \alpha$ avec probabilité non-negligeable.

Spécification informelle Un mécanisme de chiffrement authentifié sémantiquement sûr doit avoir les caractéristiques suivantes :

- *Sécurité sémantique* : connaissante $E_k(\alpha)$, il est impossible d'extraire quelque information que ce soit sur α sans connaître k (exceptée la taille de α).
- *Intégrité des chiffrés* : il est impossible de produire une chaine de bits β telle que $D_k(\beta) \neq \perp$ sans connaître k.

Le premier point affirme que le mécanisme de chiffrement peut servir à garantir la confidentialité de messages qui doivent transiter par un canal non-fiable. Le deuxième point affirme qu'il garantit l'authenticité du message et l'identité de l'expéditeur : si le déchiffrement est correct, c'est que le message a été produit par le bon expéditeur (qui connaît la clef secrète), et qu'il n'a pas été modifié entre temps.

Justification Le chiffrement authentifié est la « bonne » notion de sécurité. C'est la bonne interface entre les utilisateurs du chiffrement et le mécanisme de chiffrement.

Imaginons qu'un processus S (sender) envoie des messages $m_1, m_2, \ldots$ à un autre processus R (receiver) à travers un réseau non-fiable (il peut s'agir d'un protocole de discussion, ou bancaire, etc.). Un mécanisme de chiffrement authentifié est utilisé : R et S se sont déjà mis d'accord sur une clef symétrique aléatoire. Pour envoyer un message, R le place dans une sorte « boite d'envoi » et ne se préoccupe pas des détails.

En fait, quand un message m_i est placé dans la boite d'envoi, il est chiffré en $c_i \leftarrow E_k(m_i)$ puis envoyé sur le réseau. Lorsqu'une chaine de bits $\hat{c}$ arrive de l'autre côté du réseau, le déchiffrement a lieu : $\hat{m} \leftarrow D_k(\hat{c})$. Si $\hat{m} \neq \perp$, alors $y\hat{m}$ est placé dans la « boite de réception » du processus R. Il peut alors être traité, sans que R n'ait vraiment à se préoccuper de comment il est arrivé là.

Un adversaire pourrait manipuler les communications entre R et S de plusieurs manières :

- il peut re-ordonner dupliquer ou encore jeter à la poubelle des messages chiffrés envoyés par S.
- Il peut modifier des chiffrés envoyés par S, ou bien en forger de nouveaux.
- Il peut connaître partiellement, voire être en mesure d'influencer, le contenu des messages clairs envoyés par S.
- Il peut, en observant le comportement de R, apprendre des informations partielles sur le contenu des messages reçus.

Imaginons un instant que ce mécanisme de communication « concret » soit remplacé par un mécanisme de communication « idéal » qui téléporte directement les messages (et qui n'est donc pas réalisable dans ce bas-monde). Quand S dépose m_i dans sa boite d'envoi, un chiffré bidon est produit : $c_i \leftarrow E_k\left(1^{|m_i|}\right)$. Il ne contient aucune information sur m_i, mais sert « d'identifiant » pour m_i. Lorsque c_i arrive chez R, le clair correspondant m_i est copié par magie de la boite d'envoi de S vers la boite de réception de S. Si un chiffré arrive chez R qui n'est pas l'un des c_i préalablement produit, il est ignoré.

Dans cette implémentation « idéale », les chiffrés peuvent toujours être retirés, dupliqués, etc. et par conséquent les clairs aussi pourront être permutés, dupliqués, etc. Mais en incluant des numéros de séquence dans les message, le protocole à l'oeuvre dans R et S peut détecter ces problèmes et les régler.

On prétend que, du point de vue de l'adversaire, la situation réelle et la situation idéale sont indistinguable. Or, dans la situation idéale, les « vrais » messages ne transitent *pas* par le réseau. Justifions ceci informellement, en considérant une succession de versions du système. On prétend que l'adversaire ne peut pas faire la différence entre deux versions successives.

Version 0. C'est l'implémentation réelle.

Version 1. On modifie la gestion de la boite de réception de R. Lorsqu'une chaine de bits $\hat{c}$ arrive, les chiffrés $c_1, c_2, \ldots$ produits par S sont examinés. Si $\hat{c} = c_i$, alors m_i est copié magiquement dans la boite de réception de R, sans exécuter l'algorithme de déchiffrement. Sinon, on fait comme avant.

Du point de vue de R et S, ça ne change rien, car exécuter le déchiffrement aurait bien donné le même résultat (le mécanisme de chiffrement est correct). En particulier, le comportement extérieur de R n'est pas modifié. Par conséquent, l'adversaire ne peut pas percevoir de différence entre la version 0 et la version 1.

Version 2. On remodifie la gestion de la boite d'arrivée de R : si $\hat{c}$ arrive, et que $\hat{c}$ n'est pas l'un des chiffrés produits par S, alors il est ignoré. Sinon, on fait comme avant. (Remarque : après cette modification, l'algorithme de déchiffrement n'est plus jamais exécuté).

Par rapport à la version 1, le comportement de R est modifié : dans la version 1, R aurait traité un chiffré valide même si ce dernier n'est pas l'un de ceux produits par S. Mais pour percevoir un comportement de R différent entre la v1 et la v2, il faudrait que l'adversaire soit capable de produire un chiffré valide sans connaître la clef, ce qui contredirait la propriété d'intégrité des chiffrés du mécanisme de chiffrement authentifié. Par conséquent, l'adversaire ne peut pas observer de différence entre la version 1 et la version 2.

Version 3. On modifie la gestion de la boite d'envoi de S : les chiffrés produits et expédiés sont maintenant les chiffrés bidons $c_i \leftarrow E_k\left(1^{|m_i|}\right)$. Tout le reste se passe comme dans la version 2.

Le comportement de R n'est pas modifié par ce changement. Si l'adversaire était capable de percevoir la différence entre la v2 et la v3, alors il serait capable de percevoir des informations sur les messages clairs en observant les messages chiffrés, ce qui contredirait la sécurité sémantique du mécanisme de chiffrement authentifié.

Version 4. C'est l'implémentation idéale définie ci-dessus. En fait, elle est identique à la version 3.

Par transitivité, l'adversaire ne peut donc pas faire la différence entre l'implémentation réelle, avec chiffrement authentifié, et l'implémentation idéale avec téléportation. C'est donc que le chiffrement authentifié permet la transformation d'un canal de communication non-fiable en canal fiable.

Construction générique Le chiffrement authentifié dont il est question peut être obtenu en combinant un mécanisme de chiffrement sémantiquement sûr sous attaques à clairs choisis d'une part, et un Code d'Authentification de Message (MAC) d'autre part.

Chapitre 2

Mécanismes de chiffrement

Dans ce chapitre, on définit la sécurité du chiffrement symétrique, et on construit des mécanismes de chiffrements sûrs sous l'hypothèse que l'AES est un système de chiffrement par bloc sûr.

2.1 Notions de sécurité avec un seul message

On a déjà vu la syntaxe mais on avait laissé de côté la définition précise de la sécurité. La difficulté consiste à formaliser la notion de « un adversaire n'apprend *rien* à partir des chiffrés ».

Il est possible de formaliser cette notion, la *sécurité sémantique*, qui est puissante, mais la définition n'est pas très commode. Du coup, comme on a vu pour la cryptographie à clef publique, on utilise une notion d'indistinguabilité.

Pour donner un sens à cette notion, imaginons qu'un adversaire $\mathcal{A}$ (c'est-à-dire un algorithme polynomial randomizé) doive jouer à une sorte de **jeu** sadique mettant en oeuvre deux joueurs : l'adversaire lui-même et le « maître du jeu » qu'on nommera le **challenger**. Le jeu se déroule de la façon suivante :

Jeu 1 (Indistinguabilité des chiffrés)

> Pour un mécanisme de chiffrement $\mathcal{E} = (G, E, D)$ et étant donné un adversaire représenté par deux machines polynomiales probabilistes $\mathcal{A}_1$ et $\mathcal{A}_2$, l'**expérience** $b \in \{0, 1\}$ est définie par :
>
> 1. Le Challenger génère une clef symétrique : $k \leftarrow G(1^n)$.
> 2. L'Adversaire choisit deux messages (de la même taille) : $m_0, m_1, \sigma \leftarrow \mathcal{A}_1(1^n)$.
> 3. Le Challenger calcule $c \leftarrow E_k(m_b)$.
> 4. L'adversaire produit un bit $\hat{b} \leftarrow \mathcal{A}_2(\sigma, c)$
>
> On dit que $\hat{b}$ est la **valeur de retour** de $\mathcal{A}$. On note W_b l'évènement « $\mathcal{A}$ renvoie 1 dans l'expérience b » et on définit l'**avantage** de $\mathcal{A}$ par :
>
> $$\mathbf{AdvINV}[\mathcal{E}, \mathcal{A}] = \left|\mathbb{P}\left[W_0\right] - \mathbb{P}\left[W_1\right]\right|.$$

L'avantage est un nombre compris entre 0 et 1. L'avantage mesure la capacité de l'adversaire à se comporter différemment selon qu'on lui donne un chiffré de m_0 ou de m_1. S'il vaut zéro, c'est que l'adversaire n'a aucune efficacité contre le mécanisme de chiffrement. S'il vaut 1, c'est que l'adversaire « casse » le mécanisme de chiffrement avec de façon déterministe.

Par exemple, si l'adversaire renvoie un bit aléatoire, alors il a un avantage de zéro, puisqu'il renvoie 1 avec probabilité 1/2 dans les deux expériences ($b = 0$ et $b = 1$). Ceci est conforme à notre intuition que « le chiffrement ne doit laisser filtrer aucune information ».

L'adversaire, qui ne connaît pas la clef k, n'a aucun moyen de faire chiffrer/déchiffrer quoi que ce soit (contrairement à ce qui se passe dans le monde asymétrique, où il peut chiffrer car il possède la clef publique). La chaine de bits σ, transmise entre les deux « phases » de l'adversaire lui permet de maintenir une « mémoire » entre les deux phases.

Définition 2 : Indistinguabilité des chiffrés

Un mécanisme de chiffrement $\mathcal{E}$ **possède l'indistinguabilité des chiffrés** si l'avantage de tout adversaire polynomial au jeu ci-dessus est négligeable (c.a.d. s'il existe un polynôme p tel que pour tout n suffisamment grand, l'avantage est inférieur à $1/p(n)$).

Pour voir la définition à l'oeuvre, imaginons un référendum en ligne où les électeurs votent 0 (`NON`) ou 1 (`OUI`). Lors de son inscription sur les listes électorales, chaque électeur i reçoit une clef secrète $k_i \leftarrow G(1^n)$. Pour voter, chaque électeur chiffre son vote $c_i \leftarrow E_{k_i}(\texttt{vote}_i)$ et transmet c_i au bureau de vote. Supposons qu'un adversaire (c.a.d. un algorithme polynomial randomisé) intercepte les votes et tente de déterminer si un électeur donné a voté `OUI` ou `NON`. Disons qu'il se donne comme objectif de renvoyer 1 si le vote chiffré était `OUI`. Si le mécanisme de chiffrement possède des chiffrés indistinguables, alors l'adversaire ne peut pas faire grand-chose : il suffit d'appliquer la définition avec $m_0 = \texttt{OUI}$ et $m_1 = \texttt{NON}$ pour voir que l'adversaire renvoie 1 dans les deux cas avec quasiment la même probabilité.

Définition alternative . On considère une autre manière de définir l'indistinguabilité des chiffrés, en laissant le challenger choisir aléatoirement le bit b. L'adversaire doit tenter de renvoyer b (c'est une manière plus spécifique d'avoir un comportement qui dépend de b).

Jeu 2 (Indistinguabilité des chiffrés, version « bit-guessing »)

1. Le Challenger génère une clef symétrique : $k \leftarrow G(1^n)$ et tire au hasard un bit b.
2. L'Adversaire choisit deux messages (de la même taille) : $m_0, m_1, \sigma \leftarrow \mathcal{A}_1(1^n)$ et transmet (m_0, m_1) au challenger.
3. Le Challenger calcule $c \leftarrow E_k(m_b)$.
4. L'adversaire renvoie un bit $\hat{b} \leftarrow \mathcal{A}_2(\sigma, c)$

On dit que $\mathcal{A}$ **gagne** le jeu si $b = \hat{b}$ et on définit l'**avantage** de $\mathcal{A}$ par :

$$\mathbf{AdvIND^*}[\mathcal{E}, \mathcal{A}] = 2\left(\mathbb{P}\left[b = \hat{b}\right] - \frac{1}{2}\right).$$

Un adversaire qui devine systématiquement le *contraire* de b a un avantage AdvIND* de -1. Un adversaire qui répond toujours 0, toujours 1, ou au hasard a un avantage nul.

En fait, les deux définitions sont rigoureusement équivalentes.

Théorème 1. *Pour tout algorithme polynomial randomisé $\mathcal{A}$, on a* $\boldsymbol{AdvIND}[\mathcal{E}, \mathcal{A}] = \left|\boldsymbol{AdvIND^*}[\mathcal{E}, \mathcal{A}]\right|$.

