Cours de cryptographie

Balthazar Bauer

19 février 2025

Chapitre 1

Le formalisme en cryptographie (5 heures)

On a tous une intuition sur ce qu'est la chaleur, la taille ou la masse, mais définir formellement ces grandeurs n'a rien d'un processus trivial, et a été le fruit de nombreuses réflexions et recherches.

De manière analogue la notion de sécurité entre en résonance avec une définition intuitive.

Le flou de cette définition ne convient pas à la sensibilité du domaine. Si des développeurs ne sont pas d'accord sur ce que signifie "coder proprement", ce n'est pas forcément grave pour les utilisateurs. En revanche il est très important lors de l'usage d'un produit comportant des dangers qu'on soit bien au clair sur quels usages sont dangereux, et lesquels ne le sont pas.

Au flou va se coupler également la polysémie de la définition de sécurité. Il est évident que la sécurité d'une centrale nucléaire (même en ignorant ce qu'on entend par là), n'est pas la même que la sécurité d'une messagerie numérique.

Dans ce chapitre nous allons nous focaliser sur un contexte précis qui correspond à un usage facile à comprendre : les signatures numériques.

Mais avant de parler de sécurité, nous allons devoir préciser de quoi on parle.

1.1 Physiologie d'un schéma de signature numérique

En général une procédure de signatures consiste en deux phases : une procédure de signature, et une procédure de vérification de validation de la signature (on dira par abus de langange qu'on vérifie la signature). Étant donné qu'on est dans un cadre numérique, ces procédures consistent à appliquer un algorithme (ou une machine de Turing si l'on veut se référer au cadre usuel de l'informatique théorique) qui prennent en entrée et renvoie une donnée numérique.

Définition 1. *Un schéma de signature numérique sur l'ensemble des messages $\mathcal{M}$ est un triplet d'algorithmes probabilistes "efficaces" :*

- `KeyGen`() *qui ne prend pas d'entrée et qui renvoie une paire de clefs (sk, vk), la première clef est une clef de signatures (appelée génériquement clef secrète ou clef privée), tandis que vk est une clef de vérification (appelée aussi génériquement clef publique).*
- `Sign`(sk, m) *qui prend en entrée une clef de signatures sk un message $m \in \mathcal{M}$, et renvoie une signature σ.*
- `Verify`(vk, m, σ) *qui prend en entrée une clef de vérification, un message m, une signature σ et renvoie un bit $b \in \{0, 1\}$. Le bit ici doit être compris comme un bit logique qui nous indique si la signature est valide. C'est à dire qu'il renvoie* 1 *si c'est valide et* 0 *sinon.*

Pour bien comprendre informellement de quoi on parle faisons un parallèle avec les signatures manuscrites qu'on utilise encore de nos jours assez couramment.

Le concept de signature manuscrite repose sur une capacité à reproduire un certain motif. Cette capacité est censé caractériser notre identité. Pour une signature numérique, ce qui va caractériser une identité est la connaissance d'un certain secret c'est à dire de la clef de signature sk.

Remarque Évidemment, il s'agit ici d'une identité numérique et non d'une identité physique.

Remarque On laisse pour le moment floue la notion "d'efficacité", mais on peut imaginer qu'elle correspond à une norme industrielle de type "s'exécute en moins d'un dixième de seconde sur n'importe quel ordinateur de moins de 20 ans."

Pour comprendre le cas d'usage on est ici dans un cadre *asymétrique*. C'est à dire qu'on prend en compte deux types d'entités un signataire et un vérifieur, et ces derniers n'ont *pas accès aux mêmes informations* :

Le signataire utilise sa clef secrète pour signer un message en particulier et les vérifieurs utilisent la clef de vérification (qui en pratique est associée à l'utilisateur-signataire) pour vérifier que le message a été bien signé.

Afin que le schéma puisse être utilisé ainsi, il est nécessaire qu'il vérifie une propriété qu'on appelle *propriété de correction*.

Définition 2. *On dit qu'un schéma de signature numérique* $\Pi_{\mathcal{S}}$ *est correct si et seulement si : Pour toute paire* (sk, vk) *générée par* `KeyGen`*, pour tout message* $m \in \mathcal{M}$:

$$\Pi_{\mathcal{S}}.\texttt{Verify}(vk, m, \Pi_{\mathcal{S}}.\texttt{Sign}(sk, m)) = 1.$$

1.2 Définissons la sécurité pas à pas

Il y a trois écueils à éviter lorsqu'on construit un modèle :
- Déjà, le manque de formalisme qui rend le modèle imprécis.
- Si intrinsèquement le modèle de sécurité contredit les propriétés de correction, c'est à dire rendent inopérant tout schéma qui le satisferait. Par exemple si je considère qu'une centrale nucléaire est sécurisée uniquement si il n'y a aucun humain dans les 500 kilomètres à la ronde du réacteur, alors qu'il est nécessaire pour la faire fonctionner (et donc qu'elle soit utile) que des agents travaillent à proximité, on peut dire que le modèle est *structurellement trop fort*.
- Si on se rend compte qu'en pratique le modèle ne couvre pas des stratégies d'attaques tout à fait faisable en pratique. On va considérer que le modèle est *trop faible*.

Remarque On notera qu'un modèle *structurellement trop fort* n'a aucun intérêt dans l'absolu (car l'ensemble des constructions correctes qui le vérifie est vide). En revanche un modèle trop faible peut avoir un intérêt si on change de contexte.

Puisqu'il faut toujours partir de quelque part, on va commencer avec cette définition :

Définition 3. *Un schéma* $\Pi_{\mathcal{S}}$ *est sécurisé si aucun individu malveillant ne peut signer un message qu'il n'est pas censé signer.*

On peut remarquer que dans ce cours, on appelle sécurité le fait de prévenir contre des risques d'actes malveillants prémédités, et non contre des aléas "naturelles" (comme la météo, le bruit électrique), où là on parlerait plus de sûreté (*safety* en anglais). On va commencer par évacuer toutes notions morales qui seraient beaucoup trop compliquées à définir. D'une manière général dans le domaine de sécurité on ne présuppose jamais des intentions finales de l'adversaire.

Définition 4. *Un schéma* $\Pi_{\mathcal{S}}$ *est sécurisé si aucun individu ne peut signer un message qu'il n'est pas censé signer.*

On remarque qu'on a alors renforcé la définition paradoxalement, puisque la classe d'individus ne pouvant pas signer un message augmente. Mais qu'on s'interdit d'avoir une propriété magique de type *si un individu est bien intentionné il peut alors signer un message*.

C'est peut-être malheureux, mais c'est quelque chose qu'on admet. Tout le monde accepte l'idée que si il fait claquer une porte blindée en laissant la clef intérieur, il ne peut plus rentrer sans endommager la porte.

A présent, réfléchissons à ce que signifie "réussir à signer un message". Il ne s'agit pas forcément de dire que l'individu utilise l'algorithme `Sign` puisqu'on ne regarde pas précisément l'exécution qui a généré la signature (de la même manière qu'on ne vous demande pas une vidéo en train de signer manuscriptement un document attaché au document).

Évidemment, on doit considérer dans notre définition l'ensemble de messages $\mathcal{M}$ duquel dépend le schéma de signature.

On va reprendre notre analogie, avec les signatures manuscrites, une signature est valide si la personne qui la vérifie est convaincue ; Étant donné qu'on est dans un cadre numérique, la vérification consiste à appliquer une procédure algorithmique prenant en entrée le message et la signature. Dans notre cas cela va donc donner :

Définition 5. *Un schéma $\Pi_{\mathcal{S}}$ est sécurisé si pour tout message $m \in \mathcal{M}$, aucun individu ne peut générer une signature σ, telle que $\Pi_{\mathcal{S}}.\texttt{Verify}(vk, m, \sigma) = 1$.*

Maintenant il serait bon de définir comment on sait que quelqu'un est censé signer un message ou pas.

Étant donné la propriété de correction qu'on a définit, on sous-entend qu'au moins tout utilisateur qui a la clef secrète sk peut signer. On va partir sur cet ensemble et donc les utilisateurs qui ne sont pas censés signer sont ceux qui n'ont pas la clef secrète.

En revanche l'adversaire (l'individu qui cherche à *casser le schéma*) est censé avoir accès à toutes les données publiques, en particulier à vk.

Comme dans les signatures "de la vie de tous les jours", ceux qui sont susceptibles de vérifier un document ne sont pas nécessairement les individus qui sont officiellement en capacité de signer, c'est à dire que nous sommes dans un cadre *asymétrique*[1].

Définition 6. *Un schéma $\Pi_{\mathcal{S}}$ est sécurisé si pour tout message $m \in \mathcal{M}$ aucun individu possédant une clef vk ne peut générer une signature σ, telle que $\Pi_{\mathcal{S}}(vk, m, \sigma) = 1$.*

Bien sur il ne faut pas oublier de préciser l'origine de la clef c'est à dire qu'il est le résultat d'une exécution de l'algorithme `KeyGen`.

Définition 7. *Un schéma $\Pi_{\mathcal{S}}$ est sécurisé si pour tout message $m \in \mathcal{M}$, pour toute clef vk générée par $\Pi_{\mathcal{S}}.\texttt{KeyGen}$, aucun individu possédant vk ne peut générer une signature σ, telle que $\Pi_{\mathcal{S}}.\texttt{Verify}(vk, m, \sigma) = 1$.*

Maintenant, il serait bon de se passer la notion d'individu qui n'a pas de réalité mathématique. On va donc utiliser la notion d'algorithme en nous basant sur l'idée que ça représente bien les capacités calculatoires d'un humain du vingt-et-unième siècle, c'est à dire une machine qui prend en entrée des données (ici vk), fait une série de calculs élémentaires.

Définition 8. *Un schéma $\Pi_{\mathcal{S}}$ est sécurisé si pour tout message $m \in \mathcal{M}$, pour toute clef vk générée par $\Pi_{\mathcal{S}}.\texttt{KeyGen}$, aucun algorithme possédant vk ne peut générer une signature σ, telle que $\Pi_{\mathcal{S}}.\texttt{Verify}(vk, m, \sigma) = 1$.*

Remarque Ici on ne précise pas, mais par algorithme on entend selon votre contexte préféré : Machines de Turing probabilistes, fonctions récursives, fonctions de lambda calcul, machine RAM ou tout paradigme de calcul universel ayant la particularité d'avoir été correctement et rigoureusement définis formellement, et censé être "équivalent" aux capacités de calcul d'un homo sapiens (et donc équivalents à tous les modèles cités).

Malheureusement, cette définition semble un peu trop puissante, en effet on peut toujours programmer un algorithme trivial comme l'attaque $\mathcal{FB}$ de la figure 1.1 qui fait une recherche exhaustive parmi les chaînes de caractères σ, d'une qui vérifie $\Pi_{\mathcal{S}}.\texttt{Verify}(vk, m, \sigma)$.

Heureusement en pratique un adversaire n'a pas une capacité de temps infini.

Ici on ne va pas chercher à définir précisément les capacités de l'adversaire, puisque ça va dépendre du contexte. En effet, un pirate informatique avec un ordinateur portable n'a pas la même puissance de calcul que la NSA.

De plus certaines signatures n'ont qu'une valeur temporelle assez limitée. Par exemple sur internet dans certains protocoles, il y a des clefs éphémères valides seulement le temps d'une session de connexion.

Ici on va rester assez abstrait, et on va paramétrer la sécurité par rapport à une puissance de calcul donnée et on laissera aux utilisateurs le soin de choisir les paramètres qui les intéressent selon l'application qui leur convient.

A noter qu'on pourrait affiner la définition en prenant en compte d'autre types de ressources que le temps de calcul (la mémoire par exemple).

1. Pour voir l'équivalent *symétrique*, lire le chapitre 3 sur les codes d'authentification de messages.

Remarque En général pour estimer des grandeurs universelles, on utilise des bornes physiques par rapport à l'univers : son âge (en secondes) et son nombre d'atomes. Par exemple, si on est dans un scénario du pire-cas où tous les atomes sont utilisés comme des unités de calculs (c'est à dire comme des processeurs), et peuvent faire autant d'opérations que l'âge de l'univers en secondes, on arrive à borner cette capacité de calcul par 2^{128} opérations élémentaires, mais des estimations un peu plus fines considèrent également les grandeurs 2^{64} ou 2^{80}.

Définition 9. *Soit $t \in \mathbb{N}$. Un schéma $\Pi_{\mathcal{S}}$ est t-sécurisé si pour tout message $m \in \mathcal{M}$, pour toute clef vk générée par $\Pi_{\mathcal{S}}$.*`KeyGen`*, aucun algorithme utilisant un temps au plus t possédant vk ne peut générer une signature σ, telle que $\Pi_{\mathcal{S}}$.*`Verify`*$(vk, m, \sigma) = 1$.*

On avance, mais ce n'est toujours pas suffisant. En effet un adversaire particulièrement chanceux peut tirer au hasard la signature valide en très peu de temps.