Démonstration

Soit $\mathcal{A}$ un algorithme polynomial randomisé, et faisons-le jouer au jeu 2. D'après la définition de **AdvIND***, on a :

$$\begin{aligned}
\mathbb{P}\left[\hat{b} = b\right] &= \mathbb{P}\left[\hat{b} = 0 \text{ et } b = 0\right] + \mathbb{P}\left[\hat{b} = 1 \text{ et } b = 1\right] \\
&= \mathbb{P}\left[\hat{b} = 0 \mid b = 0\right]\mathbb{P}\left[b = 0\right] + \mathbb{P}\left[\hat{b} = 1 \mid b = 1\right]\mathbb{P}\left[b = 1\right] \\
&= \frac{1}{2}\left(\mathbb{P}\left[\hat{b} = 0 \mid b = 0\right] + \mathbb{P}\left[\hat{b} = 1 \mid b = 1\right]\right) \\
&= \frac{1}{2}\left(1 - \mathbb{P}\left[\hat{b} = 1 \mid b = 0\right] + \mathbb{P}\left[\hat{b} = 1 \mid b = 1\right]\right) \\
&= \frac{1}{2} + \frac{1}{2}\left(\mathbb{P}\left[\hat{b} = 1 \mid b = 1\right] - \mathbb{P}\left[\hat{b} = 1 \mid b = 0\right]\right)
\end{aligned}$$

(Ce calcul a été volontairement très détaillé. On ira plus vite la prochaine fois). On a donc :

$$\mathbf{AdvIND^*}[\mathcal{E}, \mathcal{A}] = \mathbb{P}\left[\hat{b} = 1 \mid b = 1\right] - \mathbb{P}\left[\hat{b} = 1 \mid b = 0\right].$$

Maintenant, faisons jouer le même algorithme au jeu 1 (après tout, du point de vue de $\mathcal{A}$, les deux jeux ont la même « interface »). Par définition de AdvIND, on trouve que :

$$\mathbf{AdvIND}[\mathcal{E}, \mathcal{A}] = \left|\mathbb{P}\left[\hat{b} = 1 \mid b = 1\right] - \mathbb{P}\left[\hat{b} = 1 \mid b = 0\right]\right|.$$

On en conclut donc que $\mathbf{AdvIND}[\mathcal{E}, \mathcal{A}] = \left|\mathbf{AdvIND^*}[\mathcal{E}, \mathcal{A}]\right|$. Ceci conclut la preuve du théorème. □

Le chiffrement est à sens unique Démontrons que la notion de sécurité a bien les conséquences auquelles on s'attend, en démontrant qu'un adversaire ne peut pas récupérer le clair en observant le chiffré.

On considère un nouveau jeu.

Jeu 3 (Inversion du chiffrement)

Pour un mécanisme de chiffrement $\mathcal{E} = (G, E, D)$ et étant donné un adversaire représenté par une machine polynomiale probabiliste $\mathcal{A}$, l'expérience est définie par :

1. Le challenger génère une clef aléatoire $k \leftarrow G(1^n)$, puis tire aléatoirement un message $m \in \{0,1\}^{\ell(n)}$, où $\ell(n) \geq n$ est polynomialement bornée.
2. Le challenger chiffre m en calculant $c \leftarrow E_k(m)$ puis soumet le chiffré c à l'adversaire.
3. L'adversaire renvoie une chaine de bits $\hat{m}$ et il gagne si $\hat{m} = m$.

Si l'adversaire gagne, on dit que l'évènement W est réalisé. Son avantage est $\mathbf{AdvOW}[\mathcal{E}, \mathcal{A}] = \left|\mathbb{P}[W] - \frac{1}{2^{\ell(n)}}\right|$. Ceci capture la capacité de $\mathcal{A}$ à faire mieux de de répondre au hasard.

Définition 3 : Chiffrement à sens unique

Un mécanisme de chiffrement $\mathcal{E}$ est **à sens unique** si $\mathbf{AdvOW}[\mathcal{E}, \mathcal{A}]$ est négligeable pour tout adversaire $\mathcal{A}$ fonctionnant en temps polynomial.

On aboutit logiquement à :

Théorème 2. *Si un mécanisme de chiffrement $\mathcal{E}$ possède des chiffrés indistinguables, alors il est à sens unique.*

La preuve de ce théorème repose sur une *réduction*, qui est une technique de preuve standard.

Démonstration

On démontre la contraposée en effectuant une réduction : on suppose qu'il existe un adversaire $\mathcal{A}$ qui a un avantage $\mathbf{AdvOW}[\mathcal{E}, \mathcal{A}]$ non-négligeable, et on démontre que $\mathcal{E}$ ne possède pas des chiffrés indistinguables.

Supposons donc qu'il existe un adversaire $\mathcal{A}$ dont l'avantage ne soit pas négligeable : il existe donc un polynôme p tel que pour une infinité de n on ait $\mathbf{AdvOW}[\mathcal{E}, \mathcal{A}] > 1/p(n)$.

Grâce à cette hypothèse, on va pouvoir construire un adversaire $\mathcal{B}$ contre la notion d'indistinguabilité des chiffrés qui va avoir un avantage **AdvIND** non-négligeable.

Construisons donc $\mathcal{B}$: il fabrique deux messages aléatoires de taille $\ell(n)$, puis les envoie au challenger. Celui-ci renvoie $c = E_k(m_b)$. L'adversaire $\mathcal{B}$ qu'on essaye de construire doit déterminer s'il s'agit d'un chiffré de m_0 ou de m_1. Pour cela, on utilise l'adversaire $\mathcal{A}$: on s'en sert pour calculer $\hat{m} \leftarrow \mathcal{A}(c)$ [en fait on tente le déchiffrement]. Si $\hat{m} = m_1$, $\mathcal{B}$ renvoie 1, sinon $\mathcal{B}$ renvoie 0.

Tout d'abord, on remarque que $\mathcal{B}$ simule correctement le challenger avec lequel $\mathcal{A}$ s'attend à interagir. L'intuition, c'est que si $\mathcal{A}$ marche bien, alors $\mathcal{B}$ a un comportement très différent selon que $b = 0$ ou $b = 1$.

Il nous faut démontrer que l'avantage **AdvIND** n'est pas négligeable. Allons-y :

$$\begin{aligned}\mathbf{AdvINV}[\mathcal{E}, \mathcal{B}] &= \left|\mathbb{P}[W_0] - \mathbb{P}[W_1]\right| \\ &= \left|\mathbb{P}[\mathcal{B} \text{ renvoie 1 et } b = 1] - \mathbb{P}[\mathcal{B} \text{ renvoie 1 et } b = 0]\right| \\ &= \left|\mathbb{P}[\mathcal{A}(E_k(m_1)) = m_1] - \mathbb{P}[\mathcal{A}(E_k(m_0)) = m_1]\right|.\end{aligned}$$

Considérons les deux probabilités séparément.

La première est la probabilité que l'algorithme $\mathcal{A}$ fonctionne correctement (c.a.d. réussisse à inverser le chiffrement).

Penchons-nous ensuite sur la deuxième : on note que, dans la mesure où m_0 et m_1 sont choisis aléatoirement, l'algorithme $\mathcal{A}$ ne peut pas « deviner m_1 » car il ne possède aucune information dessus (m_1 est indépendant de m_0). Comme m_1 est aléatoire, la probabilité que la chaine de bits renvoyée par $\mathcal{A}$ soit m_1 est exactement $2^{-\ell(n)}$.

Par conséquent, on a :

$$\begin{aligned}\mathbf{AdvINV}[\mathcal{E}, \mathcal{B}] &= \left|\mathbb{P}[\mathcal{A} \text{ gagne le jeu d'inversion}] - 2^{-\ell(n)}\right| \\ &= \mathbf{AdvOW}[\mathcal{E}, \mathcal{A}].\end{aligned}$$

Ceci conclut la preuve, car du coup $\mathbf{AdvIND}[\mathcal{E}, \mathcal{B}]$ n'est pas négligeable, et $\mathcal{E}$ n'a donc pas des chiffrés indistinguables. □

2.2 Le One-Time Pad

Le One-Time Pad est un mécanisme de chiffrement qui chiffre n bits avec une clef de n bits et qui consiste à poser $G(1^n) := (n \text{ bits aléatoires}), E(k,x) := x \oplus k$ et $D(k,y) := y \oplus k$. Il a été utilisé pour le « téléphone rouge » entre Washington et Moscou pendant la guerre froide (le fameux téléphone était en fait un telex, et les clefs étaient transportées par avion dans des valises diplomatiques).

Le One-Time Pad ne satisfait pas complètement la définition 1 du chiffrement, car il ne peut chiffrer que des messages de taille n.

Théorème 3. *Le One-Time Pad possède des chiffrés indistinguables. En fait, aucun adversaire ne peut obtenir un avantage supérieur à zéro face au One-Time Pad.*

Démonstration

> Notons U_n une variable aléatoire uniformément distribuée dans $\{0,1\}^n$, et considérons deux chaines de bits m_0, m_1 de taille n. Par définition du One-Time Pad, on a : $E_{G(1^n)}(m_0) = m_0 \oplus U_n$ et $E_{G(1^n)}(m_1) = m_1 \oplus U_n$.
>
> **Claim** : $E_{G(1^n)}(m_0)$ et $E_{G(1^n)}(m_1)$ sont deux variables aléatoires uniformément distribuées dans $\{0,1\}^n$.
>
> **Démonstration**
>
> > En fait, c'est plus fort que ça : pour toute permutation $f : \{0,1\}^n \to \{0,1\}^n$, $Vn := f(U_n)$ est également une variable aléatoire uniformément distribuée dans $\{0,1\}^n$. En effet, pour tout $\alpha \in \{0,1\}^n$, $\mathbb{P}\left[f(U_n) = \alpha\right] = \mathbb{P}\left[U_n = f^{-1}(\alpha)\right]$ (ceci car f est bijective). Cette dernière probabilité vaut 2^{-n}, car U_n est uniformément distribué sur $\{0,1\}^n$. Pour revenir au « claim » : il suffit de considérer les permutations $z \mapsto z \oplus m_0$ et $z \mapsto z \oplus m_1$.
>
> Par conséquent, considérons un adversaire $\mathcal{A}$ qui joue au jeu de l'indistinguabilité des chiffrés. Le claim affirme que le chiffré que $\mathcal{A}$ reçoit est une séquence de n bits uniformément aléatoire, dans les deux expériences $b = 0$ et $b = 1$. Du coup, du point de vue de l'adversaire, il se passe *exactement la même chose* dans les deux expériences, et il ne peut pas les différencier.
>
> $$\begin{aligned}\mathbf{AdvINV}[\mathcal{E},\mathcal{A}] &= \left|\mathbb{P}\left[\mathcal{A} \text{ renvoie 1 et } b=1\right] - \mathbb{P}\left[\mathcal{A} \text{ renvoie 1 et } b=0\right]\right|,\\ &= \left|\mathbb{P}\left[\mathcal{A}_2(\mathcal{A}_1(1^n), U_n) = 1\right] - \mathbb{P}\left[\mathcal{A}_2(\mathcal{A}_1(1^n), U_n) = 1\right]\right|,\\ &= 0. \quad \square\end{aligned}$$

L'inconvénient de ce système, c'est que la clef est trop grosse. La solution naturelle consiste à remplacer cette grosse clef aléatoire par une clef plus petite et un générateur pseudo-aléatoire.

2.3 Chiffrement à base de générateur pseudo-aléatoire

Informellement, un PRG (Pseudo-Random Generator) est un algorithme qui, à partir d'une courte chaine de bits aléatoire, sa *graine* (« seed »), est capable de produire un nombre plus grand de bits pseudo-aléatoires, c.a.d. impossibles à distinguer efficacement de bits « vraiment » aléatoires.

Les PRG sont des outils indispensables en cryptographie. Pour ne donner qu'un exemple, la génération d'une paire de clefs RSA de n bits (en trouvant les deux nombres premiers par rejection sampling) nécessite environ $3n^2/2$ bits aléatoire, ce qui donne plus de 6 millions pour une clef de 2048 bits. En fait, on utilise systématiquement un PRG initialisé à partir d'une graine de 128 ou 256 bits.

Exemples répandus La plupart des langages de programmation comportent des fonctions qui permettent d'obtenir des séquences de nombre réputés pseudo-aléatoires. Par exemple, les normes qui spécifient le langage C imposent la présence de la fonction `mrand48`, qui renvoie le prochain entier de 32 bits de la séquence (et met à jour son état interne).

```
uint64_t rand48_state;

void srand48(uint32_t seed) {
  rand48_state = seed;
  rand48_state = 0x330e + (rand48_state << 16);
}

uint32_t mrand48() {
```

```
  rand48_state = (0x00000005deece66d * rand48_state + 11) & 0x0000ffffffffffff;
  return (rand48_state >> 16);
}
```

Python et PHP en utilisent un autre, le « Mersenne Twister ». Malheureusement, aucun de ces PRGs ne sont cryptographiquement sûrs. Par exemple, après avoir observé deux fois la sortie de `mrand48()`, il est aisé de produire *soi-même* toutes les prochaines valeurs, sans accéder à la variable d'état interne `rand48_state` (on laisse au lecteur le soin de trouver comment). Il y a donc un algorithme qui, à partir d'un préfixe de la séquence de bits pseudo-aléatoire, est capable de *prédire* la suite de la séquence — avec probabilité 1.

Un tel phénomène de prédiction est impossible avec une séquence de bits « vraiment aléatoires ». C'est quasiment vrai par définition : quelque chose d'aléatoire est imprédictible. Essayons de définir précisément ce qu'est un PRG, et montrons qu'on peut s'en servir pour améliorer le One-Time Pad.

Définition d'un PRG Généralement, lorsqu'on parle *d'ensemble probabiliste*, on parle en fait d'ensembles de variables aléatoires $\{X_n\}_{n\in\mathbb{N}}$ à valeur dans $\{0,1\}^*$ et telles que $|X_n| = \text{poly}(n)$.

Définition 4 : Indistinguabilité calculatoire

> Deux ensembles probabilistes $\{X_n\}_{n\in\mathbb{N}}$ et $\{Y_n\}_{n\in\mathbb{N}}$ sont **calculatoirement indistinguables** si pour tout algorithme polynomial randomisé, tout polynôme positif $p(\cdot)$ et tous n suffisamment grand, on a :
>
> $$\left|\mathbb{P}\left[\mathcal{A}(1^n, X_n) = 1\right] - \mathbb{P}\left[\mathcal{A}(1^n, Y_n) = 1\right]\right| < \frac{1}{p(n)}.$$
>
> La grandeur dans le membre gauche est l'avantage du distingueur $\mathcal{A}$.

On laisse au lecteur le soin de formuler une définition alternative avec un jeu où b est choisi « à l'avance » d'une part, et où b est choisi aléatoirement par le challengeur d'autre part.

Par exemple, l'hypothèse Diffie-Hellman Décisionnelle (DDH) pour un groupe $\langle g\rangle$ d'ordre $q_n > 2^n$, affirme que les ensembles $\left\{\left(g^{A_n}, g^{B_n}, g^{A_nB_n}\right)\right\}_n$ et $\left\{\left(g^{A_n}, g^{B_n}, g^{C_n}\right)\right\}$ sont calculatoirement indistinguables (A_n, B_n et C_n sont trois variables aléatoires uniformément distribuées dans $\mathbb{Z}_{q_n}$).

Définition 5 : Ensemble pseudo-aléatoire

> Un ensemble probabiliste (de chaine de bits) $\{X_n\}_{n\in\mathbb{N}}$ est **pseudo-aléatoire** s'il est calculatoirement indistinguable de $\{U_{|X_n|}\}_{n\in\mathbb{N}}$.

Définition 6 : Générateur pseudo-aléatoire avec taille arbitraire

> Un **générateur pseudo-aléatoire** est un algorithme déterministe polynomial G qui satisfait les deux conditions suivantes :
>
> 1. *Sortie arbitrairement grande* : pour toute graine $s \in \{0,1\}^*$ et tout $t \in \mathbb{N}$, on a $|G(s, 1^t)| = t$ et la chaine de bits $G(s, 1^t)$ est un préfixe de $G(s, 1^{t+1})$.
> 2. *Pseudo-aléatoire* : pour tout polynôme p, l'ensemble $\{G(U_n, 1^{p(n)})\}_{n\in\mathbb{N}}$ est pseudo-aléatoire.

Il faut noter que les PRGs les plus courants dans les langages de programmation, et en particulier la fonction `mrand48` montrée ci-dessus, ne satisfont *PAS* cette définition. Tout d'abord, la taille de leur graine est fixée, ce qui ne rentre pas dans le cadre de la définition. Ensuite, il existe des algorithmes efficaces qui distinguent leur sortie de bits uniformément aléatoires.

Il est possible de construire un PRG sûr sous l'hypothèse que le problème du logarithme discret est difficile ou que la factorisation des grands entiers est difficile. Ces construction sont (extrêmement) rarement utilisées en pratique. On verra plus loin des exemples de PRG cryptographiques raisonnables.

Chiffrement à partir de PRG Étant donné un PRG $\mathcal{G}$, on peut construire un mécanisme de chiffrement (G, E, D) qui *i*) a des clefs courtes et *ii*) possède des chiffrés indistinguables. Pour cela, G est un algorithme qui renvoie n bits aléatoires et on pose $E(k, \alpha) := \alpha \oplus \mathcal{G}(k, 1^{|\alpha|})$. Le déchiffrement se fait avec $D(k, \beta) := \alpha \oplus \mathcal{G}(k, 1^{|\alpha|})$.

Théorème 4. *Si $\mathcal{G}$ est un PRG, alors le mécanisme de chiffrement (G, E, D) défini ci-dessus possède des chiffrés indistinguables.*

Démonstration

> On effectue une réduction pour démontrer la contraposée : on suppose qu'il existe un adversaire polynomial randomisé qui « casse » le chiffrement, et on en déduit qu'il existe un autre adversaire qui « casse » le PRG.

On utilise une technique de preuve standard lorsque des constructions pseudo-aléatoires sont utilisées : on démontre dans un premier temps que si le mécanisme « pseudo-aléatoire » est remplacé par un mécanisme « vraiment aléatoire », alors la sécurité est garantie. On démontre ensuite que si la sécurité est violée, c'est qu'il est possible de distinguer le pseudo-aléa du « vrai aléa ». Comme c'est censé être impossible, alors la sécurité est bien garantie.

Supposons donc que $\mathcal{A}$ soit un algorithme qui contredise le fait que (G, E, D) possède des chiffrés indistinguables. Admettons que $\mathcal{A}$ joue à la version « bit-guessing ».

On forme un autre adversaire $\mathcal{D}$ qui tente cette fois de distinguer $\mathcal{G}(U_n, 1^t)$ de U_t. Voici comment fonctionne $\mathcal{D}$:

- $\mathcal{D}$ reçoit une chaine de bits α de taille t.
- $\mathcal{D}$ choisit aléatoirement un bit b.
- $\mathcal{D}$ joue au jeu d'indistinguabilité des chiffrés (version « bit-guessing ») avec $\mathcal{A}$, en simulant le challenger. Lorsque $\mathcal{A}$ envoie (m_0, m_1), alors $\mathcal{B}$ calcule $c \leftarrow m_b \oplus \alpha$ et renvoie c à $\mathcal{A}$.
- $\mathcal{A}$ calcule $\hat{b}$.
- Si $\mathcal{A}$ devine b correctement (c.a.d. si $\hat{b} = b$), alors $\mathcal{D}$ renvoie 1, sinon $\mathcal{D}$ renvoie 0.

Par définition, l'avantage du distingueur $\mathcal{D}$ est :

$$\mathsf{AdvPRG}[\mathcal{G}, \mathcal{D}] = \left| \mathbb{P}\left[\mathcal{D}(\mathcal{G}(U_n, 1^t)) = 1\right] - \mathbb{P}\left[\mathcal{D}(U_t) = 1\right] \right|.$$

Lorsque $\mathcal{D}$ reçoit U_t (une chaine de t bits uniforméments aléatoires), alors le mécanisme de chiffrement présenté à $\mathcal{A}$ est en fait le one-time pad. Il découle de la sécurité du one-time-pad (théorème 3), et du théorème 1 que $\mathcal{A}$ possède un avantage nul. Il devine donc correctement b avec probabilité $1/2$. Il s'ensuit que dans ce cas-là, $\mathcal{D}$ renvoie 1 avec probabilité $1/2$. Ceci prouve donc que $\mathbb{P}\left[\mathcal{D}(U_t) = 1\right] = 1/2$.

Lorsque $\mathcal{D}$ reçoit $\mathcal{G}(U_n, 1^t)$ (t bits produits par le PRG), alors $\mathcal{D}$ fait jouer « honnêtement » l'algorithme $\mathcal{A}$ au jeu de distinguabilité des chiffrés de $\mathcal{E}$ pour lequel il est supposé avoir un avantage non-négligeable. Dans ce cas, il existe donc un polynôme à valeurs positives p, tel que pour une infinité de n on a :

$$\mathsf{AdvIND*}[\mathcal{E}, \mathcal{A}] = 2\left(\mathbb{P}\left[\hat{b} = b\right] - \frac{1}{2}\right) > \frac{1}{p(n)}$$

Par conséquent, il existe une infinité de n pour lesquels le distingueur $\mathcal{D}$ renvoie 1 (quand $\hat{b} = b$) avec probabilité supérieure à $\frac{1}{2} + \frac{1}{2p(n)}$.

Il en découle qu'il existe un polynôme p et une infinité de n tels que :

$$\mathsf{AdvPRG}[\mathcal{G}, \mathcal{D}] > \frac{1}{2p(n)}.$$

L'avantage du distingueur $\mathcal{D}$ pour distinguer la sortie du PRG $\mathcal{G}$ de bits aléatoire est donc non-négligeable, et le théorème est prouvé. □

2.4 Sécurité sous attaques à clairs choisis

Pouvons-nous nous satisfaire d'un chiffrement qui a des chiffrés indistinguables ? La réponse est « ça dépend ».

Pour reprendre le scénario du vote, tout va bien s'il n'y a *qu'une seule* consultation électorale. En effet, regardons ce qui se passe s'il y en a *deux*. Les électeurs utilisent le One-Time Pad, qui possède bien des chiffrés indistinguables. L'electeur i vote deux fois, une fois $\texttt{vote}_i$ et une fois $\texttt{vote}'_i$. Les deux votes chiffrés sont $c_i = k_i \oplus \texttt{vote}_i$ et $c'_i = k_i \oplus \texttt{vote}'_i$. Un adversaire qui capte les deux peut calculer $\Delta_i \leftarrow c_i \oplus c'_i = \texttt{vote}_i \oplus \texttt{vote}'_i$. Si l'électeur vote deux fois la même chose aux deux référendums, alors $\Delta_i = 0$. Sinon, on a $\Delta_i \neq 0$.

Par conséquent, le mécanisme de chiffrement laisse fuir de l'information lorsqu'il est utilisé plusieurs fois avec la même clef, ce qui n'est pas satisfaisant. Que s'est-il passé ? En fait, la notion de sécurité considérée est *trop faible*.

On pourrait la renforcer en disant que l'adversaire reçoit *plusieurs* chiffrés (un nombre polynomial) produits avec la même clef, et qu'il doit décider s'il s'agit des chiffrés de $(x_1, x_2, \dots)$ ou bien de $(y_1, y_2, \dots)$. On verrait alors que les mécanismes de chiffrement discutés jusqu'ici ne satisfont pas cette notion « d'indistinguabilité des chiffrés avec plusieurs messages ».

Mais, tant qu'à faire, on peut encore la renforcer en permettant à l'adversaire de *choisir* les messages clairs dont il reçoit les chiffrés. Voici un scénario (presque) réaliste : Alice chiffre tous les fichiers sur le disque dur de son ordinateur portable avec une clef symétrique k, connue d'elle seule. Bob lui envoie plusieurs emails, avec des pièces-jointes soigneusement forgées. Le client mail d'Alice stocke, en les chiffrant, les pièces-jointes sur le disque dur. Bob vole l'ordinateur portable d'Alice, et observe le chiffrement des emails qu'il a lui-même envoyé. Il apprend alors les chiffrés de messages qu'il a choisi.

Définition de la sécurité sémantique sous attaques à clairs choisis Pour formaliser une notion de sécurité complexe comme celle-ci, on utilise un nouveau « jeu d'attaque ». Dans ce jeu, l'adversaire doit avoir accès au mécanisme de chiffrement (sous une clef inconnue) pour pouvoir obtenir les chiffrés de ses clairs choisis. On définit donc formellement la notion de machine à oracle. Intuitivement, c'est un algorithme qui a la possibilité d'envoyer des chaines de bits à un mécanisme externe (complètement opaque), et d'en recevoir des chaines de bits en réponse. On peut imaginer un algorithme qui a accès à un web-service auquel il peut envoyer des requêtes et en recevoir des réponses. L'algorithme peut donc *invoquer* le mécanisme externe, mais pas examiner ce qui se passe à l'intérieur. Le mécanisme externe peut donc contenir des secrets (tels que des clefs de chiffrement).

Définition 7 : Machine de Turing avec Oracle

> Une **machine de Turing avec Oracle** est une machine de Turing munie d'une bande spéciale (la « bande de l'oracle ») et de deux états spéciaux nommés invocation oracle et résultat oracle. Le calcul réalisé par une machine à oracle M ayant accès à un **oracle** $f : \{0,1\}^* \to \{0,1\}^*$ est défini par la relation qui donne la configuration suivante. Lorsque l'état courant n'est pas l'un des états spéciaux, tout se passe comme dans une machine de Turing normale. Soit γ une configuration où l'état courant est invocation oracle et où le contenu de la bande de l'oracle est la séquence de symboles q. Alors la configuration qui suit γ est indentique à γ, excepté que l'état est résultat oracle et que le contenu de la bande de l'oracle est remplacé par $f(q)$. On dit que q est la **requête** de M et que $f(q)$ est la **réponse** de l'oracle f. Si la bande d'entrée de M contenait x et que M s'arrête dans un un état acceptant en ayant accès à l'oracle f, alors le contenu de la bande de sortie de M est noté $M^f(x)$.

Ceci étant posé, on peut décrire le jeu qui définit l'indistinguabilité des chiffrés sous attaques à clairs choisis.

Jeu 4 (sécurité sémantique)

> Pour un mécanisme de chiffrement $\mathcal{E} = (G, E, D)$ et étant donné un adversaire représenté par deux machines polynomiales probabilistes $\mathcal{A}_1$ et $\mathcal{A}_2$ qui auront accès à un oracle de chiffrement. L'**expérience** $b \in \{0,1\}$ est définie par :
>
> 1. Le Challenger génère une clef symétrique : $k \leftarrow G(1^n)$.
> 2. L'Adversaire choisit deux messages (de la même taille) : $m_0, m_1, \sigma \leftarrow \mathcal{A}_1^{E_k}(1^n)$.
> 3. Le Challenger calcule $c \leftarrow E_k(m_b)$.
> 4. L'adversaire produit un bit $\hat{b} \leftarrow \mathcal{A}_2^{E_k}(\sigma, c)$
>
> On note W_b l'évènement « $\mathcal{A}$ renvoie 1 dans l'expérience b ». On définit l'**avantage** de $\mathcal{A}$ par :
>
> $$\mathbf{AdvCPA}[\mathcal{E}, \mathcal{A}] = \big|\mathbb{P}[W_0] - \mathbb{P}[W_1]\big|.$$

La seule différence avec le jeu 1, c'est que l'adversaire a en permanence accès à l'oracle de chiffrement pour la clef k. On pourrait définir de manière identique une version « bit-guessing » où b est choisi aléatoirement par le challenger, et définir alors

$$\mathbf{AdvCPA*}[\mathcal{E}, \mathcal{A}] = \left| \mathbb{P}\left[\hat{b} = b\right] - \frac{1}{2} \right|.$$

On trouverait fatalement que $\mathbf{AdvCPA*}[\mathcal{E}, \mathcal{A}] = \frac{1}{2}\mathbf{AdvCPA}[\mathcal{E}, \mathcal{A}]$.