Évidemment, sa probabilité de succès sera très faible, mais ce cas n'est pas encore pris en compte par l'actuelle définition.

Une fois de plus, on peut définir un seuil de probabilité à partir duquel on tolère que l'adversaire puisse casser un système.

Et comme pour le temps de calcul, on va éluder le problème de déterminer ce seuil en le posant en paramètre.

Définition 10. *Soit t un entier naturel, $\epsilon \in [0; 1]$ Un schéma $\Pi_{\mathcal{S}}$ est (t, ϵ)-sécurisé si pour tout message $m \in \mathcal{M}$, pour toute clef vk générée par Π.*`KeyGen`*, pour tout algorithme probabiliste $\mathcal{A}$, qui s'exécute en temps au plus t :*

$$\Pr\left[\Pi_{\mathcal{S}}.\texttt{Verify}(vk, m, \sigma) = 1\right] \leq \epsilon.$$

A noter que la probabilité se fait par rapport à l'aléa interne de l'adversaire.

Malheureusement, une fois de plus il existe toujours un attaquant structurel, celui qui a accès à sk codé en dur, par exemple $\mathcal{A}(vk) : \Pi.\texttt{Sign}(sk, m)$.

On peut naïvement chercher à indiquer que $\mathcal{A}$ peut être n'importe quel algorithme sauf celui là, mais ça ne règle rien, on peut artificiellement créer un algorithme qui est complètement équivalent sans être précisément celui là (par exemple : $\mathcal{A}(vk) : \Pi.\texttt{Sign}(sk, m) \bigoplus (1 - 1 + 0 * 2^{54})$.)

Ainsi encore une fois aucun schéma ne peut vérifier cette propriété de sécurité. Elle est donc trop forte.

La vérité étant qu'a priori $\mathcal{A}$ est conçu indépendamment de (sk, vk). Et donc tout se passe comme si `KeyGen` était exécuté après qu'$\mathcal{A}$ ait été défini.

Remarque A noter qu'on ne peut pas faire le même raisonnement par rapport à $\Pi_{\mathcal{S}}$, qui contrairement à la clef secrète sk est censée être publiquement accessible en amont (en cohérence avec le principe de Kerckhoffs).

Définition 11. *Soit t un entier naturel, $\epsilon \in [0; 1]$ Un schéma $\Pi_{\mathcal{S}}$ est (t, ϵ)-sécurisé si pour tout message $m \in \mathcal{M}$, pour tout algorithme probabiliste $\mathcal{A}$, qui s'exécute en temps au plus t :*

$$\Pr\left[\begin{matrix}(sk, vk) \leftarrow \Pi_{\mathcal{S}}.\texttt{KeyGen}() \\ \Pi_{\mathcal{S}}.\texttt{Verify}(vk, m, \sigma) = 1\end{matrix}\right] \leq \epsilon.$$

A ce moment là, on va chercher à être un peu plus compact et lisible en écrivant cette propriété de manière algorithmique avec le jeu Inforg01 de la figure 1.1 :

D'une manière générale le jeu (parfois appelé expérience) est en fait un algorithme qui prend en entrée le schéma ainsi que l'adversaire et simule un scénario d'attaque censé modéliser une attaque du monde réel.

Et donc la nouvelle définition sera :

Définition 12. *Soit t un entier naturel, $\epsilon \in [0; 1]$. Un schéma $\Pi_{\mathcal{S}}$ est (ϵ, t)-sécurisé si pour tout algorithme probabiliste $\mathcal{A}$, qui s'exécute en temps au plus t et pour tout message $m \in \mathcal{M}$:*

$$\Pr\left[\text{Inforg01}_{\mathcal{A},\Pi,m}() = 1\right] \leq \epsilon.$$

Imaginons qu'on a un schéma $\Pi_{\mathcal{S}}$ qui est sécurisé par rapport à $\mathcal{M}$, l'ensemble des chaînes de caractères, selon la définition 10. On va construire à partir de lui un schéma un peu particulier $\Pi'_{\mathcal{S}}$ identique à $\Pi_{\mathcal{S}}$ à l'exception de l'algorithme de vérification :

$\underline{\mathcal{FB}_{\Pi,m}(vk)}$:	$\underline{\text{Inforg01}_{\mathcal{A},\Pi,m}()}$:	$\underline{\text{Inforg02}_{\mathcal{A},\Pi}()}$:
$\sigma := \epsilon$	$(sk, vk) \leftarrow \Pi.\texttt{KeyGen}()$	$(sk, vk) \leftarrow \Pi.\texttt{KeyGen}()$
Tant que $\Pi.\texttt{Verify}(vk, m, \sigma) \neq 1$	$\sigma \leftarrow \mathcal{A}(vk)$	$(m, \sigma) \leftarrow \mathcal{A}(vk)$
$\sigma \xleftarrow{\$} \cup_{i=0}^{t} \{0,1\}^i$ [a]	Retourner $\Pi.\texttt{Verify}(vk, m, \sigma)$	Retourner $\Pi.\texttt{Verify}(vk, m, \sigma)$
Retourner σ		

a. ici t représente la taille maximale d'une signature

FIGURE 1.1 – Attaquant $\mathcal{F}$orce$\mathcal{B}$rute, et Jeux d'inforgeabilité : Versions 0.1 et 0.2

— $\Pi'_{\mathcal{S}}.\texttt{Verify}(vk, m, \sigma)$: Si $m =$ "Moi l'utilisateur ayant pour clef vk j'ai volé un yaourt à la cantine.", on renvoie 1, sinon on renvoie $\Pi_{\mathcal{S}}(vk, m, \sigma)$.

Si on fixe le message à l'avance

$m =$ "Moi l'utilisateur ayant pour clef A07618E918CD180DDB19911, j'ai volé un yaourt à la cantine."

Il est improbable que l'adversaire réussisse à forger une signature par rapport à ce message, car le seul cas où il est pourra est lorsque la clef de vérification vk devient A07618E918CD180DDB19911 au moment de l'exécution de `KeyGen`, ce qui arrive avec une faible probabilité.

Donc ce schéma $\Pi'_{\mathcal{S}}$ est sécurisé par rapport à la définition 10, or il ne l'est pas d'un point vue concret, car on ne pourra considérer comme authentique aucun aveu de vol de yaourt.

On se rend compte que m ne peut être déterminé en amont. Car on risque de ne pas prendre en considération les messages dépendants des clefs.

On va donc changer le jeu d'inforgeabilité en passant de la version 0.1 à la version 0.2 (cf figure 1.1), et ne plus mentionner de messages dans la définition :

Définition 13. *Soit t un entier naturel, $\epsilon \in [0;1]$. Un schéma $\Pi_{\mathcal{S}}$ est (ϵ, t)-sécurisé si pour tout algorithme probabiliste $\mathcal{A}$, qui s'exécute en temps au plus t :* $\Pr\left[\text{Inforg02}_{\mathcal{A},\Pi}() = 1\right] \leq \epsilon$.

Il est clair que si on veut que les signatures soient comparables avec les signatures manuscrites, on va avoir a priori signer plusieurs documents et certaines signatures seront potentiellement accessibles à autrui.

Il est donc nécessaire si on veut éviter le troisième écueil, c'est à dire que notre modèle ne soit pas trop faible et reste pertinent qu'on donne un accès à des signatures à l'adversaire.

La première tentation est de chercher à coller le plus possible à la réalité en essayant d'imaginer quelles personnes auront accès aux signatures de quelles messages : Évidemment cela va dépendre du contexte : Par exemple si on est dans le cadre de signatures de courriels d'un francophone très poli et soucieux de la grammaire nommé Didier, on va essayer d'imaginer tous les types de messages ayant le format d'un courriel écrit en français qui commence par $S_{\text{debut}} := \{\text{Chère Madame}, \text{Cher Monsieur}\}$, termine par un élément de $S_{\text{fin}} := \{\text{Cordialement Didier}, \text{Respectueusement Didier}\}$. et contient entre ses deux séparateurs : S_3 l'ensemble des textes grammaticalement correct.

On va donc appeler cet ensemble de messages plausibles $\mathcal{M}_{\text{courriels de Didier}}$.

Évidemment, il ne s'agit pas de dire que l'adversaire a accès à toutes les signatures d'éléments de $\mathcal{M}_{\text{courriels de Didier}}$, car ce nombre étant infini, cela contredirait la borne temporelle de son temps de calcul.

On peut d'abord se dire qu'on va chercher à lui donner accès à des messages aléatoires, mais on cela complexifie encore le modèle puisqu'il faut définir une distribution de probabilité.

En fait, à ce stade là le modèle est trop compliqué : Une solution radicale apparaît : laisser à l'adversaire le choix des messages qu'il veut signer. Cela couvre largement le cas d'usage énoncé mais également bien d'autres : Par exemple en envoyant certains messages à Didier qui vont provoquer avec forte probabilité certaines réponse, il est facile d'imaginer que l'adversaire a une prise sur les messages pour lesquels il pourra accéder à leurs signatures.

Par ailleurs toujours dans ce paradigme "simplification-renforcement," on va écarter l'idée d'un ensemble de messages $\mathcal{M}_{\text{courriels de Didier}}$ qui dépendent du contexte pour se baser uniquement sur l'ensemble $\mathcal{M}$ des messages concernés par la propriété de correction du schéma $\Pi_{\mathcal{S}}$. On simplifie tout en étant bien moins dépendant du contexte.

En nous inspirant de la théorie de la complexité, nous allons utiliser un oracle, c'est à dire une petite machine extérieure à l'adversaire et avec laquelle il peut interagir.

$\underline{\text{Inforg03}_{\mathcal{A},\Pi}()}$:	$\underline{\mathcal{A}_1^{O}(vk)}$:
$(sk, vk) \leftarrow \Pi.\texttt{KeyGen}()$ $(m, \sigma) \leftarrow \mathcal{A}^{\texttt{Sign}(sk,\cdot)}(vk)$ Retourner $\Pi.\texttt{Verify}(vk, m, \sigma)$	$m \xleftarrow{\$} \mathcal{M}$ $\sigma \xleftarrow{\$} O(m)$ Retourner (m, σ)

FIGURE 1.2 – Version 0.3 du jeu d'inforgeabilité, et adversaire $\mathcal{A}_1$ structurel de ce jeu

$\underline{\text{Inforg04}_{\mathcal{A},\Pi}()}$:	$\underline{\text{Inforg}_{\mathcal{A},\Pi}(\lambda)}$:	$\underline{OSign_{sk}(m)}$:
$(sk, vk) \leftarrow \Pi.\texttt{KeyGen}()$ $Q := \varnothing$ $(m, \sigma) \leftarrow \mathcal{A}^{OSign_{sk}(\cdot)}(vk)$ Retourner $\Pi.\texttt{Verify}(vk, m, \sigma) \wedge (m \notin Q)$	$(sk, vk) \leftarrow \Pi.\texttt{KeyGen}(1^\lambda)$ $Q := \varnothing$ $(m, \sigma) \leftarrow \mathcal{A}^{OSign_{sk}(\cdot)}(vk)$ Retourner $\Pi.\texttt{Verify}(vk, m, \sigma) \wedge (m \notin Q)$	$Q := Q \cup \{m\}$ Retourner $\Pi.\texttt{Sign}(sk, m)$

FIGURE 1.3 – Versions 0.4, et 1.0 du jeu d'inforgeabilité, avec l'oracle de signatures

En l'occurrence l'oracle de la figure 1.1 utilise la clef de signature (auquel l'adversaire n'a pas accès) pour signer des messages.

Malheureusement, on va alors tomber dans le deuxième écueil, c'est à dire qu'aucun schéma ne peut vérifier cette propriété, car si le schéma est correct (c'est à dire fonctionnel) alors l'adversaire structurel $\mathcal{A}_1$ de la figure 1.2 pourra toujours forger une signature valide.

Doit-on renoncer à la sécurité des signatures numériques ou à leur formalisation ? En fait non, car si on se penche sur ce que fait $\mathcal{A}_1$, on se rend compte que son "attaque" ne devrait pas être considérée comme telle. En effet, il n'y a rien de choquant à ce qu'on puisse retrouver un document signé, si il a effectivement été signé par le passé. On peut argumenter que peut-être ce document n'est plus valide. Mais alors dans la vraie vie, on gère ce genre de chose autrement qu'avec juste une signature. En effet tant qu'on parle de données numériques classiques, il est toujours possible de les copier et de les renvoyer à un autre moment. On doit donc gérer ce genre de chose en datant les messages par exemple, ou en précisant qu'une clef de vérification vk a une "date de péremption".

Si on veut réparer notre modèle, il va donc falloir considérer l'ensemble Q des messages qui ont été signés, puis vérifier que les signatures sont considérées comme des attaques uniquement si elles concernent des messages hors de cet ensemble.

On va donc utiliser le jeu Inforg04 de la figure 1.3 :

Définition 14. *Soit t un entier naturel, $\epsilon \in [0;1]$. Un schéma $\Pi_{\mathcal{S}}$ est (ϵ, t)-sécurisé si pour tout algorithme probabiliste $\mathcal{A}$, qui s'exécute en temps au plus t :* $\Pr\left[\text{Inforg04}_{\mathcal{A},\Pi}() = 1\right] \leq \epsilon$.

A présent que nous avons une définition qui semble satisfaire les trois conditions qu'on s'étaient fixées, le lecteur pourra se familiariser avec l'exercice suivant :

Exercice 1. *Soit $(\epsilon, \epsilon') \in [0;1]^2, (t, t') \in \mathbb{N}^2$. Soit Π_S, et Π'_S deux systèmes de signatures numériques respectivement $(\epsilon, t)-$sécurisé, et (ϵ', t')-sécurisé par rapport à la définition 14.*

On pose $\Pi_{concatet}$:

$\texttt{KeyGen} : (sk, vk) \leftarrow \Pi_S.\texttt{KeyGen}(), (sk', vk') \leftarrow \Pi'_S.\texttt{KeyGen}()$. *Retourner* $((sk, sk'), (vk, vk'))$.

$\texttt{Sign}((sk, sk'), m) : \sigma \leftarrow \Pi_S.\texttt{Sign}(sk, m); \sigma' \leftarrow \Pi'_S.\texttt{Sign}(sk', m)$. *Renvoyer* (σ, σ').

$\texttt{Verify}((vk, vk'), m, (\sigma, \sigma'))$ *: Retourner* $\Pi_S.\texttt{Verify}(vk, m, \sigma) \wedge \Pi'_S.\texttt{Verify}(vk', m, \sigma')$.

Montrer qu'il existe une constante K tel que $\Pi_{concatet}$ est $(\max(\epsilon, \epsilon'), \min(t, t') - K)$-sécurisé.

1.3 De la sécurité concrète à la sécurité asymptotique : Universalisons la définition

Si lorsqu'on utilise un produit cryptographique directement pour une application concrète la sécurité concrète est sans doute la plus pertinente. Mais lors d'une première analyse théorique, on souhaiterait avoir quelque chose de plus universel, c'est à dire de pouvoir se débarrasser au maximum des paramètres ϵ et t.

On part donc de ce genre de définition :

Définition 15 (Sécurité Concrète)**.** *Un schéma est dit (t, ϵ)-sécurisé si pour tout algorithme ayant un temps d'exécution t, la probabilité que ce dernier "casse" le schéma est bornée par ϵ.*

En faisant l'exercice 1, on se rend compte que l'évaluation de la sécurité de combinaisons de plusieurs schéma sécurisés peut s'avérer assez complexe à définir notamment à cause de l'éternel problème de la non universalité d'une métrique temporelle. Pour remédier à ça, on va s'inspirer de ce qui se fait usuellement en théorie de la complexité : Si l'évaluation du temps d'un algorithme pseudo-code est très dépendant du modèle de calcul considéré ou du langage de programmation dans lequel il sera codé, si on fait varier la taille de l'entrée et qu'on regarde la courbe du temps d'exécution de l'algorithme, alors quel que soit le langage ou modèle choisit le temps de calcul sera le même *à une constante multiplicative près.*

Ainsi tous les algorithmes linéaires sont considérés comme efficaces, car on sait qu'ils n'auront pas de problème à être exécutés par une machine capable de lire l'entrée : Évidemment si la constante multiplicative est énorme, ce ne sera pas le cas, mais aucun algorithme conçu *naturellement* (c'est à dire pas un exemple pathologique) n'a eu une constante qui rend un calcul inaccessible.

Les algorithmes linéaires vérifient la bonne propriété qu'ils se combinent relativement bien : si une procédure consiste en la concaténation de trois fois l'exécution d'un algorithme linéaire puis une fois un autre algorithme linéaire, alors la procédure sera elle même linéaire.

Malheureusement, les théoriciens de la complexité se sont vite rendus compte que certaines combinaisons ne fonctionnaient pas. Par exemple si je fais un nombre d'appel linéairement grand à une fonction qui n'a pas un coût constant, mais un coût linéaire ou même logarithmique, la procédure ne sera plus un algorithme linéaire.

Ainsi, il a été décidé de considérer les calculs de temps polynomial en la taille de l'entrée comme efficace. Si en faisant ça on se démarque beaucoup plus de la pratique qu'en négligeant seulement les constantes multiplicatives, il se trouve qu'en *première analyse*, cela reste pertinent.

Ainsi, on peut donc définir ce qu'est un attaquant efficace et évacuer toute notion floue concernant le temps de la définition de sécurité, en disant qu'un attaquant efficace est un algorithme polynomial. A noter qu'ici le fait de prendre en compte *trop* d'algorithmes, c'est à dire des algorithmes avec des trop grosses constantes ou de trop gros exposants n'est pas du tout un problème, cela ne fait que renforcer notre définition de sécurité [2].

Définition 16. *Soit $\epsilon \in [0; 1]$. Un schéma $\Pi_{\mathcal{S}}$ est ϵ-sécurisé si pour tout algorithme probabiliste borné polynomialement $\mathcal{A}$:* $\Pr\left[\text{Inforg04}_{\mathcal{A},\Pi}() = 1\right] \leq \epsilon$.

Mais cette définition n'est pas consistante, en effet, à partir du moment où le schéma est fixé, on a donc une borne sur la taille clef publique, les tailles de signatures (par rapport au fait qu'on impose que l'exécution de Π.`Sign` soit rapide), et un temps d'exécution *constant* de Π.`Verify`$(vk, m, \cdot)$, donc l'attaquant $\mathcal{FB}$ de la figure 1.1 pour n'importe quel message fixé $m \in \mathcal{M}$ est en fait un attaquant en temps constant (qui n'est même pas défini pour des entrées à partir d'une certaine taille), et donc va casser à nouveau *universellement* la propriété d'inforgeabilité.

A ce moment là, il est très tentant de revenir sur notre définition originelle de schéma de signature.

Et nous allons donc céder à la tentation en prenant le risque de se détacher des schémas de signature du monde réel (mais sans perdre des yeux ce risque).

En fait on va essayer de se sortir par le haut de ce problème ("tout ce qui ne tue pas..." tout ça, tout ça). Le problème étant que tant que les signatures auront une taille bornée l'attaquant $\mathcal{FB}$ mettra en danger notre définition. Il est donc nécessaire que la taille de la sortie de Π.`Sign` grandissent. On va donc laisser tomber la définition pas assez formelle "s'exécute en moins d'une seconde", et dire que Π.`Sign` est aussi un algorithme PPT(c'est à dire un algorithme probabiliste qui s'exécute en un temps polynomial).

Mais ça ne suffit pas car son entrée est de taille bornée (si on fixe un message). En effet vk est générée par Π.`KeyGen`, qui elle même doit s'exécuter en moins d'une seconde. Il est donc également nécessaire que la taille de vk puisse grossir et donc que Π.`KeyGen` soit considéré aussi comme un PPT.

2. Enfin pour être plus précis cela pourrait être un problème si aucun schéma ne vérifierait la propriété à cause de l'existence d'un attaquant universel peu réaliste (car avec un exposant très gros par exemple), mais divulgachons : ça n'arrivera pas.

Sauf que Π.`KeyGen` ne prend pas d'entrée dans la définition actuelle. Nous allons donc lui donner artificiellement une entrée dont la taille est variable. Comme seule la taille de l'entrée est pertinente pour cet algorithme on décide de lui donner en entrée uniquement des 1 (par convention).

Ainsi les tailles de clefs et de signatures vont donc grossir polynomialement [3] avec le nombre de 1 de l'entrée de Π.`KeyGen`.

Du coup, comme la taille de ses entrées (aussi bien la clef de vérification que la signature) ne sont plus bornées par une constante, il n'est plus possible que le temps d'exécution de Π.`Sign` soit borné par une constante, et on va donc logiquement se contenter d'exiger qu'il soit polynomialement borné en la taille de l'entrée (et donc soit un PPT), ce qui achève la formalisation de ce qu'est un schéma de signatures numériques.

Définition 17. *Un schéma de signature numérique sur l'ensemble de messages $\mathcal{M}$ est un triplet d'algorithmes PPT :*

- `KeyGen`(1^λ) *qui prend en entrée un paramètre de sécurité écrit en unaire et qui renvoie une paire de clefs (sk, vk), la première clef est une clef de signatures, tandis que vk est une clef de vérification.*
- `Sign`(sk, m) *qui prend en entrée une clef de signatures sk un message $m \in \mathcal{M}$, et renvoie une signature σ.*
- `Verify`(vk, m, σ) *qui prend en entrée une clef de vérification, un message m, une signature σ et renvoie un bit $b \in \{0, 1\}$.*

On doit donc légèrement modifier le jeu de sécurité pour introduire la nouvelle entrée de `KeyGen`, mais à présent qu'on a dit que notre adversaire était polynomialement borné, et pas seulement borné par une constante t, il sera toujours possible de créer des adversaires exhaustifs pour certains lambdas précis. Leur caractérisation de "polynomialement borné" n'ayant qu'une signification asymptotique. On va donc regarder leur probabilité de succès asymptotiquement.

Définition 18. *Soit $\epsilon \in [0; 1]$. Un schéma $\Pi_{\mathcal{S}}$ est ϵ-sécurisé si pour tout algorithme probabiliste borné polynomialement $\mathcal{A}$:* $\lim_{\lambda \to \infty} \left(\Pr\left[\text{Inforg}_{\mathcal{A},\Pi}(\lambda) = 1\right]\right) \leq \epsilon$.

L'introduction de ce paramètre de sécurité peut être vu comme une abstraction de théoricien qui divague car trop éloigné du "terrain". Mais il s'avère qu'en réalité cette considération théorique correspond à une réalité concrète : Si on met plus de 1, les tailles des objets produits (ici les signatures) vont potentiellement augmenter et donc les adversaires (superpolynomiaux si la définition de sécurité est vérifiée) qui cassent le schéma vont donc voir leur temps de calcul monter.

Et si on est pas convaincu par les considération asymptotiques, mais qu'on veut quand même pouvoir dire des choses par rapport à la paramétrisation du schéma on pourra prendre cette définition :

Définition 19. *Soit $\epsilon \in [0; 1]^{\mathbb{N}\times\mathbb{N}}$. Un schéma $\Pi_{\mathcal{S}}$ est ϵ-sécurisé si pour tout algorithme probabiliste $\mathcal{A}$, qui s'exécute en temps au plus t :* $\Pr\left[\text{Inforg}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \epsilon(t, \lambda)$ [4].

A noter qu'augmenter ce paramètre n'est pas anodin, il va *a priori* augmenter le temps d'exécution des algorithmes du système. Mais comme par définition une fonction superpolynomiale (qui serait le temps de calcul d'un attaquant efficace) écrase asymptotiquement une fonction polynomiale, on sait qu'il existe un λ où les temps de calcul des algorithmes du système seront encore réalisable, alors que ceux de l'attaquant ne le seront plus [5].

On se retrouve donc avec un paramètre qui fixe le niveau de sécurité (par rapport à la variété des contextes dont on a parlé plus tôt dans ce chapitre). On appelle donc cette grandeur le paramètre de sécurité.

Exercice 2. *Soit $(\epsilon, \epsilon') \in [0; 1]^2$. Soit Π_S, et Π'_S deux systèmes de signatures numériques respectivement ϵ-sécurisé, et ϵ'-sécurisé par rapport à la définition 18.*

On pose $\Pi_{concatou}$:

3. en pratique ce sera souvent linéaire et quasiment jamais plus de quadratique

4. On peut affiner le calcul de ϵ, en prenant en compte la variété des ressources que peut utiliser $\mathcal{A}$ et pas seulement le temps de calcul. Par exemple on pourrait considérer m les ressources en mémoire qu'il utilise, ainsi que q le nombre d'appels à oracles *OSign*, ϵ serait alors dans $[0; 1]^{\mathbb{N}^4}$.

5. En théorie, ce λ peut ne pas exister si les constantes pour les algorithmes du système sont tellement catastrophiques, que leur vitesse d'exécution est supérieur à celle d'un potentiel attaquant pour des paramètres où ils sont eux même déjà complètement inefficaces, mais généralement ça ne pose pas de problème, les courbes se croisant bien assez tôt.

$\underline{\text{InforgMU}_{\mathcal{A},\Pi}(\lambda)}$: $C \leftarrow U \leftarrow Q \leftarrow \varnothing$ $(vk, m, \sigma) \leftarrow \mathcal{A}^{OCreate(),OCorrupt(\cdot),OSign(\cdot,\cdot)}$ Retourner $\Pi.\texttt{Verify}(vk, m, \sigma)$ $\wedge ((vk, m) \notin Q) \wedge ((vk, \cdot) \in U)$	$\underline{OCreate()}$: $(sk, vk) \leftarrow \Pi.\texttt{KeyGen}(1^\lambda)$ $U := U \cup \{(sk, vk)\}$ Retourner vk $\underline{OCorrupt(vk)}$: $Q := Q \cup (\{vk\} \times \mathcal{M})$ Si $\exists sk/(sk, vk) \in U$ Retourner sk Sinon Retourner $\perp$	$\underline{OSign(vk, m)}$: $Q := Q \cup \{(vk, m)\}$ Si $\exists sk/(sk, vk) \in U$ $\sigma \leftarrow \Pi.\texttt{Sign}(sk, m)$ Retourner σ Sinon Retourner $\perp$

FIGURE 1.4 – Jeu de sécurité d'inforgeabilité dans un contexte multi-utilisateurs.