Définition 8 : Sécurité sémantique sous attaque à clairs choisis

> Un mécanisme de chiffrement $\mathcal{E}$ est **sémantiquement sûr sous attaques à clair choisis** (en abrégé IND-CPA) si l'avantage de tout adversaire polynomial au jeu 4 est négligeable.

Une première constatation est qu'un mécanisme de chiffrement *déterministe* (c.a.d. où E est déterministe) ne peut pas satisfaire cette définition. La suite de ce chapitre montre plusieurs manières de construire des chiffrement IND-CPA, ce qui nécessite de faire intervenir de nouvelles primitives cryptographiques.

2.5 Le bestiaire de la cryptographie symétrique

On a déjà vu les *générateurs pseudo-aléatoires*. On va voire maintenant les *fonctions* (resp. *permutations*) *pseudo-aléatoires*. Les PRF sont des fonctions qui ne peuvent pas être distinguées de fonctions « vraiment » aléatoires par un algorithme efficace qui aurait la possibilité d'évaluer la fonction sur les valeurs de son choix.

2.5.1 Fonctions aléatoires

Une fonction aléatoire $\{0,1\}^n \to \{0,1\}^n$ est une fonction choisie uniformément au hasard parmi l'ensemble des 2^{n2^n} fonctions qui envoient n bits sur n bits. De manière équivalente, c'est une fonction où pour toute entrée $x \in \{0,1\}^n$, la sortie $f(x)$ est choisie indépendamment et uniformément au hasard dans $\{0,1\}^n$.

Une fonction aléatoire est un objet puissant. Par exemple, si $\phi : \{0,1\}^n \to \{0,1\}^n$ est une fonction aléatoire, alors $\phi(x)$ est uniformément distribué, pour tout $x \in \{0,1\}^n$.

Démonstration

> Fixons $x_0 \in \{0,1\}^n$. Pour chaque $y_0 \in \{0,1\}^n$ il y a exactement $2^{(n-1)2^n}$ fonctions $f : \{0,1\}^n \to \{0,1\}^n$ telles que $f(x_0) = y_0$. Par conséquent, si ϕ est une fonction choisie uniformément au hasard, alors $\mathbb{P}\left[\phi(x_0) = y_0\right] = 2^{-n}$.

Un argument similaire permettrait de conclure que pour tous $x \neq x'$, les variables aléatoires $\phi(x)$ et $\phi(x')$ sont *indépendantes* et uniformément distribuées. On en conclut que ϕ contient $n2^n$ bits d'entropie (c.a.d. on peut extraire $n2^n$ bits uniformément aléatoires et indépendants de ϕ ; d'autre part ϕ est entièrement spécifiée par la donnée de ces $n2^n$ bits).

Définition 9 : Ensemble de fonctions

> Un **ensemble de fonctions** est une séquence de variables aléatoires $\{F_n\}_{n\in\mathbb{N}}$ où F_n est une fonction de $\{0,1\}^n \to \{0,1\}^n$. L'**ensemble de fonctions uniforme**, noté $\{H_n\}_{n\in\mathbb{N}}$ est tel que H_n est uniformément distribuée parmi les 2^{n2^n} fonctions qui envoient $\{0,1\}^n$ sur $\{0,1\}^n$.

Pour tout n, H_n est donc une « fonction aléatoire ».

2.5.2 Fonctions pseudo-aléatoires

Le caractère pseudo-aléatoire d'un ensemble de fonction est défini de manière analogue à celui d'une chaine de bits. C'est une fonction qu'on ne peut pas distinguer efficacement d'une fonction aléatoire, même si on peut l'évaluer arbitrairement.

Définition 10 : Ensemble de fonctions pseudo-aléatoire

> Un ensemble de fonctions $\{F_n\}_{n\in\mathbb{N}}$ est **pseudo-aléatoire** si pour toute machine M, polynomiale probabiliste avec oracle, tout polynôme p et tout n suffisamment grand :
>
> $$\left| \mathbb{P}\left[M^{F_n}(1^n) = 1\right] - \mathbb{P}\left[M^{H_n}(1^n) = 1\right] \right| < \frac{1}{p(n)}.$$

On pourrait également définir un jeu d'attaque : dans l'expérience $b = 0$, l'adversaire a accès à une fonction tirée aléatoirement dans F_n, tandis que dans l'expérience $b = 1$, il a accès à une fonction tirée aléatoirement

dans H_n. Il reste un point à préciser : pour être utile, une fonction pseudo-aléatoire doit pouvoir être calculée efficacement.

Définition 11 : Ensemble de fonctions efficacement calculable

Un ensemble de fonctions $\{F_n\}_{n\in\mathbb{N}}$ est **efficacement calculable** si les deux conditions suivantes sont réunies :

1. *Indexation efficace* : il existe un algorithme polynomial probabiliste I et une correspondance ϕ entre chaines de bits et fonctions tels que les variables aléatoires $\phi(I(1^n))$ et F_n sont identiquement distribuées.

   On note f_i la fonction associée à la chaine de bits i (c.a.d. $f_i = \phi(i)$).
2. *Évaluation efficace* : il existe un algorithme polynomial V tel que $V(i,x) = f_i(x)$ pour tout indice i appartenant à l'ensemble image de I et tout $x \in \{0,1\}^n$.

En gros, l'algorithme I produit des « indices » aléatoires (on dit parfois des « clefs ») pour la PRF. Une **fonction pseudo-aléatoire** (PRF) est finalement un ensemble de fonctions peudo-aléatoire et efficacement calculable. Autrement dit, aucun algorithme ne peut se comporter différemment en ayant accès à la PRF (sous un indice aléatoire) et en ayant accès à une fonction « vraiment aléatoire ».

Par exemple, la fonction $f_{\alpha,\beta}(x) := \alpha x + \beta$, avec $\alpha, \beta, x \in \mathbb{Z}_p$ n'est pas une PRF : il existe un distingueur qui fonctionne en trois requêtes avec avantage exponentiellement proche de 1. Autre exemple, la fonction $f_M(x) := M \cdot x$, où $x \in {\mathbb{F}_2}^n$ et où M est une matrice $n \times n$ à coefficients dans $\mathbb{F}_2$, n'est pas une PRF (il y a là aussi un distingueur en trois requêtes).

Pour démontrer l'utilité des PRFs, on remarque qu'à toutes fins utiles, pour toute séquence $(x_i)_{i\in\mathbb{N}}$ avec $x_i \in \{0,1\}^n$ et $x_i \neq x_j$ lorsque $i \neq j$, l'ensemble $\{V(I(1^i), x_i)\}_{i\in\mathbb{N}}$ est pseudo-aléatoire (si ce n'était pas le cas, on aurait un distingueur entre la PRF et une fonction aléatoire). Par conséquent, pour tout x, évaluer une PRF sur x renvoie une valeur qui peut être considérée comme uniformément aléatoire.

Comparées à des PRG, les PRF sont plus flexibles et plus commodes d'utilisation. Un PRG produit un « flux », séquentiel et de taille polynomiale, de bits pseudo-aléatoires. Si on dispose d'une PRF f, alors on peut en extraire $n2^n$ bits pseudo-aléatoires, et l'avantage c'est qu'on peut accéder efficacement à n'importe lequel sans devoir générer tous les autres.

2.5.3 Permutations (pseudo-)aléatoires

Les concepts définis ci-dessus pour les fonctions s'étendent naturellement aux fonctions particulières que sont les permutations.

Définition 12 : Ensemble de permutations

Un **ensemble de permutations** est une séquence de variables aléatoires $\{P_n\}_{n\in\mathbb{N}}$ où P_n est une permutation de $\{0,1\}^n$. L'**ensemble de permutation uniforme**, noté $\{K_n\}_{n\in\mathbb{N}}$ est tel que K_n est uniformément distribuée parmis les $2^n!$ permutations de $\{0,1\}^n$.

Pour tout n, K_n est donc une « permutation aléatoire ». La définition s'étend naturellement.

Définition 13 : Ensemble de permutations pseudo-aléatoire

Un ensemble de permutations $\{P_n\}_{n\in\mathbb{N}}$ est **pseudo-aléatoire** si pour toute machine polynomiale probabiliste avec oracle, tout polynôme p et tout n suffisamment grand :

$$\left|\mathbb{P}\left[M^{P_n}(1^n) = 1\right] - \mathbb{P}\left[M^{K_n}(1^n) = 1\right]\right| < \frac{1}{p(n)}.$$

Ceci dit, il y a une subtilité supplémentaire. Si f est une permutation, un distingueur potentiel doit pouvoir évaluer f. Mais on pourrait aussi lui donner, en plus, la possibilité d'évaluer f^{-1}.

Définition 14 : Ensemble de permutations fortement pseudo-aléatoire

Un ensemble de permutations $\{P_n\}_{n\in\mathbb{N}}$ est **fortement pseudo-aléatoire** si pour toute machine polynomiale probabiliste avec oracle, tout polynôme p et tout n suffisamment grand :

$$\left|\mathbb{P}\left[M^{P_n, P_n^{-1}}(1^n) = 1\right] - \mathbb{P}\left[M^{K_n, K_n^{-1}}(1^n) = 1\right]\right| < \frac{1}{p(n)}.$$

Enfin, il faut adapter la notion d'efficacité pour inclure la possibilité de l'évaluation de la réciproque.

Définition 15 : Ensemble de permutations efficacement calculable

Un ensemble de permutations $\{P_n\}_{n\in\mathbb{N}}$ est **efficacement calculable** si les deux conditions suivantes sont réunies :

1. *Indexation efficace* : il existe un algorithme polynomial probabiliste I et une correspondance ϕ entre chaines de bits et permutations tels que les variables aléatoires $\phi(I(1^n))$ et P_n sont identiquement distribuées.

   On note f_i la permutation associée à la chaine de bits i (c.a.d. $f_i = \phi(i)$).
2. *Évaluation efficace* : il existe deux algorithmes polynomiaux V et V^{-1} tel que $V(i,x) = f_i(x)$ et $V^{-1}(i,x) = f_i^{-1}(x)$ pour tout indice i appartenant à l'ensemble image de I et tout $x \in \{0,1\}^n$.

2.5.4 Systèmes de chiffrement par bloc

Les systèmes de chiffrement par bloc (block cipher) constituent incontestablement la « brique de base » cryptographique la plus répandue et dont l'humanité possède la meilleure compréhension.

En deux mots, un block cipher (G,E,D) est un mécanisme de chiffrement restreint à des entrées de taille fixe. En gros, il existe une fonction polynomialement bornée $\ell : \mathbb{N} \to \mathbb{N}$, la « taille du bloc », telle que la définition 1 fonctionne avec des messages appartenant à $\{0,1\}^{\ell(n)}$. La plupart du temps, on considère $\ell(n) = n$. Généralement, un block cipher est déterministe (l'algorithme de chiffrement E ne fait pas appel à une source externe d'aléa).

L'AES (Advanced Encryption Standard) est un block-cipher conçu par deux chercheurs Belges, Vincent Rijmen et Joan Daemen et à la fin des années 1990 et adopté par le gouvernement américain pour son propre usage. C'est un standard industriel *de facto* et sa sécurité n'a pas été mise en défaut après bientôt 20 ans. Il chiffre des blocs de 128 bits avec des clefs de 128, 192 ou 256 bits.

En deux mot, un block cipher est une famille de permutations de $\{0,1\}^n$ indistinguable de permutations aléatoires. Donc c'est une permutation fortement pseudo-aléatoire et efficacement calculable de $\{0,1\}^n$. En gros, ça chiffre n bits !

Théorème 5. *Indéfiniment, et jusqu'à nouvel ordre, l'AES est est un block-cipher sûr, c'est-à-dire une permutation fortement pseudo-aléatoire.*

Démonstration

Il suffit d'y croire. Et d'oublier le fait que le théorème n'a pas de sens, car l'AES n'est défini que pour des tailles de blocs et de clefs fixes, alors que la définition donne un sens asymptotique à la notion de sécurité, lorsque « n » tend vers $+\infty$...

2.6 Block-cipher IND-CPA à partir d'une fonction pseudo-aléatoire

On va voir comment réaliser un mécanisme de chiffrement IND-CPA, mais qui ne chiffre que des messages de n bits, à partir d'une PRF (I,V). C'est donc, techniquement, un « block cipher ». Pour générer une clef de chiffrement, on génère un indice de la PRF : $G(1^n) := I(1^n)$. Pour chiffrer un message $\alpha \in \{0,1\}^n$, on tire aléatoirement une chaine $r \in \{0,1\}^n$ et on pose $E_k(\alpha) := (r, \alpha \oplus V(k,r))$. Pour déchiffrer, on fait $D_k(r,\beta) := \beta \oplus V(k,r)$. (Quand on y réflechit bien, ceci est analogue au chiffrement Elgamal).

On note que ceci donne une sorte de block-cipher non-déterministe, ce qui n'est pas la norme ; c'est un mal nécessaire : aucun dispositif déterministe ne peut être IND-CPA.