$\texttt{KeyGen}(1^\lambda) : (sk, vk) \leftarrow \Pi_S.\texttt{KeyGen}(1^\lambda), (sk', vk') \leftarrow \Pi'_S.\texttt{KeyGen}(1^\lambda)$. *Retourner* $((sk, sk'), (vk, vk'))$.
$\texttt{Sign}((sk, sk'), m) : \sigma \leftarrow \Pi_S.\texttt{Sign}(sk, m); \sigma' \leftarrow \Pi'_S.\texttt{Sign}(sk', m)$. *Retourner* (σ, σ').
$\texttt{Verify}((vk, vk'), m, (\sigma, \sigma'))$ *: Retourner* $\Pi_S.\texttt{Verify}(vk, m, \sigma) \vee \Pi'_S.\texttt{Verify}(vk', m, \sigma')$.
Montrer que $\Pi_{concatou}$ *est* $(\epsilon+\epsilon')$*-sécurisé. Méditer sur le fait qu'a priori, on ne pourra pas avoir mieux (en construisant des exemples pathologiques).*

Si le schéma de l'exercice 2 peut paraître artificiel et n'a aucune intérêt pratique, il représente un genre de combinaisons assez courantes où l'on combine deux schémas de sécurité (souvent de types différents) afin de créer un schéma qui vérifiera de nouvelles fonctionnalités (ici ce n'est pas le cas [6]).

On a donc un problème du fait de la paramétrisation de la définition. En effet si on considère qu'il y a un seuil ϵ_{seuil} au delà duquel les propriétés. Alors, je vais avoir besoin d'établir une sécurité plus forte sur Π_S, Π'_S qu'une ϵ_{seuil}-sécurité (par exemple une $\frac{\epsilon_{\text{seuil}}}{2}$-sécurité).

Ce qui implique *a priori* une nouvelle preuve de sécurité, voir un nouveau schéma. Cela peut paraître un peu frustrant de devoir reconstruire un schéma alors qu'on a déjà une définition paramétrique pour notre schéma.

On pourrait se rabattre sur la définition 19, en supposant que la fonction epsilon est la plus fine possible, on pourra le faire dans certains contextes, mais on perd en capacité d'analyse. Et surtout on s'éloigne de notre objectif d'universalité de la définition.

La solution qui apparaît la plus canonique est donc la suivante : ne considérer que les schémas 0-sécurisés :

Définition 20. *Un schéma* $\Pi_{\mathcal{S}}$ *est sécurisé si pour tout algorithme probabiliste borné polynomialement* $\mathcal{A}$*, il existe* $\epsilon \in [0; 1]^{\mathbb{N}}$ *une suite qui tend vers* 0 *tel que :* $\Pr\left[\text{Inforg}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \epsilon(\lambda)$.

Avec cette nouvelle définition on a enfin une notion "universelle" de la sécurité. C'est à dire qu'on a enfin déparamétré la notion de sécurité.

A présent essayons de réfléchir à un cas d'usage multi-utilisateurs de notre schéma de sécurité :

On peut regarder deux versions de définitions de sécurité :

Définition 21. *Soit* $\epsilon \in [0; 1]^{\mathbb{N}^3}$*. Un schéma* $\Pi_{\mathcal{S}}$ *est* ϵ*-sécurisé dans le contexte multi-utilisateur si pour tout algorithme probabiliste* $\mathcal{A}$*, qui s'exécute en temps au plus* t *et fait au plus* q *requêtes* OCreate *:*

$$\Pr\left[\text{InforgMU}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \epsilon(t, q, \lambda).$$

Définition 22. *Un schéma* $\Pi_{\mathcal{S}}$ *est sécurisé dans le contexte multi-utilisateurs si pour tout algorithme probabiliste polynomiales* $\mathcal{A}$*, il existe* $\epsilon \in [0; 1]^{\mathbb{N}}$ *une suite qui tend vers* 0 *tel que :*

$$\Pr\left[\text{InforgMU}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \epsilon(\lambda).$$

Exercice 3. *Montrer que si* $\Pi_{\mathcal{S}}$ *est* ϵ*-sécurisé selon la définition 19, alors il est* $q\epsilon$*-sécurisé dans le contexte multi-utilisateurs selon la définition 21.*

Indice : On pourra essayer de deviner quel utilisateur parmi les q *du système va se faire effectivement attaquer et imaginer que celui ci a les clefs du système originel du jeu* Inforg*, tandis que les* $(q-1)$ *autres utilisateurs seront simulés artificiellement.*

6. Mais on peut toujours y voir une propriété "si-je-perds-une moitié-de-clef-je-peux-quand-même-faire-des-signatures-valides"

On se rend alors compte que l'on peut avoir un schéma sécurisé selon la définition 16. En effet le nombre de requêtes à *OCreate* pouvant être polynomial (ou dit autrement, le nombre d'utilisateurs de notre système pouvant être polynomial) on a aucune garantie que $q(\lambda) \cdot \epsilon(\lambda)$ tende vers zéro.

Être stable par addition n'est donc pas suffisant pour notre ensemble de fonctions ϵ pertinentes. On veut aussi être stable par multiplication par un polynôme en λ.

On va donc introduire l'ensemble des fonctions négligeables.

Définition 23. *Une fonction f de $\mathbb{N} \to \mathbb{R}$ est négligeable si pour tout polynôme P, f est asymptotiquement bornée devant $\left(\frac{1}{P(n)}\right)_{n\in\mathbb{N}}$*[7].

Propriété 1. *Si on pose P, l'ensemble des fonctions polynomiales. L'ensemble des fonctions négligeables est un P-module.*

Preuve laissée en exercice pour le lecteur. □

On va donc avoir comme définition asymptotique définitive :

Définition 24. *Un schéma $\Pi_{\mathcal{S}}$ est sécurisé si pour tout algorithme probabiliste polynomialement borné $\mathcal{A}$: $\Pr\left[\text{Inforg}_{\mathcal{A},\Pi}(\lambda) = 1\right]$ est une fonction négligeable en λ.*

1.4 Conclusion

Pour conclure, on va retenir deux définitions : une concrète à paramètres (la définition 19) et une définition plus universelle, et plus théorique qui prend son sens asymptotiquement (la définition 24).

La première fait office de définition "finale", et nécessite d'avoir en tête ce qu'on prend en compte comme capacité de calcul assez précisément. Tandis que la seconde est une définition qui sert de première ébauche, afin d'établir la sécurité d'un schéma en omettant le choix des paramètres, et les capacités de calcul de l'adversaire prise en compte.

A noter que si la vision générique des définitions concrètes et universelles va s'appliquer dans énormément de contextes très différents de celui des signatures numériques. Le jeu de sécurité (appelé aussi expérience), lui en revanche doit être repensé pour chaque contexte. Et il n'y aura jamais de formule mathématique pour construire de tels jeux (de la même manière qu'il n'existe pas de formule mathématique pour construire des modèles physiques). Afin de s'exercer à la création de nouveaux modèles de sécurité, le lecteur pourra réfléchir à l'exercice suivant :

Définition 25. *Un système de mot de passe est un triplet d'algorithmes PPT* $(\texttt{Init}, \texttt{Add}, \texttt{Verify})$ *:*

- $\texttt{Init}\left(1^{\lambda}\right)$ *prend en entrée un entier écrit en unaire, et renvoie une base de données B.*
- $\texttt{Add}\left(B, id, mdp\right)$ *prend en entrée une base de données, un identifiant et un mot de passe et renvoie une base de données, ainsi qu'un message m.*
- $\texttt{Verify}\left(B, id, mdp\right)$ *prend en entrée une base de données, un identifiant et un mot de passe et renvoie un message m.*

Exercice 4. *Définir une ou plusieurs propriétés de correction, ainsi que des propriétés de sécurité modélisant ce qu'on attend d'un tel système dans le monde réel.*

7. Par rapport à la définition usuelle de négligeable, on peut le traduire ainsi, une fonction est négligeable en λ, si elle est négligeable en $+\infty$ par rapport à $\left(\frac{1}{P(\lambda)}\right)_{\lambda\in\mathbb{N}}$ pour tout polynôme P.

Chapitre 2

Chiffrement à clef secrète (7 heures)

A présent que nous sommes au point sur le formalisme des jeux de sécurité, nous allons nous intéresser à la problématique historique de la cryptographie : La confidentialité.

Il va s'agir de dissimuler de l'information sur des espaces non sécurisés (c'est à dire accessibles à des entités auxquels on ne fait pas confiance a priori).

Nous allons donc nous focaliser sur la primitive vedette de la cryptographie : Le chiffrement.

On peut retenir deux contextes très génériques dans lequel le chiffrement d'une donnée s'applique : Celui des communications sur des canaux non sécurisés (métaphore du convoyeur de fonds), celui du stockage de données sur un serveur a priori accessible à des entités auxquelles on ne fait pas confiance (métaphore du casier à la piscine). Dans les deux cas en plus de l'exigence de sécurité, on va bien sûr souhaiter l'intégrité de la donnée (de la même manière qu'un incinérateur qui vérifie une bonne propriété de sécurité en tant que casier n'est pas adapté pour servir de casier).

2.1 Premières définitions

Pour repartir sur des bases saines, et en remarquant que la problématique de la confidentialité est plus complexe à définir que l'authentification, commençons par définir les algorithmes qui caractérisent un système de chiffrement. Soit $\mathcal{M}$ l'ensemble des messages en clair.

Définition 26. *Un schéma de chiffrement est un triplet d'algorithmes PPT :* $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$.
$\texttt{KeyGen} : \emptyset \to \mathcal{K}$
$\texttt{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$
$\texttt{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$
où les éléments de $\mathcal{K}$ et $\mathcal{C}$ sont respectivement appelés clefs de chiffrement (qu'on peut parfois appeler tout simplement clefs selon le contexte) ou chiffres, et chiffrés ou cryptogrammes.

Comme pour les signatures numériques vues dans le chapitre 1, avant de discuter de sécurité, il nous faut parler de ce qu'on attend comme propriété minimale afin qu'il puisse être utiliser en pratique :

Définition 27. *On dit que le schéma est correct si et seulement si* $\forall (k, m) \in \mathcal{K} \times \mathcal{M}$ *:* $\texttt{Dec}_k(\texttt{Enc}_k(m)) = m$.

2.1.1 La sécurité parfaite

On peut partir sur une définition informelle qui dit qu'un schéma Π est sécurisé si on ne déduit aucune information sur un texte clair en voyant un chiffré d'un texte clair.

Ce qui d'un point de vue probabiliste peut s'exprimer de cette manière là :

Définition 28 (Sécurité de Shannon)**.** *On dit qu'un schéma Π est confidentiel au sens de Shannon si pour toute distribution d sur $\mathcal{M}$, $\forall m \in \mathcal{M}$ et $\forall c \in \mathcal{C}$,*

$$\Pr_{k \leftarrow \Pi.\texttt{KeyGen}; m' \xleftarrow{d} \mathcal{M}} [m = m' | \Pi.\texttt{Enc}_k(m') = c] = \Pr_{m' \xleftarrow{d} \mathcal{M}} [m = m'] .$$

Expliquons un peu en détail cette définition : la distribution d correspond au contexte sur lequel on applique notre méthode de chiffrement. Ce contexte est supposé connu de tous (et en particulier d'un curieux qui n'est pas censé avoir accès à la donnée). Par exemple si le message m' est *publiquement connu* pour être extrait d'une correspondance épistolaire entre deux footballeurs le lendemain d'un match, il est plus probable que le message termine par le nom du rédacteur (comme c'est l'usage pour un courrier), et que les mots "ballon", "but", "tir" soient mentionnés dans le message. Ceci est une information *a priori*, considérée comme non sensible. On va donc chercher à ce qu'une personne qui n'a aucune information sur la clef de chiffrement k, même si il sait que le chiffré de m' par la clef k est le cryptogramme c. Cela revient à dire que le distribution de m' ne change pas même si l'on connaît c, donc que

$$\Pr_{k\leftarrow\Pi.\mathtt{KeyGen};m'\xleftarrow{d}\mathcal{M}}[m=m'|\Pi.\mathtt{Enc}_k(m')=c]=\Pr_{m'\xleftarrow{d}\mathcal{M}}[m=m'].$$

On peut simplifier cette définition grâce à la définition de la sécurité parfaite qui permet d'évacuer la notion de distribution des messages :

Définition 29 (Sécurité parfaite)**.** *On dit qu'un schéma* Π *est parfaitement sécurisé si*

$$\forall m_1,m_2\in\mathcal{M},c\in C,\Pr_{k\leftarrow\Pi.\mathtt{KeyGen}}[\Pi.\mathtt{Enc}_k(m_1)=c]=\Pr_{k\leftarrow\Pi.\mathtt{KeyGen}}[\Pi.\mathtt{Enc}_k(m_2)=c].$$

Propriété 2. *Les définitions 28, et 29 sont équivalentes.*

On suppose qu'on a un schéma Π parfaitement sécurisé. Soit d une distribution sur $\mathcal{M}$. Soit $m,,m_2,c\in\mathcal{M}\times\mathcal{M}\times\mathcal{C}$. Sans perdre de généralité $\exists m',k\in\mathcal{M}\times\mathcal{K}$ tel que $\Pr[\Pi.\mathtt{Enc}_k(m')=c]\neq 0$. Donc sans perdre de généralité $k\in\Pi.\mathtt{KeyGen}()$. Donc sans perdre de généralité $\Pr_{m'\xleftarrow{d}\mathcal{M},k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m')=c]\neq 0$. On peut donc faire le calcul suivant :

$$\begin{aligned}
&\Pr_{m'\xleftarrow{d}\mathcal{M},k\leftarrow\Pi.\mathtt{KeyGen}()}[m=m'|\Pi.\mathtt{Enc}_k(m')=c]\\
=&\frac{\Pr_{m'\xleftarrow{d}\mathcal{M},k\leftarrow\Pi.\mathtt{KeyGen}()}[m=m'\wedge\Pi.\mathtt{Enc}_k(m')=c]}{\Pr_{m'\xleftarrow{d}\mathcal{M},k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m')=c]}\\
=&\frac{\Pr_{m'\xleftarrow{d}\mathcal{M},k\leftarrow\Pi.\mathtt{KeyGen}()}[m=m'\wedge\Pi.\mathtt{Enc}_k(m)=c]}{\Pr_{m'\xleftarrow{d}\mathcal{M},k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m')=c]}
\end{aligned}$$

Comme l'aléa de la clef et de $\mathtt{Enc}$ est indépendant de celui de d :

$$=\frac{\Pr_{m'\xleftarrow{d}\mathcal{M}}[m=m']\Pr_{k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m)=c]}{\Pr_{m'\xleftarrow{d}\mathcal{M},k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m')=c]}. \tag{2.1}$$

Or

$$\begin{aligned}
&\Pr_{m'\xleftarrow{d}\mathcal{M},k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m')=c]\\
&=\sum_{m''\in\mathcal{M}}\Pr_{m'\xleftarrow{d}\mathcal{M}}[m''=m']\Pr_{k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m'')=c]\\
&\text{par sécurité parfaite on en déduit :}\\
&=\sum_{m''\in\mathcal{M}}\Pr_{m'\xleftarrow{d}\mathcal{M}}[m''=m']\Pr_{k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m)=c]\\
&=\sum_{m''\in\mathcal{M}}\Pr_{m'\xleftarrow{d}\mathcal{M}}[m''=m']\Pr_{k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m)=c]\\
&=\Pr_{k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m)=c]\left(\sum_{m''\in\mathcal{M}}\Pr_{m'\xleftarrow{d}\mathcal{M}}[m''=m']\right)\\
&=\Pr_{k\leftarrow\Pi.\mathtt{KeyGen}()}[\Pi.\mathtt{Enc}_k(m)=c].
\end{aligned} \tag{2.2}$$

En combinant (2.1), et (2.2), on en déduit ce qu'il fallait démontrer.

On suppose à présent que Π est sécurisé au sens de Shannon.
L'égalité (2.1) reste valide :

$$\Pr_{m' \xleftarrow{d} \mathcal{M}, k \leftarrow \Pi.\mathtt{KeyGen}()} [m = m' | \Pi.\mathtt{Enc}_k(m') = c] = \frac{\Pr_{m' \xleftarrow{d} \mathcal{M}} [m = m'] \Pr_{k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m) = c]}{\Pr_{m' \xleftarrow{d} \mathcal{M}, k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m') = c]}.$$

Donc par sécurité de Shannon :

$$\Pr_{m' \xleftarrow{d} \mathcal{M}} [m = m'] = \frac{\Pr_{m' \xleftarrow{d} \mathcal{M}} [m = m'] \Pr_{k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m) = c]}{\Pr_{m' \xleftarrow{d} \mathcal{M}, k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m') = c]}.$$

Donc si m est dans le support de d :

$$1 = \frac{\Pr_{k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m) = c]}{\Pr_{m' \xleftarrow{d} \mathcal{M}, k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m') = c]}.$$

On pose d la distribution équiprobable sur $\{m, m_2\}$. On a alors :

$$\Pr_{k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m) = c] = \frac{1}{2} \Pr_{k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m) = c] + \frac{1}{2} \Pr_{k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m_2) = c].$$

On en déduit

$$\Pr_{k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m) = c] = \Pr_{k \leftarrow \Pi.\mathtt{KeyGen}()} [\Pi.\mathtt{Enc}_k(m_2) = c].$$

Et ce pour tout $(m, m_2) \in \mathcal{M}^2$. Donc Π est parfaitement sécurisé. □

Exercice 5 (Théorème de Shannon)**.** *Montrer que dans un cadre où $\mathcal{K} = \mathcal{M} = \mathcal{C}$ sont finis, le chiffrement est parfaitement sécurisé, et correct si et seulement si :*

1. *L'algorithme de génération de clefs consiste à tirer uniformément une clef.*
2. *Il existe pour chaque couple $(m, c) \in \mathcal{M} \times \mathcal{C}$ une unique clef k tel que $\mathtt{Enc}_k(m) = c$.*

Par la suite dans ce chapitre on se restreindra au cas où $\mathtt{KeyGen}\left(1^\lambda\right) := \mathrm{Unif}\left(\{0,1\}^\lambda\right)$. En fait, efficacité mise à part, on ne perd pas en généralité avec cette considération si on imagine qu'il s'agit des bits d'aléa de la fonction `KeyGen`, et que la clef effective peut donc être recalculée à chaque chiffrement [1].

Le Chiffrement de Vernam A présent, nous allons définir dans la figure 2.1 un premier système de chiffrement où les messages sont des chaînes de bits de taille λ : le masque jetable appelé aussi Chiffrement de Vernam ou one-time-pad en anglais. Le concept est simple, on pose $\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^\lambda$ [2]. L'algorithme de chiffrement, respectivement de déchiffrement, consiste à "xorer" le message, respectivement le cryptogramme, avec une clef qui a été générée uniformément.

Par exemple, si on veut chiffrer le mot binaire 011011 avec la clef 110011, on produit le cryptogramme 101000, qui si il est de nouveau xoré avec le clef redevient le message originel 011011.

Théorème 1. *Le chiffrement de Vernam est un système de chiffrement* correct *et* parfaitement sécurisé.

Preuve laissée en exercice pour le lecteur. □

1. On peut donc réinterpréter le paramètre de sécurité comme l'entropie de la clef secrète.
2. Le schéma est facilement adaptable en remplaçant $\{0,1\} = \mathbb{F}_2$ par n'importe quel groupe abélien $\mathbb{Z}/N\mathbb{Z}$. En particulier pour $N = 26$, on se retrouve tout naturellement à la version historique du système appliqué à des mots constitués des lettres de l'alphabet latin.

Π_{MJ} :	Soit G, un GPA.	Soit F, une FPA.
$\texttt{KeyGen}\left(1^\lambda\right) : k \xleftarrow{\$} \{0,1\}^\lambda$	Π_1 :	Π_2 :
$\texttt{Enc}(k, m \in \{0,1\}^\lambda)$:	$\texttt{KeyGen}(1^\lambda) : k \xleftarrow{\$} \{0,1\}^\lambda$	$\texttt{KeyGen}\left(1^\lambda\right) : k \xleftarrow{\$} \{0,1\}^\lambda$
Retourner $(k \oplus m)$	$\texttt{Enc}(k, m \in \{0,1\}^{\ell(\lambda)}) : G(k) \oplus m$	$\texttt{Enc}(k, m \in \{0,1\}^{\ell(\lambda)})$:
$\texttt{Dec}\,(k, c)$:	$\texttt{Dec}(k, c)$: Retourner $(G(k) \oplus c)$	$r \xleftarrow{\$} \{0,1\}^\lambda$
Retourner $(k \oplus c)$		Retourner $(r, F_k(r) \oplus m)$
		$\texttt{Dec}\,(k, (c_1, c_2))$:
		Retourner $(F_k(c_1) \oplus c_2)$

FIGURE 2.1 – Systèmes de chiffrement

Un problème de taille On pourrait se dire que le problème de la confidentialité est donc complètement résolu avec le système de Vernam. Mais en réalité il suppose une taille de clef aussi grosse que celle du message (donc un partage d'information secrète en amont a priori aussi "difficile" que le partage du secret qu'on cherche un résoudre). Cela peut avoir du sens dans un contexte où on a accès à un canal sécurisé, avant l'envoi du message en lui même. Par exemple, durant la guerre froide les dirigeant des états-unis et de l'union des républiques socialistes soviétiques communiquaient via une ligne (surnommé le téléphone rouge) où les communications étaient chiffrés avec un chiffrement de Vernam dont les clefs avaient été communiquées en amont par valises diplomatiques (considérées comme canal sécurisé).

Mais dans de très nombreux contextes, cela n'est pas satisfaisant, si je dois chiffrer des données sur un ordinateur collectif, puis garder avec moi un disque dur de la même capacité (au bit près), c'est aussi efficace d'effacer complètement les données après avoir copier les données sur mon disque dur amovible.

Exercice 6. *Vous rencontrez quelqu'un qui pratique cette stratégie (chiffrer vernamiquement ses données sur un ordinateur collectif, puis se balader avec la clef de chiffrement sur son disque dur externe), vous lui faîtes la remarque ci-dessus, et il vous rétorque "Oui, mais si je perds mon disque dur externe, au moins mes données sont toujours sur l'ordinateur collectif, alors qu'avec ce que tu dis, je perds tout".*

1. *Expliquer à cette personne pourquoi elle se trompe lourdement.*
2. *L'individu semble convaincu par vos explications, il s'écrit, "J'ai une idée ! Je vais compresser mes données, puis les chiffrer vernamiquement, ainsi la clef sera beaucoup plus petite, et je pourrais la mettre sur une clef usb, ce qui est beaucoup plus pratique à transporter que toutes mes données sur un gros disque dur externe." Expliquer avec pédagogie et sans condescendance déplacée à cette personne que même là, le chiffrement ne lui sert à rien.*
3. *Trouver quand même un scénario (autre que la perte du disque dur) qui justifie qu'il s'y prenne comme ça.*

En fait, on souhaite que les clefs soient de taille plus petite que les messages. Ce qui si on évacue les questions de compression d'information (cf question 2 de l'exercice 6) implique par rapport aux cardinaux : $|\mathcal{M}| > |\mathcal{K}|$. Malheureusement, on peut montrer le résultat suivant :

Théorème 2. *Si le système est correct et parfaitement sécurisé on a $|\mathcal{M}| \leq |\mathcal{K}|$.*

Lemme 1. *Si le système est correct alors on peut supposer sans perdre de généralités que $\Pi.\texttt{Dec}_k$ est déterministe.*

Soit $c \in \mathcal{C}$. 1er cas, il existe $m_0 \in \mathcal{M}$, et un aléa r tel que $\Pi.\texttt{Enc}_k(m_0; r) = c$. Par correction $\Pr[\Pi.\texttt{Dec}_k(c) = m_0] = 1$. Donc on utilise pas d'aléa dans ce cas. Deuxième cas, $c \notin \Pi.\texttt{Enc}_k(\mathcal{M})$, on peut décider de déterminiser l'algorithme sans avoir d'impact ni sur la correction ni sur la sécurité parfaite (qui ne dit rien sur $\Pi.\texttt{Dec}$). On en déduit qu'on peut considérer que $\Pi.\texttt{Dec}$ est déterministe. □

Sans perdre de généralité, on peut supposer qu'il existe $(m_0, c) \in \mathcal{M} \times \mathcal{C}$, tel que
$\Pr_{k \leftarrow \Pi.\texttt{KeyGen}}\left[\Pi.\texttt{Enc}_k\left(m_0\right) = c\right] \neq 0$.
Et comme le système est parfaitement sécurisé, on en déduit :
$\forall m \in \mathcal{M} : \Pr_{k \leftarrow \Pi.\texttt{KeyGen}}\left[\Pi.\texttt{Enc}_k\left(m\right) = c\right] \neq 0$.