Théorème 6. *Si (I,V) est une PRF, alors le block cipher ci-dessus est IND-CPA.*

Démonstration

La preuve fonctionne, comme d'habitude, par contraposée : on montre que si le chiffrement n'est pas IND-CPA, alors (I,V) n'est pas une PRF.

On utilise une stratégie de preuve en deux phases, habituelle dans les constructions à base de dispositifs pseudo-aléatoires. Phase 1 : on remplace la PRF par une « vraie » fonction aléatoire et on montre que le chiffrement est sûr. Phase 2 : par conséquent, si le chiffrement est cassé, c'est que la PRF ne se comporte pas comme une « vraie » fonction aléatoire.

C'est parti pour la phase 1. On commence par considérer une version modifiée du mécanisme de chiffrement, qu'on notera $\mathcal{E}'$, où la PRF est remplacée par une « vraie » fonction aléatoire ϕ (c.a.d. choisie uniformément au hasard parmi toutes les fonctions $\{0,1\}^n \to \{0,1\}^n$)

Claim : Pour tout adversaire $\mathcal{A}$ effectuant moins de t requêtes à l'oracle, on a $\mathbf{AdvCPA}[\mathcal{E}', \mathcal{A}] \leq \frac{(t+1)^2}{2^n}$.

Démonstration

Notons $x_0, \ldots, x_t$ les messages que l'adversaire envoie à son oracle de chiffrement, et $(\hat{r}, m_b \oplus \phi(\hat{r}))$ le chiffré envoyé par le challenger.

De ses requêtes à l'oracle de chiffrement, l'adversaire obtient les t chiffrés $\big(r_i, x_i \oplus \phi(r_i)\big)_{1 \leq i \leq t}$. Notons X l'évènement « les $t+1$ valeurs $r_1, \ldots, r_t, \hat{r}$ sont toutes distinctes ». Parce que les r_i sont des variables aléatoires indépendantes et uniformément distribuées, on trouve que $\mathbb{P}[X] \geq 1 - (t+1)^2 2^{-n}$. Si tel est le cas, les $t+1$ valeurs $x_i \oplus \phi(r_i)$, de même que $m_b \oplus \phi(\hat{r})$ sont indépendantes et uniformément distribuées (comprendre : c'est du bruit pur). Ceci est vrai indépendamment des valeurs des x_i et de m_b.

Tant que l'évènement X est réalisé, les chaines de bits reçues par l'adversaire suivent exactement la même distribution (uniforme) dans les deux expériences $b = 0$ et $b = 1$. Notons W'_b l'évènement « l'adversaire renvoie 1 dans l'expérieure b avec $\mathcal{E}'$ ». On en déduit que $\mathbb{P}[W'_0 \mid X] = \mathbb{P}[W'_1 \mid X]$, donc que l'adversaire a un avantage nul si X est réalisé.

En renonçant à contrôler ce qui se passe si X n'est pas réalisé, on obtient :

$$\begin{aligned}
\mathbf{AdvCPA}[\mathcal{E}', \mathcal{A}] &= \big|\mathbb{P}[W'_0] - \mathbb{P}[W'_1]\big| \\
&= \Big|\mathbb{P}[W'_0 \mid X]\,\mathbb{P}[X] + \mathbb{P}[W'_0 \mid \neg X]\,\mathbb{P}[\neg X] - \mathbb{P}[W'_1 \mid X]\,\mathbb{P}[X] - \mathbb{P}[W'_1 \mid \neg X]\,\mathbb{P}[\neg X]\Big| \\
&\leq \underbrace{\Big|\mathbb{P}[W'_0 \mid X] - \mathbb{P}[W'_1 \mid X]\Big|}_{=0}\,\mathbb{P}[X] + \underbrace{\Big|\mathbb{P}[W'_0 \mid \neg X] - \mathbb{P}[W'_1 \mid \neg X]\Big|}_{\leq 1}\,\mathbb{P}[\neg X] \\
&\leq (t+1)^2 \cdot 2^{-n}.
\end{aligned}$$

On en conclut que n'importe quel adversaire ne possède qu'un avantage négligeable (exponentiellement faible) contre $\mathcal{E}'$, car son temps d'exécution polynomial impose $t = \mathrm{poly}(n)$.

On peut maintenant passer à la phase 2. Le résultat ci-dessus va faire fonctionner l'argument suivant : si un adversaire contre le chiffrement a un avantage non-negligeable, c'est qu'il distingue la PRF de ϕ.

On procède par réduction. Supposons qu'un adversaire $\mathcal{A}$ obtienne un avantage non-négligeable au jeu qui définit la sécurité sémantique à clair choisi, et servons-en nous pour construire un adversaire $\mathcal{B}$ qui distingue la PRF d'une fonction aléatoire avec un avantage non-négligeable.

On construit donc cet adversaire $\mathcal{B}$, qui a accès à un oracle réalisant une fonction F. $\mathcal{B}$ choisit au hasard un bit b puis simule le challenger du jeu qui définit la sécurité sémantique. $\mathcal{B}$ simule l'oracle de chiffrement de l'adversaire $\mathcal{A}$ de la façon suivante : pour chiffrer x, $\mathcal{B}$ choisit uniformément au hasard une chaine de n bits $r \in \{0,1\}^n$, soumet r à son F-oracle, et renvoie $(r, x \oplus F(r))$ à $\mathcal{A}$. À la fin, $\mathcal{B}$ récupère le bit $\hat{b}$ de l'adversaire et renvoie 1 si $b = \hat{b}$.

On prétend que $\mathcal{B}$ brise la propriété de pseudo-aléa de la PRF. L'argument essentiel est que lorsque l'oracle auquel $\mathcal{B}$ a accès est en réalité une fonction aléatoire, l'adversaire $\mathcal{A}$ est confronté au chiffrement $\mathcal{E}'$.

Pour formaliser cela, on considère les évènements W_b (resp. W'_b) : « l'adversaire $\mathcal{A}$ répond 1 dans l'expérience b et la fonction F est F_n [la PRF] (resp. F est H_n [fonction aléatoire]) ». Comme le bit b est choisi aléatoirement, on a :

$$\begin{aligned}
\mathbb{P}\big[\mathcal{B}^{F_n} = 1\big] &= \frac{1}{2}\Big(\mathbb{P}[\mathcal{A} \text{ renvoie } 1 \mid b = 1] + \mathbb{P}[\mathcal{A} \text{ renvoie } 0 \mid b = 0]\Big) = \frac{1}{2} + \frac{W_1 - W_0}{2} \\
\mathbb{P}\big[\mathcal{B}^{H_n} = 1\big] &= \frac{1}{2} + \frac{W'_1 - W'_0}{2} \leq \frac{1}{2} + \frac{(t+1)^2}{2^{n+1}}
\end{aligned}$$

On a alors :

$$\begin{aligned}
\Big|\mathbb{P}\big[\mathcal{B}^{Fn} = 1\big] - \mathbb{P}\big[B^{H_n} = 1\big]\Big| &= \frac{1}{2}\big|(W_1 - W_0) - (W'_1 - W'_0)\big| \\
&\geq \frac{1}{2}\Big|\big|W_1 - W_0\big| - \big|W'_1 - W'_0\big|\Big| \\
&\geq \frac{1}{2}\Big|\mathbf{AdvCPA}[\mathcal{E}, \mathcal{A}] - \mathbf{AdvCPA}[\mathcal{E}', \mathcal{A}]\Big|.
\end{aligned}$$

Ceci permet de conclure la preuve : comme $\mathbf{AdvCPA}[\mathcal{E}', \mathcal{A}]$ est exponentiellement faible, et qu'il y a un polynôme p tel que pour une infinité de n, $\mathbf{AdvCPA}[\mathcal{E}, \mathcal{A}] > 1/p(n)$, alors on conclut que pour une infinité de n, l'avantage de $\mathcal{B}$ est supérieur à $1/(3p(n))$ (ceci est une majoration brutale).

2.7 Chiffrement IND-CPA à base d'une PRF

La construction ci-dessus obtient donc un bon niveau de sécurité, mais elle est peu pratique : la taille des clairs est restreinte à n bits, et les chiffrés sont deux fois plus gros que les clairs.

Il serait possible de réparer notre construction à base d'un PRG (cf. § 2.3) pour la rendre résistante aux attaques à clairs choisis. Étant donné un PRG $\mathcal{G}$, on pourrait tenter de définir un mécanisme de chiffrement (G, E, D) de la façon suivante. $G(1^n)$ renvoie n bits aléatoires. Pour chiffrer α, on choisit uniformément au hasard $r \in \{0,1\}^n$ et on pose $E_k(\alpha) := (r, \alpha \oplus \mathcal{G}(k\|r, 1^{|\alpha|}))$. Le déchiffrement se fait simplement $D_k(r, \beta) := \beta \oplus \mathcal{G}(k\|r, 1^{|\beta|})$.

Le problème, c'est que pour obtenir la sécrité IND-CPA il faudrait avoir des garanties sur le comportement de $\mathcal{G}$ sous des *clefs liées*. Plus précisément, il faudrait que $\mathcal{G}(k\|r_0)$ et $\mathcal{G}(k\|r_1)$ soient indistinguables. Or rien dans la définition des PRG ne garantit ceci. En pratique, les stream ciphers sont des PRG à deux entrées (une pour k, une pour r) qui offrent cette garantie.

Mais on peut faire marcher cette idée sans trop d'efforts supplémentaires, en partant non pas d'un PRG mais d'une PRF.

Tout d'abord, on note qu'on peut produire un PRG à partir d'une PRF : c'est le « mode compteur ». Il suffit de poser $\mathcal{G}(s, 1^{\ell n}) = F_s(0)\|F_s(1)\|\ldots\|F_s(\ell)$. Dans cette construction, rien n'oblige à faire commencer le « compteur » à zéro. Au contraire, on peut ajouter un argument « vecteur d'initialisation » de n-bit à $\mathcal{G}$ pour cela, en posant $\mathcal{G}(s, IV, 1^{\ell n}) = F_s(IV)\|F_s(IV+1)\|\ldots\|F_s(IV+\ell)$ (il est entendu que l'addition se fait modulo 2^n). Du coup, ceci permet de modifier le mécanisme de chiffrement ci-dessus en $E'_k(\alpha) := (IV, \alpha \oplus \mathcal{G}(k, IV, 1^{|\alpha|}))$ et $D'_k(IV, \beta) := \beta \oplus \mathcal{G}(k, IV, 1^{|\beta|})$, où IV est choisi aléatoirement lors du chiffrement.

Théorème 7. *Si F est une PRF, alors la construction précédente (le « mode compteur randomizé ») est un chiffrement IND-CPA.*

Démonstration

C'est grosso-modo la même preuve que dans le théorème précédemment démontré. On refait marcher l'argument en deux phase, et ce qui change ici, c'est la phase 1.

On considère donc encore le chiffremenr $\mathcal{E}'$ où la PRF est remplacée par une fonction aléatoire tirée uniformément.

Claim : pour tout adversaire $\mathcal{A}$, $\mathbf{AdvCPA}[\mathcal{E}', \mathcal{A}] = \mathrm{negl}(n)$.

Démonstration

Comme précédemment, l'adversaire soumet des clairs $m_0, \ldots, m_{t-1}$ de son choix à l'oracle de chiffrement, et le challengeur effectue le chiffrement de $\hat{m} := m_b$. À chaque fois, un IV aléatoire est choisi (on les note $IV_0, \ldots, IV_{t-1}$) et $\hat{IV}$ est choisi pour c. Lors du chiffrement de m_i, la fonction aléatoire est évaluée sur les entrées $IV_i, IV_i + 1, \ldots, IV_i + \lceil |m_i|/n \rceil$.

Tant que toutes les entrées de ϕ sont toutes distinctes, le chiffrement agit comme un One-Time Pad et l'avantage de l'adversaire est nul. Il reste à évaluer la probabilité d'une « collision » entre les entrées de ϕ. Notons ℓ une borne supérieure sur la taille des messages chiffrés (exprimée en nombre de blocs de n bits). On note que si IV_i et IV_j sont distants d'au moins 2ℓ, alors il ne peut pas y avoir de collision entre $IV_i + k$ et $IV_j + k'$ avec $0 \leq k, k' < \ell$. Dit autrement, pour toute paire de IV_i, IV_j choisis aléatoirement, la probabilité qu'il existe $0 \leq k, k' < \ell$ tels que $IV_i + k = IV_j + k'$ est inférieure à $2\ell/2^n$. On obtient une majoration brutale en sommant sur toutes les paires possibles :

$$\mathbb{P}[\mathsf{BAD}] \leq 2t^2\ell/2^n.$$

Comme t et ℓ sont tous les deux majorés par le temps d'exécution de $\mathcal{A}$, qui est polynomial, ceci est négligeable.

La suite de la preuve est identique à celle du théorème 6.

2.8 Chiffrement IND-CPA à partir de l'AES

Venons-en à notre objectif majeur : montrer qu'on peut realiser un chiffrement IND-CPA partant de l'AES. Pour faire marcher la construction précédente, il nous faudrait une PRF, or l'AES est (a priori) une PRP.

Il y a plusieurs moyens de transformer un block cipher en PRF. Par exemple la construction de Davies-Meyer, $F_k(x) = E_k(x) \oplus x$, ou bien encore $F_{k,k'}(x) = E_k(x) \oplus E_{k'}(x)$. Mais en réalité, il y a plus simple : une PRF et une PRP sont indistinguables pour des adversaires

polynomialement bornés, comme on va le voir plus bas. Par conséquent, on obtient un mode de chiffrement IND-CPA, le « mode compteur randomizé », en remplaçant la PRF de la construction précédente par l'AES.

Pour démontrer cela (le « PRF switching lemma »), considérons un nouveau jeu d'attaque.

Pour $b = 0, 1$, on définit l'expérience b :

1. Le Challenger choisit aléatoirement f. Si $b = 0$, alors f est une permutation aléatoire de $\{0,1\}^n$. Si $b = 1$, alors f est une fonction aléatoire de $\{0,1\}^n \to \{0,1\}^n$.
2. L'adversaire soumet au Challenger une séquence de requêtes $x_1, x_2, \ldots$, où chacun des x_i est une chaine de n bits. Le Challenger calcule $y_i \leftarrow f(x_i)$ et renvoie y_i à l'adversaire.
3. L'adversaire renvoie un bit $\hat{b}$.

On définit l'évènement W_b par « l'adversaire renvoie 1 dans l'expérience b ». L'avantage de l'adversaire est :

$$\mathbf{PFAdv}[\mathcal{A}] := \Big|\mathbb{P}\left[W_0\right] - \mathbb{P}\left[W_1\right]\Big|.$$

Théorème 8. *Soit $\mathcal{A}$ un adversaire qui joue au jeu-ci-dessus en faisant moins de Q requêtes. Alors* $\mathbf{PFAdv}[\mathcal{A}] \leq Q^2/2^n$.

Démonstration

Si l'adversaire fait Q requêtes au maximum, alors l'expérience 0 (avec la permutation) du jeu décrit ci-dessus peut être réalisé par le Challenger de la façon suivante :

1. Choisir $z_1, \ldots, z_Q$ uniformément au hasard dans $\{0,1\}^n$.
2. Lors de la réception de la i-ème requête x_i venant de $\mathcal{A}$, tester si $x_i = x_j$ avec $j < i$.
   (a) Si oui (requête déjà vue avant), poser $y_i \leftarrow y_j$.
   (b) Si non :
      i. poser $y_i \leftarrow z_i$.
      ii. Si $y_i \in \{y_1, \ldots, y_{i-1}\}$, alors remplacer y_i par un élément choisi aléatoirement dans $\{0,1\}^n \setminus \{y_1, \ldots, y_{i-1}\}$.

En effet, la condition 2.b.*ii* garantit que le Challenger simule bel et bien une permutation aléatoire, en éliminant la possibilité que $f(x_i) = f(x_j)$ avec $x_i \neq x_j$.

Maintenant, observons que l'expérience 1 du jeu (avec la fonction) est obtenue en *retirant* l'étape 2b.*ii*. Dans ce cas-là, le Challenger simule une fonction aléatoire.

Définissons l'évènement Z : « il existe $i \neq j$ tels que $z_i = z_j$ ». Si l'évènement Z ne se produit pas, alors les z_i sont tous différents. Ceci implique que le « remplacement » prévu dans l'étape 2b.*ii* n'a jamais lieu, et donc que la présence ou l'absence de cette vérification n'influe pas sur le résultat de l'expérience.

Imaginons qu'on exécute les deux expériences $b = 0$ et $b = 1$ dans le même « espace probabiliste » (c.a.d. que l'adversaire et le Challenger font les mêmes choix aléatoires). Si Z n'est pas réalisé, le challengeur est *exactement le même* dans les deux expériences, et, soumis aux mêmes requêtes de l'adversaire, il va renvoyer *mêmes* réponses. Par conséquent, l'adversaire va renvoyer le *même* bit dans les deux expériences — avec probabilité 1.

On en conclut que $\mathbf{PFAdv}[\mathcal{A}] \leq \mathbb{P}\left[\neg Z\right]$. Comme $\mathbb{P}\left[z_i = z_j\right] = 2^{-n}$, et qu'il y a moins de Q^2 paires (z_i, z_j), on obtient bien le résultat annoncé. □

Examinons la conséquence de ce thèorème : l'AES est indistinguable d'une permutation aléatoire ; une permutation aléatoire est indistinguable d'une fonction aléatoire. Donc l'AES est indistinguable d'une fonction aléatoire, et c'est bien une PRF !

Chapitre 3

Hachage et intégrité

Dans ce chapitre, on définit précisément la notion de « chiffrement authentifié » et on montre comment on peut en construire.

3.1 Notions d'intégrité et chiffrement authentifié

On rappelle que le mécanisme de déchiffrement peut produire une sortie spéciale $\perp$ lorsqu'on lui soumet un chiffré invalide. Pour garantir l'intégrité des communications chiffrées, on utilise une idée assez simple : on souhaite que le seul moyen de produire un chiffré valide (sous une clef k) consiste à utiliser le mécanisme de chiffrement (avec la même clef k).

Du coup, si on reçoit un message valide, c'est qu'il a été produit par quelqu'un qui connaît la clef secrète, et qu'il n'a pas été modifié pendant le transport. Il faut noter que dans le cas du chiffrement à clef publique, tout le monde peut générer des chiffrés valides, donc l'expéditeur n'est pas identifié de manière irrévocable.

Pour définir tout ceci, on considère un « jeu d'attaque ».

Jeu 5 (intégrité des chiffrés)

> Étant donné un mécanisme de chiffrement $\mathcal{E} = (G, E, D)$ et un adversaire $\mathcal{A}$, le jeu se déroule de la façon suivante :
>
> 1. Le challenger génère une clef aléatoire pour $\mathcal{E}$ en faisant $k \leftarrow G(1^n)$.
> 2. L'adversaire soumet au challenger des requêtes de chiffrement. Pour $i = 1, 2, \ldots$, la i-ème requête est composée d'une chaine de bits m_i. Le challenger calcule $c_i \leftarrow E(k, m_i)$ et renvoie c_i à l'adversaire.
> 3. L'adversaire $\mathcal{A}$ produit une chaine de bits c.
>
> On dit que $\mathcal{A}$ gagne si c n'est pas l'un des chiffrés c_i renvoyés par le challenger en réponse à une de ses requêtes et si c est un chiffré valide, donc si $D(k, c) \neq \perp$. L'avantage de l'adversaire $\mathbf{CIAdv}[\mathcal{A}, \mathcal{E}]$ est la probabilité qu'il gagne.

Définition 16 : Intégrité des chiffrés

> Un mécanisme de chiffrement offre **l'intégrité des chiffrés** si l'avantage de tout adversaire au jeu d'attaque ci-dessus est négligeable.

On aurait pu définir sensiblement la même notion en considérant un adversaire ayant accès à un oracle de chiffrement et dont la mission consisterait à produire un chiffré valide qu'il n'a pas demandé à son oracle.

Définition 17 : Chiffrement authentifié

> Un mécanisme de chiffrement authentifié $\mathcal{E} = (G, E, D)$ offre le **chiffrement authentifié** si 1) il est sémantiquement sûr sous des attaques à chiffrés choisis (IND-CPA) et 2) il offre l'intégrité des chiffrés.

Un mécanisme de chiffrement authentifié est nécessairement résistant aux attaques à *chiffrés* choisis. En effet, imaginons un adversaire ayant accès à deux oracles, un pour le chiffrement et un autre pour le déchiffrement. En fait, le deuxième ne sert à rien : le seul moyen d'en obtenir une autre réponse que $\perp$ consiste à lui transmettre un chiffré produit par le premier.

On voir comment construire un mécanisme de chiffrement authentifié en combinant un chiffrement IND-CPA d'une part, et un mécanisme garantissant l'intégrité des messages d'autre part.

3.2 Codes d'Authentification de Messages (MAC)

Un code d'authentification de message est à la cryptographie symétrique ce que la signature est à la cryptographie à clef publique.

Définition 18 : Code d'authentification de message

> Un **code d'authentification de message** est un triplet (G, S, V) d'algorithmes polynomiaux qui satisfont les conditions suivantes :
> — G (key Generation) exécuté sur 1^n produit une chaine de bits (la clef).
> — *Correction* : pour tout $k \leftarrow G(1^n)$ et pour tout $\alpha \in \{0,1\}^*$, les algorithmes S (Signature) et V (Verification) satisfont $\mathbb{P}\left[V(k, \alpha, S(k, \alpha)) = 1\right] = 1$.
> — V est (généralement) déterministe, S peut être randomizé.
> La chaine de bits $S(k, \alpha)$ est la « signature » du message α sous la clef k. Pour éviter l'ambiguité avec les signatures à clefs publiques, on appelle $S(k, \alpha)$ le **tag** (étiquette).
>
> Lorsque (α, t) est accepté par l'algorithme V (il renvoie 1), alors t est un tag valide.

Un MAC est *sûr* s'il est impossible de forger un tag valide sans connaître la clef. La définition est quasiment exactement la même que celle de l'intégrité des chiffrés. Pour changer, on donne une définition sans jeu d'attaque.

Définition 19 : Inforgeabilité existentielle

> Pour une machine probabiliste à oracle M, on note $Q_M^O(x)$ l'ensemble des requêtes adressées par M à son oracle O lorsqu'on lui soumet l'entrée x. Comme précédemment, on note $M^O(x)$ la sortie de M à partir de l'entrée x lorsque M à accès à l'oracle O.
>
> Il faut noter que $Q_M^O(x)$ et $M^O(x)$ sont deux variables aléatoires qui sont « dérivées » du même processus de calcul, donc elles ne sont pas indépendantes.
>
> Un MAC est **sûr** (c.a.d. **existentiellement inforgeable**) si pour toute machine à oracle polynomiale probabiliste, pour tout polynôme (positif) et pour tout n suffisamment grand, on a :
>
> $$\mathbb{P}\left[\begin{array}{l} V_k(\alpha, t) = 1 \text{ et } \alpha \notin Q_M^{S_k}(1^n) \\ \qquad \text{avec } k \leftarrow G(1^n) \text{ et } (\alpha, t) \leftarrow M^{S_k}(1^n) \end{array}\right] < \frac{1}{p(n)}$$
>
> La valeur absolue ci-dessus est $\mathbf{MACAdv}[\mathcal{A}, \mathcal{M}]$, l'avantage de $\mathcal{A}$ contre $\mathcal{M}$.

Construction à partir d'une PRF $\{0,1\}^* \to \{0,1\}^n$.

Hash-and-hide avec fonctions de hachages résistantes aux collisions

Exemple : HMAC.

3.3 Chiffrement authentifié « Encrypt-then-MAC »

On va voir qu'authentifier les chiffrés avec un MAC donne un chiffrement authentifié. Étant donné un mécanisme de chiffrement $\mathcal{E} = (G_\mathcal{E}, E, D)$ et un MAC $\mathcal{M} = (G_\mathcal{M}, S, V)$, on forme un nouveau mécanisme de chiffrement $\mathcal{E}' = (G', E', D')$ de la façon suivante.

— Pour générer une clef de $\mathcal{E}'$, on génère une clef de chiffrement (pour $\mathcal{E}$) et une clef de vérification (pour $\mathcal{M}$). Par conséquent, $G(1^n) := (G_\mathcal{E}(1^n), G_\mathcal{M}(1^n))$.
— Pour réaliser le chiffrement, on chiffre avec $\mathcal{E}$ puis on authentifie le chiffré : $E'((k_\mathcal{E}, k_\mathcal{M}), \alpha) = (\beta, S(k_\mathcal{M}, \beta))$, où $\beta = E(k_\mathcal{E}, \alpha)$.
— Pour réaliser le déchiffrement, on vérifie l'intégrité du chiffré. S'il est invalide, on renvoie $\perp$, sinon on le déchiffre normalement.

$$D'((k_\mathcal{E}, k_\mathcal{M}), (\beta, t)) = \begin{cases} \perp & \text{si } V(k_\mathcal{M}, \beta, t) = 0 \\ D(k_\mathcal{E}, \beta) & \text{sinon} \end{cases}$$

Théorème 9. *Si $\mathcal{E}$ est IND-CPA et que $\mathcal{M}$ est un MAC sûr, alors $\mathcal{E}'$ est un chiffrement authentifié.*

Démonstration

> **Claim** : $\mathcal{E}'$ offre l'intégrité des chiffrés.
>
> Le MAC est là précisément pour ça.

Démonstration

Pour cela, on démontre la contraposée : on suppose que ce n'est pas le cas, (c.a.d. qu'il existe un adversaire $\mathcal{A}$ capable de gagner le jeu 5 avec avantage non-négligeable), et on en déduire que le MAC n'est pas sûr.

Il nous faut donc construire un adversaire $\mathcal{B}$ capable de forger le MAC avec avantage non-négligeable. C'est très simple : il suffit d'exécuter l'algortihme $\mathcal{A}$ et de simuler le challenger. Ceci nécessite de répondre aux requêtes de chiffrement de $\mathcal{A}$ (qui s'attend à recevoir des chiffrés valides de $\mathcal{E}'$).

Pour cela, $\mathcal{B}$ génère aléatoirement une clef de chiffrement k_e pour $\mathcal{E}$, ce qui lui permet de produire les chiffrés. Notons aussi que $\mathcal{B}$ à accès à un oracle de « signature » (sous une clef k_m inconnue), ce qui va lui permettre de calculer les tags associés aux chiffrés.

Lorsque $\mathcal{A}$ demande le chiffrement d'un message m_i, en résumé, la simulation effectuée par $\mathcal{B}$: calcule le chiffré $\beta \leftarrow E(k_e, \alpha)$ en exécutant E avec la clef choisie par $\mathcal{B}$, puis calcule le tag t en soumettant β à son propre oracle de signature et enfin transmet (β, t) à $\mathcal{A}$. Ceci simule correctement le chiffrement $\mathcal{E}'$ pour la clef (k_e, k_m).