Ce qui implique $\forall m \in \mathcal{M} : \Pr_{k \leftarrow \Pi.\mathtt{KeyGen}}[\Pi.\mathtt{Dec}_k(c) = m] = \Pr_{k \leftarrow \Pi.\mathtt{KeyGen}}[\Pi.\mathtt{Enc}_k(m_0) = c] \neq 0$.
Donc on en déduit $\mathcal{M} \subset \{\exists k \in \mathcal{K} : c \in \{\Pi.\mathtt{Enc}_k(m)\}\}_{k \in \mathcal{K}}$.
Donc par correction $\mathcal{M} \subset \{\exists k \in \mathcal{K} : c \in \{\Pi.\mathtt{Dec}_k(c)\}\}_{k \in \mathcal{K}} = \Pi.\mathtt{Dec}(\mathcal{K}, c)$. Comme $\Pi.\mathtt{Dec}$ est déterministe, on peut borner le cardinal de $\Pi.\mathtt{Dec}(\mathcal{K}, c)$, par $|\mathcal{K}|$. On en déduit donc ce qu'il fallait démontrer : $|\mathcal{M}| \leq |\mathcal{K}|$. □

Le théorème 2 nous contraint donc à abandonner le critère de la sécurité parfaite pour définir la confidentialité.

Un autre de problème de taille On peut réécrire la sécurité parfaite sous une forme de jeu de sécurité avec l'exercice suivant.

Exercice 7. *Montrer que la définition 29 est équivalente à dire que pour tout couples de machines de Turing $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*

$$\Pr\left[\mathrm{PrivK}^{\mathrm{eav0}}_{\mathcal{A},\Pi}() = 1\right] = \frac{1}{2}$$

où $\mathrm{PrivK}^{\mathrm{eav0}}_{\mathcal{A},\Pi}$ est le jeu de l'indiscrétion (eavsdropper en anglais) non paramétré (figure 2.2).

Avant d'aller plus loin on remarque que si on s'intéresse au cas où l'ensemble des messages est infini, on se rend compte des limitations intrinsèques de la confidentialité des systèmes de chiffrement. Pour comprendre l'intuition, il suffit de remarquer que si je veux savoir si deux personnes se sont transmis un film en 3D très lourd, ou un fichier de texte d'un haïku, je peux savoir dans quel cas on est en regardant le poids des données transmises, et ce même si les données sont chiffrées, car on s'attend à ce qu'un système de chiffrement conserve la taille des données. L'exercice 11 à la fin de ce chapitre montre en quoi cette limitation est structurelle dans le cas où la taille de l'ensemble des messages est infinie[3].

2.1.2 Sécurité calculatoire : Sécurité contre l'indiscrétion

La sécurité parfaite nous apparaît inaccessible suite au théorème 2, mais heureusement, comme dans le chapitre 1, on peut partir du principe qu'on cherche avant tout à se protéger vis à vis d'un adversaire calculatoirement borné en tolérant qu'il réussisse avec un succès très faible à "casser le système". On va donc réutiliser les notions de PPT , et de fonctions négligeables vues dans le précédent chapitre.

Et donc il est nécessaire de redéfinir les schémas de chiffrement en introduisant le paramètre de sécurité :

Définition 30. *Un schéma de chiffrement est un triplet d'algorithmes PPT :* $(\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec})$.

$$\mathtt{KeyGen} : \left\{\{0,1\}^{\lambda}\right\}_{\lambda \in \mathbb{N}} \to \mathcal{K}$$
$$\mathtt{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$$
$$\mathtt{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M}.$$

On se rend compte qu'on peut alors transformer le jeu $\mathrm{PrivK}^{\mathrm{eav0}}$ en le paramétrant ce qui donne la définition suivante :

Définition 31. *Soit Π, un schéma de chiffrement. Pour tout PPT $\mathcal{A}$, on appelle l'avantage de $\mathcal{A}$, la valeur $Adv^{\mathrm{eav}}_{\mathcal{A},\Pi}(\lambda) := \left|\frac{1}{2} - \Pr[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(\lambda) = 1]\right|$. On dit que le schéma Π est* eav*-sécurisé si et seulement si pour tout PPT $\mathcal{A}$, $Adv^{\mathrm{eav}}_{\mathcal{A},\Pi}(\lambda)$ est négligeable, où $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}$ est le jeu de l'indiscrétion (figure 2.2).*

On remarque que comme l'adversaire est en deux étapes, et que son temps de calcul est borné, l'adversaire peut fabriquer une mémoire (représentée par la chaîne de bits η) lors de la première étape de calcul (c'est à dire l'étape qui correspond au choix du contexte), puis ensuite utiliser cette mémoire dans la deuxième étape de calcul (ce qui explique que η est prise en entrée par $\mathcal{A}_2$).

On remarque également que ce jeu ne cherche à garantir la confidentialité à taille des données connues.

Dans la suite on pourra parfois utiliser une autre définition :

3. Et même dans le cas où l'ensemble des messages est fini, quand on souhaite un système de chiffrement efficace avec $|\mathtt{Enc}(m)| \approx |m|$, l'accès à une donnée chiffrée divulgue des informations sur la taille de la donnée en clair.

$\underline{\text{PrivK}^{\text{eav0}}_{\mathcal{A},\Pi}()}$:	$\underline{\text{PrivK}^{\text{eav}}_{\mathcal{A},\Pi}(\lambda)}$:	$\underline{\text{PrivK}^{\text{eav2}}_{\mathcal{A},\Pi}(\lambda, b)}$:
$k \leftarrow \Pi.\texttt{KeyGen}()$	$k \leftarrow \Pi.\texttt{KeyGen}\left(1^\lambda\right)$	$k \leftarrow \Pi.\texttt{KeyGen}\left(1^\lambda\right)$
$(m_0, m_1) \leftarrow \mathcal{A}_1()$	$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1\left(1^\lambda\right)$	$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1(1^\lambda)$
$b \xleftarrow{\$} \{0,1\}$	$b \xleftarrow{\$} \{0,1\}$	$b' \leftarrow \mathcal{A}_2\left(\Pi.\texttt{Enc}_k\left(m_b\right), \eta\right)$
$b' \leftarrow \mathcal{A}_2\left(\Pi.\texttt{Enc}_k(m_b)\right)$	$b' \leftarrow \mathcal{A}_2\left(\Pi.\texttt{Enc}_k(m_b), \eta\right)$	Si $\lvert m_0\rvert \neq \lvert m_1\rvert$: Retourner $\bot$
Retourner $b \stackrel{?}{=} b'$	Si $\lvert m_0\rvert \neq \lvert m_1\rvert$: Retourner b	Sinon Retourner b'
	Sinon Retourner $b \stackrel{?}{=} b'$	

FIGURE 2.2 – Jeux de sécurité de l'indiscrétion : une version non paramétrée, puis deux versions paramétrées

Définition 32. *Soit* Π*, un schéma de chiffrement. On dit que le schéma* Π *est* eav*-sécurisé si et seulement si pour tout PPT* $\mathcal{A}$ *:*

$$\left|\Pr[\text{PrivK}^{\text{eav2}}_{\mathcal{A},\Pi}(\lambda, 0) = 1] - \Pr[\text{PrivK}^{\text{eav2}}_{\mathcal{A},\Pi}(\lambda, 1) = 1]\right|$$

est négligeable, où $\text{PrivK}^{\text{eav2}}_{\mathcal{A},\Pi}$ *est la deuxième version du jeu de l'indiscrétion (figure 2.2).*

Exercice 8. *Montrer que les définitions 31, et 32 sont équivalentes.*

Preuve laissée en exercice au lecteur. □

Tout d'abord, on peut remarquer avec la propriété suivante que cette définition est plus faible que la sécurité parfaite :

Propriété 3. *Un schéma de chiffrement parfaitement sécurisé est eav-sécurisé*[4]*.*

Preuve laissée en exercice pour le lecteur. □

Corollaire 1. *Le masque jetable est eav-sécurisé.*

A présent, cherchons à construire un schéma qui vérifie cette propriété tout en ayant des tailles de clefs strictement plus petites que les tailles de messages. Dans l'idée on va vouloir reprendre le masque jetable, mais avec un moyen de "compresser" la clef. A priori ce n'est pas possible de compresser sans perte une clef ayant autant d'entropie que sa taille. Si on raisonne dans l'autre sens, il s'agirait de simuler une clef de $\lambda + 1$ bits d'entropie qui en réalité ne contient que λ bits d'entropie. Ceci nous fait penser aux générateurs pseudo-aléatoires largement utilisés depuis longtemps dans les simulations de phénomènes physiques, ces algorithmes sont censés prendre en entrée des chaînes d'"aléa réel" pour produire efficacement une séquence importante de bits qui "ressembleront" à des bits complètement aléatoires qui seraient beaucoup plus coûteux et compliqués à produire.

On pourrait penser à reprendre tel quels ces générateur pseudo-aléatoires, mais la notion de "ressemblance à de l'aléa", n'a pas été pensée dans un contexte de sécurité[5] ;

On va donc devoir recourir à une définition propre à un contexte de sécurité :

Définition 33 (Générateur pseudo aléatoire)**.** *Soit* $\ell : \mathbb{N} \to \mathbb{N}$*, une fonction polynomiale. Soit* G *une fonction DPT telle que pour* $\lambda \in \mathbb{N}$*, pour tout* $s \in \{0,1\}^\lambda : G(s) \in \{0,1\}^{\ell(\lambda)}$*. On dit que* G *est pseudo-aléatoire si et seulement si elle vérifie les deux propriétés suivantes :*

Expansion : $\forall \lambda \in \mathbb{N} : |\ell(\lambda)| > |\lambda|$.

Pseudo-aléa : Pour tout PPT D *:*

$$\left| \Pr_{s \xleftarrow{\$} \{0,1\}^\lambda}\left[D\left(G(s)\right) = 1\right] - \Pr_{x \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}}\left[D(x) = 1\right] \right|$$

est négligeable en λ*.*

4. On paramétrera canoniquement le schéma en redéfinissant $\forall \lambda \in \mathbb{N} : \texttt{KeyGen}\left(1^\lambda\right) := \texttt{KeyGen}()$.
5. En général un GPA est considéré comme bon si il passe un certain nombre de tests statistiques "basiques".

$\underline{\mathcal{D}_{\mathcal{A}}(x)}$:
On suppose λ calculable à partir de $|x|$ [a]
$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1\left(1^\lambda\right)$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}_2\left(x \oplus m_b, \eta\right)$
Si $|m_0| \neq |m_1|$ Retourner b
Sinon Retourner $b \stackrel{?}{=} b'$

a. A partir du moment où ℓ est un polynôme connu, on peut retrouver la préimage efficacement par dichotomie, ou même moins efficacement en évaluant tous les $\ell(\lambda)$ (ça restera une opération polynomiale en λ).

$\underline{\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi_1}(\lambda)}$:
$k \leftarrow \{0,1\}^\lambda$
$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1\left(1^\lambda\right)$
$b \xleftarrow{\$} \{0,1\}$
$b' \leftarrow \mathcal{A}_2\left(G(k) \oplus m_b, \eta\right)$
Si $|m_0| \neq |m_1|$: Retourner b
Sinon Retourner $b \stackrel{?}{=} b'$

$\underline{\mathcal{D}^{\mathrm{random}}_{\mathcal{A}}\left(1^\lambda\right)}$:
$b \xleftarrow{\$} \{0,1\}$
$x \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}$
$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1\left(1^\lambda\right)$
$b' \leftarrow \mathcal{A}_2\left(x \oplus m_b, \eta\right)$
Si $|m_0| \neq |m_1|$: Retourner b
Sinon Retourner $b \stackrel{?}{=} b'$

FIGURE 2.3 – Attaquant $\mathcal{D}$ contre le GPA G, jeu de l'indiscrétion pour Π_1, et $\mathcal{D}^{\mathrm{random}}$ un algorithme qui simule $\mathcal{D}$ qui prend du "vrai aléa" en entrée.

Dans la suite de ce cours lorsqu'on fera référence à un GPA (Générateur pseudo aléatoire), on fera référence à cette définition et non à la définition "hors cryptographie". Grâce à cette fonctionnalité, on va donc, en utilisant seulement x une chaîne de λ bits d'entropie, pouvoir générer une clef de masque jetable $G(x)$ de $\ell(\lambda)$ bits tout en conservant la sécurité de l'indiscrétion avec le schéma Π_1 :

Théorème 3. *Si G est un GPA, alors Π_1 (figure 2.1) est eav-sécurisé.*

On définit le distingueur $\mathcal{D}$ dans la figure 2.3. On remarque d'abord que si l'entrée x est égal à $G(k)$ avec k tiré uniformément dans $\{0,1\}^\lambda$. Alors la distribution $\mathcal{D}(x)$ est identique à celle $\mathcal{D}^{\mathrm{random}}_{\mathcal{A}}\left(1^\lambda\right)$.

Quand on étudie en détail cet algorithme, on se rend compte que b est indépendant de x, donc de $x \oplus m_b$ (car x suit la distribution uniforme), donc de b', ainsi $b \stackrel{?}{=} b'$ est une variable aléatoire de Bernoulli de paramètre $\frac{1}{2}$.

On en déduit donc : $\Pr_{x \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}}\left[\mathcal{D}_{\mathcal{A}}(x) = 0\right] = \frac{1}{2}$.