Lorsque $\mathcal{A}$ termine et renvoie une paire (c, t), alors $\mathcal{B}$ renvoie la paire (c, t). Lorsque $\mathcal{A}$ gagne, c'est-à-dire renvoie un chiffré valide « frais » (qu'il n'a pas obtenu du challenger simulé), alors : 1) c n'a jamais été soumis par $\mathcal{B}$ à son oracle de signature et 2) t est un tag valide pour c sous la clef k_m.

Si $\mathcal{A}$ possède un avantage non-négligeable, alors il existe un polynôme p et une infinité de n tels que $\mathbf{CIAdv}[\mathcal{A}, \mathcal{E}'] > 1/p(n)$. Par conséquent, $\mathcal{B}$ casse le MAC car la définition 19 est contredite.

Claim : $\mathcal{E}'$ est IND-CPA.

C'est logique, car la différence par rapport à $\mathcal{E}$ est l'ajout du tag, qui est calculé à partir du *chiffré* (et donc qui n'apporte à priori pas d'information sur le clair).

Démonstration

Contraposée : on suppose que $\mathcal{E}'$ n'est pas IND-CPA et on en conclut que $\mathcal{E}$ n'est pas IND-CPA. Considérons donc un adversaire $\mathcal{A}$ qui obtient un avantage non-négligeable au jeu d'attaque 4 pour $\mathcal{E}'$. On construit un adversaire $\mathcal{B}$ qui joue au même mais pour $\mathcal{E}$.

$\mathcal{B}$ fonctionne en simulant l'expérience à laquelle $\mathcal{A}$ est soumis. Il doit en particulier simuler l'oracle de chiffrement de $\mathcal{A}$ qui implémente le chiffrement $\mathcal{E}$ sous une clef (k_e, k_m) inconnue. Pour cela, $\mathcal{B}$ tire au hasard une clef de MAC k_m ; il utilise son oracle de chiffrement (qui implémente $\mathcal{E}$) pour réaliser le chiffrement, et calcule lui-même le tag en exécutant l'algorithme S afin de simuler $\mathcal{E}'$ auprès de $\mathcal{A}$.

Encore une fois, $\mathcal{B}$ répond correctement si et seulement si $\mathcal{A}$ répond correctement, donc par hypothèse $\mathcal{B}$ obtient un avantage non-négligeable.

□

3.4 Constructions d'un MAC à partir d'une PRF

On construit aisément un MAC $\mathcal{M} = (G, S, V)$ à partir d'une PRF f : il suffit d'appliquer la PRF sur le message pour obtenir le tag. L'inconvénient, c'est qu'on ne peut authentifier que des messages de taille fixe (n bits). Mais c'est déjà un début.

Pour générer une clef du MAC, on génère un indice de la PRF : $G(1^n) := I(1^n)$. Pour calculer le tag d'un message α, on pose $S(i, \alpha) := f_i(\alpha)$. Étant donné une paire (t, α), on vérifie si t est un tag valide pour α sous la clef k en recalculant le tag :

$$V(i, \alpha, t) := \begin{cases} 1 & \text{si } f_i(\alpha) = t \\ 0 & \text{sinon} \end{cases}$$

On note que l'algorithme S est déterministe.

Théorème 10. *Si f est une PRF, alors la construction $\mathcal{M}$ ci-dessus est un MAC sûr pour des messages de n bits.*

Le fond de l'affaire, c'est que la PRF f est pseudo-aléatoire, et donc *imprédictible*, et donc aucun adversaire ne peut « prédire » le MAC sans évaluer f.

Démonstration

On applique la stratégie habituelle des preuves avec des PRFs. D'abord, on montre qu'une fonction aléatoire est un MAC sûr.

Soit ϕ une fonction uniformément aléatoire $\{0,1\}^n \to \{0,1\}^n$ Considérons un adversaire $\mathcal{A}$ qui essaye de casser le MAC $\mathcal{M}'$ défini à partir de $\mathcal{M}$ en remplaçant f (la PRF) par ϕ (la fonction aléatoire).

Claim : $\mathbf{MACAdv}[\mathcal{A}, \mathcal{M}'] \leq 2^{-n}$.

Démonstration

Supposons que $\mathcal{A}$ renvoie une paire (m, t) et qu'il n'a pas soumis de requête pour m à ϕ. L'idée de fond, c'est qu'alors $\phi(m)$ est une chaine de n bits aléatoire sur laquelle $\mathcal{A}$ ne possède aucune information. $\mathcal{A}$ gagne si $\phi(m) = t$, or la probabilité que ceci arrive est de 2^{-n}, quelle que soit la valeur de t (la probabilité est prise sur le choix aléatoire de ϕ).

Après, c'est toujours la même chose : avec une fonction aléatoire, tout adversaire ne possède qu'un avantage négligeable, donc si un adversaire obtient un avantage significatif, c'est qu'il distingue la PRF d'une fonction aléatoire (on peut alors refaire marcher la 2ème partie de la preuve du théorème 6).

***Extension : PRF à entrée arbitraire**

Les PRF de la section 2.5.2 ont une entrée de taille finie, et par conséquent la construction précédente n'est capable de MACer que des messages de taille fixe. On considère ici une généralisation : les PRFs qui sont définies sur n'importe quelle chaine de bits.

Définition 20 : Ensemble de fonctions pseudo-aléatoire à entrée non-bornée

Soit $r : \mathbb{N} \to \mathbb{N}$ une fonction polynomialement bornée. On dit que

$$\left\{ f_s : \{0,1\}^* \to \{0,1\}^{r(|s|)} \right\}_{s \in \{0,1\}^*}$$

est un **ensemble de fonctions à entrée non-bornée, pseudo aléatoires et efficacement calculables** si les deux conditions suivantes sont réunies :

- *Evaluation efficace* : il existe un algorithme polynomial qui calcule $f_s(x)$ à partir de s et x.
- *Pseudo-aléa* : pour toute machine polynomiale probabiliste avec oracle M, tout polynôme p et pour tout n suffisamment grand, on a :

$$\left| \mathbb{P}\left[M^{F_n} = 1\right] - \mathbb{P}\left[M^{H_n} = 1\right] \right| < 1/p(n),$$

où F_n est une variable aléatoire uniformément distribuée dans le (multi)-ensemble $\{f_s\}_{s \in \{0,1\}^n}$ et où H_n est une fonction uniformément distribuée dans $\{0,1\}^* \to \{0,1\}^{r(n)}$.

Remarque technique : la définition de H_n pourrait sembler problématique car l'ensemble des fonctions $\{0,1\}^* \to \{0,1\}^{r(n)}$ est infini (la distribution uniforme n'est pas définie sur un ensemble infinie). En réalité, comme le temps d'exécution de M est polynomial, il existe un polynôme d tel que les requêtes adressées à son oracle par M sont de taille bornée par $d(n)$. Il suffit de considérer que H_n est uniformément distribuée dans l'ensemble (fini) $\{0,1\}^{d(n)} \to \{0,1\}^{r(n)}$.

Il est instructif de comparer cette définition avec la définition 11 (des PRFs « normales »). Ici, n'importe quelle chaine de bits s est un « indice » valable, et il n'y a pas besoin d'algorithme d'indexation. Le cadre est donc un tout petit peu moins général.

L'inconvénient de cette construction à base de PRF plus puissantes, c'est qu'elle ne nous avance pas beaucoup : il faudrait construire une telle PRF.

3.5 MAC à partir de fonctions de hachage résistantes aux collisions

Pour être capable d'authentifier des messages arbitrairement longs, plusieurs solutions sont possibles. L'une des plus simples consiste à utiliser une fonction de hachage résistante aux collisions.

3.5.1 Définition des fonctions de hachages résistantes aux collisions

Il est délicat de définir les fonctions de hachages cryptographiques. L'idée générale, légèrement contradictoire, est la suivante : il faut d'une part que l'empreinte $H(x)$ d'une donnée x (éventuellement secrète) contienne assez

d'information sur x pour qu'on puisse la distinguer d'une empreinte de $x' \neq x$ avec quasi-certitude. Mais il faut d'autre part que l'empreinte de x ne révèle *aucune information exploitable* sur x.

Une fonction de hachage (résistante aux collisions) est une fonction $\{0,1\}^* \to \{0,1\}^n$, qui ne peut pas être injective car son ensemble d'entrée est infini, et son ensemble image est fini. Mais on voudrait que tout se passe comme si elle l'était. En réalité, on ne veut pas que l'humanité soit capable de trouver des collisions dessus (des paires d'antécédents qui ont la même image).

Cette notion de résistance aux collisions est surprenament difficile à formaliser. Par exemple, la définition ci-dessous, qui semble raisonnable, ne marche pas du tout.

Définition 21 : Fonction de hachage résistante aux collisions (CRHF) — mauvaise définition

Une **fonction de hachage résistante aux collisions** est une fonction $H : \{0,1\}^* \to \{0,1\}^n$ sujette aux deux conditions suivantes :

1. *Evaluation efficace* : il existe un algorithme polynomial qui calcule $H(x)$ à partir de x.
2. *Résistance aux collisions* : pour tout algorithme polynomial $\mathcal{A}$, tout polynôme p et tout n suffisamment grand :
$$\mathbb{P}\left[\begin{array}{r} m_0 \neq m_1 \text{ et } H(m_0) = H(m_1) \\ \text{où } (m_0, m_1) \leftarrow \mathcal{A}(1^n) \end{array}\right] < 1/p(n)$$

Cette définition a plusieurs gros problèmes.

Pour commencer, y a-t-il une famille de fonctions H_n (ce qui n'apparaît pas dans la définition) ou bien une seule fonction H ? Si c'est le deuxième cas, alors que vaut n ? Dans le deuxième cas, également, l'adversaire $\mathcal{A}$ est polynomial en quoi ?

Pour essayer de s'en sortir, on pourrait garder une seule fonction, appeler n la taille de sa sortie, et demander qu'il n'existe pas d'adversaire fonctionnant en moins de $2^{n/2}$ opérations qui trouve une collision avec probabilité supérieure à 99%. En effet, c'est le temps qu'il faut pour trouver une collision avec probabilité $1 - 1/e$ en faisant une attaque par force brute.

Malheureusement, si la fonction H est fixée, il existe toujours des adversaires qui réussissent tout le temps et qui ne font... rien ! En effet, comme H n'est pas injective, il *existe* des entrées qui collisionnent, donc il *existe* un adversaire qui contient une collision et qui ne fait rien d'autre que de la renvoyer. Le problème, c'est que l'humanité ne sait pas forcément comment trouver cet adversaire. Mais il *existe*.

On peut vraiment s'en tirer en considérant des familles de fonctions de hachage indexées par des clefs. Alors, comme l'adversaire doit recevoir la clef avant de renvoyer une éventuelle collision, il ne peut pas la contenir à l'avance. Ceci est théoriquement satisfaisant, mais en pratique, les fonctions de hachage cryptographiques n'ont pas de clef.

On pourrait alors tordre un peu la définition en imposant à l'adversaire de recevoir un préfixe p, puis de produire (m_0, m_1) tels que pm_0 et pm_1 collisionnent... Mais bon on va s'arrêter là. Bref : c'est compliqué.

Pour dire des choses raisonnables, on va supposer que les fonctions de hachages ont des « clefs ».

Définition 22 : Fonction de hachage résistante aux collisions (CRHF) — bonne définition

Soit une fonction $\ell : \mathbb{N} \to \mathbb{N}$. Une famille de fonctions $\{h_s : \{0,1\}^* \to \{0,1\}^{\ell(s)}\}_{s \in \{0,1\}^*}$ est appelée une **fonction de hachage résistante aux collisions** s'il existe deux algorithmes I et H tels que les deux conditions suivantes sont réunies :

1. *Indexation efficace* : $I(1^n)$ est un indice valide pour h.
2. *Evaluation efficace* : $H(s, x)$ calcule $h_s(x)$.
3. *Résistance aux collisions* : une **collision** pour une fonction f est une paire $x \neq y$ avec $f(x) = f(y)$. Pour tout algorithme polynomial $\mathcal{A}$, tout polynôme p positif et tout n suffisamment grand :
$$\mathbb{P}\left[\mathcal{A}(s) \text{ est une collision pour } h_s \text{ où } s \leftarrow I(1^n)\right] < 1/p(n).$$

Cette dernière probabilité est l'avantage de $\mathcal{A}$ et on la note $\mathbf{CFAdv}[\mathcal{A}, h]$.

3.5.2 Construction d'un MAC

Armés d'une CRHF $\{h_s\}$ représentée par les deux algorithmes (I, H) et d'un MAC $\mathcal{M} = (G, S, V)$ pour des entrées de n bits (cf. § 3.4), nous pouvons construire un MAC à entrée arbitraire $\mathcal{M}' = (G', S', V')$.

— *Génération de clef* : on produit une clefs pour les deux sous-constructions :

$$G'(1^n) = (I(1^n), G(1^n)).$$

— *Authentification* : on authentifie l'empreinte des données avec $\mathcal{M}$:

$$S'((s,k),x) = S(k, h_s(x)).$$

— *Vérification* : on vérifie que le tag a été calculé correctement

$$V'(k',x,t) = \begin{cases} 1 & \text{si } t = S'(k',x) \\ 0 & \text{sinon} \end{cases}.$$

La résistance aux collisions de la fonction de hachage est indispensable. En effet, si un adversaire peut trouver une collision, alors il peut forger le MAC : s'il trouve une collision $x \neq y, h_s(x) = h_s(y)$, alors il peut obtenir le tag t de x (en soumettant une requête au challenger). Il suffit qu'il renvoie (y,t), et il gagne : t est un tag valide à la fois pour x et pour y.

Théorème 11. *Si $\{h_s\}$ est une CRHF et $\mathcal{M}$ est un MAC sûr pour des entrées de n bits, alors $\mathcal{M}'$ est un MAC sûr pour des entrées arbitraires.*

Démonstration

L'idée générale est la suivante : si un adversaire $\mathcal{A}$ parvient à forger $\mathcal{M}'$, alors ou bien il est capable de trouver des collisions dans la fonction de hachage, ou bien il est capable de forger $\mathcal{M}$.

Admettons qu'on ait un adversaire $\mathcal{A}$ qui forge $\mathcal{M}'$, c'est-à-dire qui contredit la définition 19. Notons $m_1, m_2, \ldots$ les requêtes qu'il soumet à son oracle de « signature » et notons (m,t) sa réponse, où on supposera que m n'est pas l'un des m_i (si ce n'est pas le cas, l'adversaire ne forge rien).

On considère les trois évènement suivant :
— X : $\mathcal{A}$ parvient à forger $\mathcal{M}'$ (il renvoie une paire valide).
— Y : l'une des requêtes m_i collisionne avec m pour la fonction de hachage utilisée, h_s.
— Z : X est réalisé mais pas Y.
On a alors :

$$\mathbf{MACAdv}[\mathcal{A}, \mathcal{M}'] = \mathbb{P}[X] \leq \mathbb{P}[X \wedge \neg Y] + \mathbb{P}[Y] = \mathbb{P}[Z] + \mathbb{P}[Y].$$

(cette majoration est brutale).

Pour démontrer le théorème, on construit deux algorithmes : $\mathcal{B}$, un adversaire contre la fonction de hachage (il cherche des collisions) et $\mathcal{C}$ un adversaire contre le MAC $\mathcal{M}$ (il tente de le forger). On va avoir $\mathbf{CFAdv}[\mathcal{B}, h] = \mathbb{P}[Y]$ et $\mathbf{MACAdv}[\mathcal{C}, \mathcal{M}] = \mathbb{P}[Z]$.

Vu que, par hypothèse, $\mathbb{P}[X]$ n'est pas négligeable, alors au moins l'un des deux adversaire $\mathcal{B}$ et $\mathcal{C}$ a un avantage non-négligeable, ce qui prouvera le théorème.

La construction des deux adversaires est assez simple : ils ne font que simuler le challenger de $\mathcal{A}$.

Construction de $\mathcal{B}$: cet adversaire reçoit en paramètre un indice s pour la fonction h. Il génère honnêtement une clef k pour le MAC $\mathcal{M}$ puis exécute $\mathcal{A}$ en simulant l'oracle de MAC (censé implémenter $\mathcal{M}'$) auquel $\mathcal{A}$ a accès. Lorsque $\mathcal{A}$ soumet sa i-ème requête m_i à son oracle, $\mathcal{B}$ lui renvoie $S(k, h_s(m_i))$. Lorsque $\mathcal{A}$ termine et renvoie (m,t), deux situations sont possibles : m collisionne avec l'un des m_i et alors $\mathcal{B}$ renvoie la collision (m, m_i) et gagne. Sinon, il renvoie $\perp$ et perd. L'avantage de $\mathcal{B}$ (la probabilité de trouver une collision) est exactement $\mathbb{P}[Y]$.

Construction de $\mathcal{C}$: cet adversaire a accès à un oracle qui implémente $\mathcal{M}$ sous une clef inconnue k. Il génère aléatoirement un indice valide pour la fonction de hachage, et exécute $\mathcal{A}$ en simulant le $\mathcal{M}'$-oracle auquel $\mathcal{A}$ a accès (il effectue le hachage lui-même puis soumet l'empreinte à son $\mathcal{M}$-oracle pour obtenir le tag à renvoyer à $\mathcal{A}$). Lorsque $\mathcal{A}$ termine et renvoie (m,t), deux situations peuvent se produire : si m collisionne avec l'un des m_i pour h_s, alors $\mathcal{C}$ renvoie $\perp$ et perd. Sinon, $\mathcal{C}$ renvoie $(h_s(m), t)$ et forge $\mathcal{M}$ avec succès. L'avantage de $\mathcal{C}$ (la probabilité de forger $\mathcal{M}$) est exactement $\mathbb{P}[Z]$. □

3.6 construction de fonctions de hachages cryptographiques

Pour faire marcher la construction précédente, il nous faut une fonction de hachage résistante aux collisions.

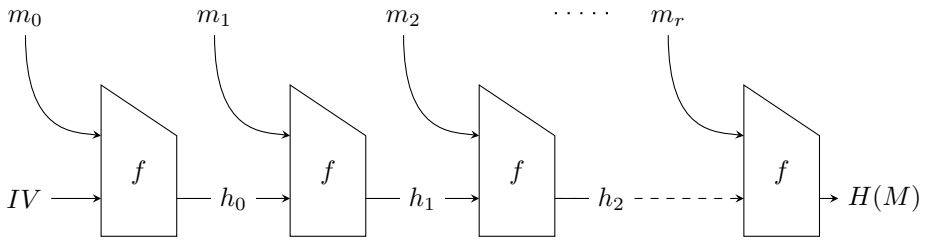
Pour hacher des chaines de bits arbitrairement longues, il est naturel d'utiliser une construction itérée, où l'entrée est découpée en blocs et traitée un bloc après l'autre. La technique décrite ici est le mode opératoire de *Merkle-Damgård* du nom de ceux qui l'ont décrit précisément en 1990, et qui ont démontré certaines de ses propriétés. Il a été largement utilisé ensuite, dans MD5, SHA-1/2... Mais à cause d'un certain nombre de ses problèmes, SHA-3 est construit différement.

3.6.1 Description

L'idée consiste à utiliser de manière itérative une *fonction de compression*

$$f : \{0,1\}^{n+m} \longrightarrow \{0,1\}^n.$$

La fonction de compression est une « petite » fonction de hachage, qui « hache » $n+m$ bits en n bits. Elle possède généralement deux entrées qui ont des rôles distincts : le *bloc de message* et la *valeur de chainage*. Un petit dessin est plus explicatif :



Le message M donc on veur une empreinte est découpé en blocs de m bits, et ces blocs sont hachés successivements. La première valeur de chainage, l'IV, est une constante fixé par la spécification des fonctions de hachages concrètes. Un *padding* est appliqué sur le dernier bloc (ceci permet de hacher des messages dont la taille n'est pas un multiple de m bits). Ce *padding* DOIT contenir la *taille du message* (avant ajout du *padding*, bien sûr).

Pour donner une idée, dans SHA-256 (resp. SHA-512), les blocs font $m = 512$ bits (resp. $m = 1024$), et taille des valeurs de chainage est de 256 bits (resp. 512). Dans le dernier bloc, la taille du message est donnée en bits, et est codée sur 128 bits. Ceci impose une limite (absurdement élévée) sur la taille des messages qui peuvent être hachés.

La valeur de l'IV fait partie de la spécification, et elle est arbitraire.

3.6.2 Résistance aux préimages et aux collisions

L'intérêt principal de ce mode opératoire c'est qu'il possède des formes de « sécurité prouvée ». Plus exactement, on sait que si la fonction de compression n'a pas trop de problèmes, alors l'ensemble sera raisonnablement sûr. Tout d'abord, il est facile de voir que si on est capable de trouver des préimages sur H, alors on est capable d'en trouver sur f (attaque sur le dernier bloc). Mais l'essentiel est dans le résultat suivant :

Théorème 12 (Merkle-Damgård, 1990)**.** *Si la fonction de compression f est résistante aux collisions, alors la fonction de hachage H^f construite à partir de f via le mode opératoire de Merkle-Damgård est résistante aux collisions.*

Démonstration

La preuve est une réduction qui transforme une collision sur H^f en collision sur f. Autrement dit, à partir d'une paire de message $M \neq M'$ telle que $H^f(M) = H^f(M')$, on produit deux messages $x \neq x'$ de $n+m$ bits tels que $f(x) = f(x')$. Si f est résistante aux collisions, trouver ces deux messages est calculatoireement impossible, donc par conséquent trouver M et M' est calculatoirement impossible aussi.

Supposons donc que nous ayons deux messages $M \neq M'$ tels que $H^f(M) = H^f(M')$, et tentons de produire une collision sur f. De deux choses l'une :

- Ou bien $|M| \neq |M'|$ (cette notation désigne la taille des messages). Dans ce cas, vu la définition du mécanisme de bourrage dans le mode opératoire de Merkle-Damgård, les blocs « bourrés » qui vont entrer dans la fonction de compression lors de sa dernière invocation seront différents lors du traitement de M et de M'. De plus, comme $H^f(M) = H^f(M')$, on sait que les sorties de la fonction de compression sont identiques dans les deux cas : c'est donc qu'il y a une collision sur f, qui a lieu lors du traitement du dernier bloc des deux messages.
- Ou bien $|M| = |M'|$. Supposons que les deux messages fassent r blocs, après bourrage. Il faut encore distinguer deux situations : soit $(h_{r-1}, m_r) \neq (h'_{r-1}, m'_r)$, et il y a alors une collisions sur f puisque $h_r = h'_r$. Ou bien $(h_{r-1}, m_r) = (h'_{r-1}, m'_r)$. Dans ce deuxième cas, l'argument se repète « un bloc

avant ». Ou bien on trouve une collision sur f à un moment donné, or bien nous devons conclure que $m_i = m'_i$ pour tout i, ce qui est impossible vu que les messages sont différents.

L'avantage principal de ce mode opératoire est que pour produire une bonne fonction de hachage, il suffit de produire une bonne fonction de compression. En plus, il est rapide, et permet de produire les empreintes « à la volée », sans avoir à stocker l'ensemble du message.

On a longtemps cru que si la fonction de compression résistait aux secondes préimages, alors la fonction itérée y résistait aussi. Ce n'est en 2006 qu'on s'est rendu compte que ce n'était pas vrai. Kelsey et Schneier ont montré qu'on pouvait calculer une seconde préimage d'un message M en temps $2^n/|M|$, donc plus rapidement que les 2^n auxquels on devrait avoir droit, et ce *quelle que soit la fonction de compression* (c'est un défaut du mode opératoire lui-même).

Pour boucler la boucle, il n'y a plus qu'à construire une fonction de compression. N'importe quelle PRF ferait l'affaire, et ol serait tentant d'utiliser directement l'AES (par exemple en chiffrant le bloc m_{i+1} avec la clef h_i) mais ceci aurait plusieurs inconvénients (exercice : lesquel ?).

En réalité, dans SHA-x, la fonction de compression est constuite autour de la construction de davies-Meyer : $f(h, m) := E_m(h) + h$, où E est un bloc-cipher spécial, avec un gros bloc (512/1024 bits) et la caractéristique qu'il doit offrir une certaine sécurité même si la clef est sous le contrôle de l'adversaire.

3.6.3 Digression : modèle de l'oracle aléatoire

S'il fallait résumer, on pourrait dire qu'une fonction de hachage est une fonction $\{0,1\}^* \to \{0,1\}^n$ dont le *code source est public* et qui ne possède *aucune structure exploitable* permettant de relier ses entrées et ses sorties.

C'est un peu comme si une fonction H avait été choisie aléatoirement parmi toutes les fonctions $\{0,1\}^*\{0,1\}n$ et mise à la disposition du public. On l'appelle donc « l'oracle aléatoire ».

On pourrait imaginer construire un tel dispositif de la façon suivante : un serveur mondial exécute un *web-service* qui simule cette fonction aléatoire et permet à tous les utilisateurs d'internet de calculer des empreintes. Pour obtenir l'empreinte de M, les utilisateurs chargent une URL qui invoque la méthode `query` ci-dessous (c'est du code Python) :

```
class RandomOracle:
    log = {}

    def query(self, M):
        if M in self.log:
            return self.log[M]
        h = RandomGenerator.getrandbits(n)
        self.log[M] = h
        return h
```

Ce *web-service* fait deux choses :

- si ce n'est pas la première fois que M est demandé, il renvoie la valeur précédement renvoyée. Du coup, le résultat est une « vraie fonction » qui renvoie toujours la même valeur quand on lui soumet le même argument.
- Si M est « nouveau », alors il choisit une empreinte *complètement au hasard*, la stocke et la renvoie. Du coup, les sorties n'ont *aucun rapport* avec les entrées. En particulier, la sortie ne fait pas « fuir » d'information sur l'entrée.

Le gros problème c'est que cette construction est impossible à réaliser dans la pratique : l'espace de stockage nécessaire n'est pas du tout borné ! Cependant, ce petit bout de code garantit, par construction, l'absence de correlation entre les entrées et les sorties.

C'est ce qu'on pourrait espérer de mieux comme fonction de hachage. Bien souvent, des raisonnements cryptographiques sont basés sur l'hypothèse que les fonctions de hachage habituelles sont des oracles aléatoires. Il est difficile de déterminer dans quelle mesure ceci est irréaliste. Il existe des protocoles, simples, qui sont sûrs si H est un oracle aléatoire, et complètement cassés si H est une des fonctions de hachages habituelles (MD5, SHA1, SHA256, ...).

Étudions l'un des exemples les plus frappants. Définissons le MAC : $S(k, x) = H(k\|x)$. Il est facile de démontrer qu'il est sûr dans le modèle de l'oracle aléatoire. Par contre, il est complètement cassé si H implémente le mode opératoire de Merkle-Damgård (par exemple avec H =SHA-256.

En effet, un adversaire peut soumettre une requête m à son oracle de MAC. Il apprend le tag t, c'est-à-dire la valeur de chainage à la fin du processus hachage. Ceci lui permet de « continuer » le hachage, et de calculer lui-même l'empreinte de $m \| suffix$. Ce problème ne se poserait pas avec $S(k, x) = H(x \| k)$.