D'un autre côté on remarque que la distribution de $\mathcal{D}_{\mathcal{A}}(G(s))$, avec s tiré uniformément dans $\{0,1\}^\lambda$, est identique à celle de $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi_1}(\lambda)$.

Donc on en déduit que $\left|\Pr[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi_1}(\lambda) = 0] - \frac{1}{2}\right|$ est égal à

$$\left| \Pr_{s \xleftarrow{\$} \{0,1\}^\lambda}\left[\mathcal{D}_{\mathcal{A}}(G(s)) = 0\right] - \Pr_{x \xleftarrow{\$} \{0,1\}^{\ell(\lambda)}}\left[\mathcal{D}_{\mathcal{A}}(x) = 0\right] \right|. \tag{2.3}$$

On remarque que comme $\mathcal{A}_1$, et $\mathcal{A}_2$ sont des algorithmes PPT par hypothèse, alors $\mathcal{D}$ est aussi un algorithme PPT. Or si G est un GPA, (2.3) est négligeable en λ. On en déduit donc

$$\left|\Pr[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi_1}(\lambda) = 0] - \frac{1}{2}\right| = neg(\lambda).$$

Ce qu'il fallait démontrer. □

Les hypothèses cryptographiques : un des fondements de la cryptologie moderne Nous avons construit le premier schéma de chiffrement "non-trivial" en terme d'efficacité. Pour passer ce cap d'efficacité, la sécurité de ce schéma doit non seulement reposer sur une preuve mathématique, mais aussi sur *une hypothèse cryptographique*, qui sera bien souvent réduit à l'état de conjecture : "G est un GPA". Ainsi on ne garantit plus stricto sensu que la sécurité de schéma restera inviolée, mais que si elle devait être brisée alors la conjecture concernant G serait cassée. Et si cela devait arriver cela ne rendrait pas notre schéma inopérant pour autant, il "suffirait" alors de l'utiliser avec un nouveau G qui vérifierait bien les propriétés des GPA.

Comment choisir un bon candidat pour une hypothèse cryptographique ? Pour choisir un bon candidat, on doit sortir des mathématiques pures, et renouer avec le principe de Kerckhoffs [6]. Les potentiels candidats (par exemple pour un GPA) doivent donc être présentés publiquement et le plus largement diffusés de manière à ce que le plus possible de gens cherchent à "casser" la conjecture "le candidat est un GPA", par exemple en trouvant un algorithme PPT D tel que la grandeur $\left|\Pr_{x\xleftarrow{\$}\{0,1\}^\lambda}[D(G(x))=1]-\Pr_{x\xleftarrow{\$}\{0,1\}^{\ell(\lambda)}}[D(x)=1]\right|$ soit minorée par une fraction rationnelle en λ. Ce paradigme a eu l'effet concret d'intégrer très fortement la cryptologie (aussi bien la cryptographie que la cryptanalyse) à la sphère académique, alors qu'elle était auparavant la chasse gardée des militaires.

2.1.3 Sécurité calculatoire : Sécurité contre les attaques à clairs choisis

Dans cette partie, nous allons chercher à renforcer notre modèle de sécurité (avec pour ambition de ne pas retomber dans la rigidité de la sécurité parfaite). La faiblesse principale des deux précédents modèles (aussi bien la sécurité parfaite que la modèle de l'indiscrétion) est dans la non réutilisation de la clef : Si on chiffre ne serait-ce qu'un deuxième message avec la même clef ces propriétés ne garantissent absolument rien sur la confidentialité des messages (y-compris du premier message chiffré).

On peut chercher à construire un modèle de sécurité dans l'exercice suivant :

Exercice 9.
1. *Construire un jeu de sécurité (avec la définition associée) qui permet de capturer un scénario ou l'adversaire peut chiffrer deux messages.*
2. *A présent, on souhaite renforcer le modèle de la question précédente en laissant à l'adversaire la possibilité de choisir le second message en fonction du chiffré du premier.*
3. *Généraliser les jeux des exercices précédents au contexte où on a n messages au lieu de 2 (avec n un paramètre du jeu traité de manière analogue au paramètre de sécurité).*

Le dernier modèle de l'exercice précédent est intéressant, mais il a l'inconvénient d'avoir un paramètre (ce qu'on veut éviter au maximum), et d'être relativement difficile à lire. On va donc utiliser le paradigme de "renforcement par simplification" en autorisant l'adversaire à demander autant de messages qu'il le souhaite. Pour modéliser cette capacité, on va revenir à la notion d'oracles vue dans le chapitre 1. Ce qui revient donc au jeu de l'attaque à texte clairs choisis, $\mathrm{PrivK}^{\mathrm{cpa}}$ de la figure 2.4 (CPA est l'acronyme de *chosen plaintext attack*), et à la définition associée :

Définition 34. *Soit Π, un schéma de chiffrement. On dit que le schéma Π est sécurisé contre les attaques à clairs choisis si et seulement si pour tout PPT $\mathcal{A}$, la valeur*

$$Adv^{\mathrm{cpa}}_{\mathcal{A},\Pi}(\lambda) := \left|\Pr[\mathrm{PrivK}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(\lambda)=1]-\frac{1}{2}\right|$$

est négligeable en λ.

Dans un exercice ultérieur on montrera pourquoi ce modèle est strictement plus puissant que ceux vus à l'exercice 9.

Construction du schéma On remarque que contrairement au modèle de sécurité de la partie précédente, étant donnée que la clef a vocation à être réutilisée plusieurs fois, on ne va plus chercher à que cette dernière soit plus petite que le message envoyé.

Pour vérifier une telle propriété de sécurité, il faudrait pouvoir générer un masque de Vernam pour chaque message que l'adversaire demanderait à l'oracle. Il apparaît clairement que les GPA ne vont pas suffire pour pouvoir générer autant de bits pseudo-aléatoire. On va donc recourir à une primitive fondamentale plus forte, les fonctions pseudo-aléatoires qui avec une unique clef peuvent former du pseudo-aléa à volonté (tant que cette volonté reste polynomialement bornée).

6. le principe de Kerckhoffs préconise que toutes les méthodes utilisées concernant la confidentialité d'un système de communication n'ont pas à être supposées secrètes, et que seule la clef (une donnée facile à générer, et à communiquer) doit être secrète.

$\underline{\text{PrivK}^{\text{cpa}}_{\mathcal{A},\Pi}(\lambda)}$:	$\underline{\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi}(\lambda)}$:	$\underline{ODec(c)}$:
$k \leftarrow \Pi.\texttt{KeyGen}(1^\lambda)$	$k \leftarrow \Pi.\texttt{KeyGen}(1^\lambda)$	Si $c = c^*$ Retourner $\perp$
$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1^{\Pi.\texttt{Enc}_k(\cdot)}(1^\lambda)$	$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1^{\Pi.\texttt{Enc}_k(\cdot), \Pi.\texttt{Dec}_k(\cdot)}(1^\lambda)$	Sinon
$b \xleftarrow{\$} \{0,1\}$	$b \xleftarrow{\$} \{0,1\}$	Retourner $\Pi.\texttt{Dec}(k, c)$
$b' \leftarrow \mathcal{A}_2^{\Pi.\texttt{Enc}_k(\cdot)}(\Pi.\texttt{Enc}_k(m_b))$	$c^* \leftarrow \Pi.\texttt{Enc}_k(m_b)$	
Si $\lvert m_0\rvert \neq \lvert m_1\rvert$ Retourner b	$b' \leftarrow \mathcal{A}_2^{\Pi.\texttt{Enc}_k(\cdot), ODec(\cdot)}(c^*)$	
Sinon Retourner $b \stackrel{?}{=} b'$	Si $\lvert m_0\rvert \neq \lvert m_1\rvert$ Retourner b	
	Sinon Retourner $b \stackrel{?}{=} b'$	

FIGURE 2.4 – Jeu de sécurité avec adversaire ayant accès aux oracles de chiffrement, puis accès aux oracles de chiffrement et de déchiffrement

$\underline{\mathcal{D}^O_{\mathcal{A}}(1^\lambda)}$:	$\underline{OEnc(x)}$:	$\underline{\text{PrivK}^{\text{cpa}}_{\mathcal{A},\Pi_2}(\lambda)}$:
$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1^{OEnc(\cdot)}(1^\lambda)$	Sinon $r \xleftarrow{\$} \{0,1\}^\lambda$	$k \leftarrow \{0,1\}^\lambda$
$b \xleftarrow{\$} \{0,1\}$	Retourner $(r, O(r) \oplus x)$	$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1^{\Pi.\texttt{Enc}_k(\cdot)}(1^\lambda)$
$r \xleftarrow{\$} \{0,1\}^\lambda$		$r \xleftarrow{\$} \{0,1\}^\lambda$
$b' \leftarrow \mathcal{A}_2^{OEnc(\cdot)}((r, O(r) \oplus m_b), \eta)$		$b \xleftarrow{\$} \{0,1\}$
Si $\lvert m_0\rvert \neq \lvert m_1\rvert$ Retourner b		$b' \leftarrow \mathcal{A}_2^{\texttt{Enc}_k(\cdot)}((r, F_k(r) \oplus m_b), \eta)$
Sinon Retourner $b \stackrel{?}{=} b'$		Si $\lvert m_0\rvert \neq \lvert m_1\rvert$: Retourner b
		Sinon Retourner $b \stackrel{?}{=} b'$

FIGURE 2.5 – Attaquant $\mathcal{D}$ contre la FPA F, et jeu de l'indiscrétion pour Π_2.

Définition 35. *[Fonction pseudo-aléatoire] Soit $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, déterministe, polynomialement bornée et vérifie $\forall k : F(k, \cdot)$ est une fonction qui préserve la taille de l'entrée. On dit que F est pseudo-aléatoire si pour tout adversaire PPT $\mathcal{A}$:*

$$\left| \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} \left(\mathcal{A}^{F_k(\cdot)}(1^\lambda) = 1 \right) - \Pr\left(\mathcal{A}^{f(\cdot)}(1^\lambda) = 1 \right) \right| \leq neg(\lambda)$$

où F_k est la restriction de F à $\{k\} \times \{0,1\}^\lambda$ et $f : \{0,1\}^\lambda \to \{0,1\}^\lambda$ est complètement aléatoire. On peut également réécrire la contrainte avec F_k et f permutations, et à ce moment là on rajoute les oracles F_k^{-1}, f^{-1} (et on parle alors de permutation pseudo-aléatoire (PPA)).

Grâce à une telle primitive fondamentale on va pouvoir générer un nouveau "masque jetable" $F_k(r)$ pour chaque chiffrement de message avec r une chaîne aléatoire qui sera accolée au chiffré de Vernam :

Théorème 4. *Si F est une FPA, Π_2 (figure 2.1) est CPA sécurisé.*

On définit le distingueur $\mathcal{D}$ dans la figure 2.5. On remarque d'abord que la distribution de $\mathcal{D}^{F_k}_{\mathcal{A}}(1^\lambda)$ avec $k \xleftarrow{\$} \{0,1\}^\lambda$ est identique à celle de $\text{PrivK}^{\text{cpa}}_{\mathcal{A},\Pi_2}(\lambda)$ (cf figure 2.5).

Regardons à présent la distribution de $\mathcal{D}^f_{\mathcal{A}}(1^\lambda)$ avec $f \xleftarrow{\$} \left(\{0,1\}^\lambda\right)^{\{0,1\}^\lambda}$ En remarquant que le choix de f revient à choisir $f(y) \in \{0,1\}^\lambda$ pour tout $y \in \{0,1\}^\lambda$. En choisissant ces $f(y)$ "à la volée", on ne change pas la distribution. Ainsi cette distribution est identique à celle de $\mathcal{D}^{(2)}_{\mathcal{A}}$, et donc à celle de $\mathcal{D}^{(3)}$ (où pour cet algorithme, on a seulement décomposé l'appel à *OAlea* pour l'entrée de $\mathcal{A}_2$).

L'algorithme $\mathcal{D}^{(4)}(1^\lambda)$ a une distribution identique à celle de $\mathcal{D}^{(3)}(1^\lambda)$ sauf si l'évènement $E := (r, \cdot) \in Q$ arrive.

Regardons à présent la distribution de $\mathcal{D}^{(4)}(1^\lambda)$. On remarque que x est indépendant de b, et donc comme x suit la distribution uniforme $x \oplus m_b$ est indépendant de b, donc b' est indépendant de b. Ainsi aussi bien b que $b \stackrel{?}{=} b'$ suivent toutes les deux des lois de Bernoulli de paramètre $\frac{1}{2}$.

$\underline{\mathcal{D}_{\mathcal{A}}^{(2)}\left(1^{\lambda}\right)}$:	$\underline{\mathcal{D}_{\mathcal{A}}^{(3)}\left(1^{\lambda}\right)}$:	$\underline{OAlea(m)}$:
$Q := \emptyset$	$Q := \emptyset$	$x \xleftarrow{\$} \{0,1\}^{\lambda}$
$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1^{OAlea(\cdot)}\left(1^{\lambda}\right)$	$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1^{OAlea(\cdot)}\left(1^{\lambda}\right)$	Si $\exists (x,y) \in Q$
$b \xleftarrow{\$} \{0,1\}$	$b \xleftarrow{\$} \{0,1\}$	Retourner $(x, y \oplus m)$
$b' \leftarrow \mathcal{A}_2\left(OAlea(m_b), \eta\right)$	$r, x \xleftarrow{\$} \{0,1\}^{\lambda}$	Sinon $y \xleftarrow{\$} \{0,1\}^{\lambda}$
Si $\lvert m_0\rvert \neq \lvert m_1\rvert$ Retourner b	$Q := Q \cup \{(r,x)\}$	$Q := Q \cup \{(x,y)\}$
Sinon Retourner $b \stackrel{?}{=} b'$	$b' \leftarrow \mathcal{A}_2^{OAlea(\cdot)}\left((r, x \oplus m_b), \eta\right)$	Retourner $(x, y \oplus m)$
	Si $\lvert m_0\rvert \neq \lvert m_1\rvert$	
	Retourner b	
	Sinon Retourner $b \stackrel{?}{=} b'$	

FIGURE 2.6 – Algorithmes $\mathcal{D}^{(2)}$, et $\mathcal{D}^{(3)}$.

$\underline{\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right)}$:	$\underline{OAlea(m)}$:
$Q := \emptyset$	$x \xleftarrow{\$} \{0,1\}^{\lambda}$
$(m_0, m_1, \eta) \leftarrow \mathcal{A}_1^{OAlea(\cdot)}\left(1^{\lambda}\right)$	Si $\exists (x,y) \in Q$ retourner $(x, y \oplus m)$
$b \xleftarrow{\$} \{0,1\}$	Sinon $y \xleftarrow{\$} \{0,1\}^{\lambda}$
$r, x \xleftarrow{\$} \{0,1\}^{\lambda}$	$Q := Q \cup \{(x,y)\}$
$b' \leftarrow \mathcal{A}_2^{OAlea(\cdot)}\left((r, x \oplus m_b), \eta\right)$	Retourner $(x, y \oplus m)$
Si $\lvert m_0\rvert \neq \lvert m_1\rvert \wedge (r, \dot{}) \in Q$	
Retourner b	
Sinon Retourner $b \stackrel{?}{=} b'$	

FIGURE 2.7 – Algorithme $\mathcal{D}^{(4)}$.

Donc on en déduit que

$$\begin{aligned} &\left|\Pr[\mathrm{PrivK}_{\mathcal{A},\Pi_1}^{\mathrm{cpa}}(\lambda) = 0] - \frac{1}{2}\right| \\ =&\left|\Pr_{k \xleftarrow{\$} \{0,1\}^{\lambda}}\left[\mathcal{D}_{\mathcal{A}}^{F_k(\cdot)}\left(1^{\lambda}\right) = 0\right] - \Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0\right]\right| \end{aligned} \tag{2.4}$$

Regardons le deuxième terme dans le détail, en posant $E :=$ le x tiré au moment du calcul du cryptogramme challenge a déjà été tiré lors des appels à $OAlea$:

$$\begin{aligned} &\Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0\right] \\ =&\Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0 | E\right]\Pr[E] - \Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0 | \bar{E}\right]\Pr[\bar{E}] \\ =&\Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0 | E\right]\Pr[E] + \Pr\left[\mathcal{D}_{\mathcal{A}}^{(3)}\left(1^{\lambda}\right) = 0 | \bar{E}\right](1 - \Pr[E]) \\ =&\Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0 | E\right]\Pr[E] + \Pr\left[\mathcal{D}_{\mathcal{A}}^{(3)}\left(1^{\lambda}\right) = 0\right] - \Pr\left[\mathcal{D}_{\mathcal{A}}^{(3)}\left(1^{\lambda}\right) = 0 | E\right]\Pr[E] \\ =&\Pr\left[\mathcal{D}_{\mathcal{A}}^{(3)}\left(1^{\lambda}\right) = 0\right] + \Pr[E]\left(\Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0 | E\right] - \Pr\left[\mathcal{D}_{\mathcal{A}}^{(3)}\left(1^{\lambda}\right) = 0 | E\right]\right) \end{aligned} \tag{2.5}$$

On remarque que $\Pr[E] \leq \frac{q}{2^{\lambda}}$, où q représente le nombre de requêtes total de $\mathcal{A}$. On déduit donc en remarquant que $\Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0 | E\right]$ et $\Pr\left[\mathcal{D}_{\mathcal{A}}^{(3)}\left(1^{\lambda}\right) = 0 | E\right]$ sont bornés par 1 à partir de (2.5) :

$$\left|\Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^{\lambda}\right) = 0\right] - \Pr\left[\mathcal{D}_{\mathcal{A}}^{(3)}\left(1^{\lambda}\right) = 0\right]\right| \leq \frac{q}{2^{\lambda}}. \tag{2.6}$$

Donc en combinant le fait que $\mathcal{D}^{(3)}$ et $\mathcal{D}^{f}(\cdot)$ (avec f tirée uniformément aléatoire parmi les fonctions

de $\{0,1\}^\lambda \to \{0,1\}^\lambda$) suivent la même distribution et (2.6) on déduit que :

$$\left| \Pr_{f \xleftarrow{\$} (\{0,1\}^\lambda)^{\{0,1\}^\lambda}} \left[\mathcal{D}_{\mathcal{A}}^{f(\cdot)}\left(1^\lambda\right) = 0\right] - \Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^\lambda\right) = 0\right] \right| \leq \frac{q}{2^\lambda}. \tag{2.7}$$

Et par inégalité triangulaire on a aussi :

$$\begin{aligned} &\left| \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} \left[\mathcal{D}_{\mathcal{A}}^{F_k(\cdot)}\left(1^\lambda\right) = 0\right] - \Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^\lambda\right) = 0\right] \right| \\ \leq &\left| \Pr_{f \xleftarrow{\$} (\{0,1\}^\lambda)^{\{0,1\}^\lambda}} \left[\mathcal{D}_{\mathcal{A}}^{f(\cdot)}\left(1^\lambda\right) = 0\right] - \Pr\left[\mathcal{D}_{\mathcal{A}}^{(4)}\left(1^\lambda\right) = 0\right] \right| \\ &+ \left| \Pr_{f \xleftarrow{\$} (\{0,1\}^\lambda)^{\{0,1\}^\lambda}} \left[\mathcal{D}_{\mathcal{A}}^{f(\cdot)}\left(1^\lambda\right) = 0\right] - \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} \left[\mathcal{D}_{\mathcal{A}}^{F_k(\cdot)}\left(1^\lambda\right) = 0\right] \right|. \end{aligned} \tag{2.8}$$

En appliquant (2.4), et (2.7) à (2.8), on en déduit :

$$\begin{aligned} &\left| \Pr[\mathrm{PrivK}_{\mathcal{A},\Pi_1}^{\mathrm{cpa}}(\lambda) = 0] - \frac{1}{2} \right| \\ \leq &\frac{q}{2^\lambda} + \left| \Pr_{f \xleftarrow{\$} (\{0,1\}^\lambda)^{\{0,1\}^\lambda}} \left[\mathcal{D}_{\mathcal{A}}^{f(\cdot)}\left(1^\lambda\right) = 0\right] - \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} \left[\mathcal{D}_{\mathcal{A}}^{F_k(\cdot)}\left(1^\lambda\right) = 0\right] \right|. \end{aligned} \tag{2.9}$$

On remarque que comme $\mathcal{A}_1$, et $\mathcal{A}_2$ sont des algorithmes PPT par hypothèse, alors $\frac{q}{2^\lambda}$ est négligeable en λ et $\mathcal{D}$ est aussi un algorithme PPT. Or si F est une FPA :

$$\left| \Pr_{f \xleftarrow{\$} (\{0,1\}^\lambda)^{\{0,1\}^\lambda}} \left[\mathcal{D}_{\mathcal{A}}^{f(\cdot)}\left(1^\lambda\right) = 0\right] - \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda} \left[\mathcal{D}_{\mathcal{A}}^{F_k(\cdot)}\left(1^\lambda\right) = 0\right] \right| = neg(\lambda).$$

On en déduit donc de (2.9) :

$$\left| \Pr[\mathrm{PrivK}_{\mathcal{A},\Pi_1}^{\mathrm{cpa}}(\lambda) = 0] - \frac{1}{2} \right| = neg(\lambda)$$

Ce qu'il fallait démontrer.

□

2.1.4 Sécurité calculatoire : Sécurité contre les attaques à chiffrés choisis

Assez naturellement, on va chercher à étendre la notion de sécurité en imaginant un adversaire qui non seulement à accès à des oracles de chiffrement, mais aussi de déchiffrement. Mais si on ne fait pas attention, lorsqu'on définit le jeu de sécurité on peut tomber dans *l'écueil du jeu structurellement trop fort* : C'est à dire que si l'adversaire demande à l'oracle de déchiffrement de déchiffrer c^*, il va très facilement se rendre compte (par correction du système de chiffrement) si le message chiffré est m_0 ou m_1. Ainsi on va légèrement contraindre ces appels à l'oracle (comme on avait légèrement modifié la condition de victoire du jeu d'inforgeabilité dans le chapitre 1 pour exclure les messages demandés à l'oracle des attaques réussies (formalisé avec l'ensemble Q)).

Définition 36. *Soit Π, un schéma de chiffrement. On dit que le schéma Π est sécurisé contre les attaques à clairs choisis si et seulement si pour tout PPT $\mathcal{A}$, la valeur*

$$Adv_{\mathcal{A},\Pi}^{\mathrm{cca}}(\lambda) := \left| \Pr[\mathrm{PrivK}_{\mathcal{A},\Pi}^{\mathrm{cca}}(\lambda) = 1] - \frac{1}{2} \right|$$

est négligeable en λ.

Pour le moment, on ne cherchera pas à construire garantir une telle sécurité et on se contentera de la définition. On verra dans le chapitre 3 comment être résilient face à un tel adversaire.

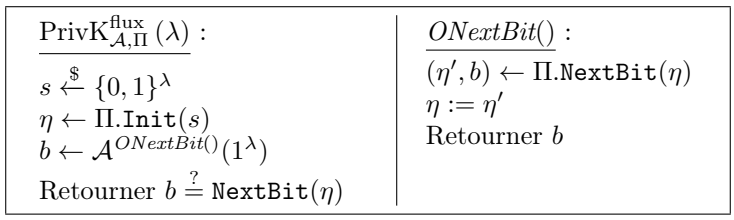

FIGURE 2.8 – Jeu de sécurité du chiffrement de flux

2.2 Systèmes de chiffrement évolués

Dans cette partie, on ne va pas chercher à gagner en sécurité, mais en efficacité. On remarque en effet que le schéma Π_2 produit des chiffrés qui font le double de la taille du message en clair. De plus il nécessite durant le processus de chiffrement de produire autant d'aléa que de quantité d'information réellement échangé. Même si cet aléa n'a pas a être secret (il fait partie du chiffré qui est publiquement accessible), il est tout de même fort coûteux à générer [7].

2.2.1 Chiffrement de flux

Si on se concentre sur ce problème d'aléa, les GPA, et les FPA ont la particularité, si on les utilise de manière basique, de créer à peine plus d'aléa que ce qu'ils ont besoin de prendre en entrée.

On va donc chercher à créer une quantité beaucoup plus importante d'aléa avec la même base (qu'on appellera la graine). Puis on xorera cette chaîne d'aléa avec le message pour le chiffrer en reprenant une fois de plus la stratégie du masque jetable.

Pour modéliser cette stratégie, nous allons créer une nouvelle primitive fondamentale : le chiffrement de flux.

Définition 37. *Un chiffrement de flux est défini par deux fonctions déterministes polynomialement bornées :*

- `Init` *: Prend en entrée une graine (une chaîne de bits de la taille du paramètre de sécurité), et renvoie un état.*
- `NextBit` *: Prend en entrée un état, et renvoie un bit et un nouvel état.*

Définition 38. *On dit qu'un chiffrement de flux est sécurisé si et seulement si pour tout PPT $\mathcal{A}$:*

$$\left|\Pr[\text{PrivK}^{\text{flux}}_{\mathcal{A},\Pi}(\lambda) = 1] - \frac{1}{2}\right| \leq neg(\lambda).$$

Cette primitive fondamentale peut être utilisée de deux manières :

Utilisation synchrone. L'idée de cette utilisation est de considérer l'état du chiffrement du flux comme une clef secrète et de la mettre à jour autant de fois que nécessaire à chaque qu'on chiffre ou qu'on déchiffre une donnée. Le schéma de chiffrement associé est formalisée dans la figure 2.9. On remarque qu'il ne respecte pas le formalisme de la définition 30, puisque la clef secrète ici est dynamique (elle varie à chaque opération de chiffrement ou de déchiffrement).

Exercice 10.

1. *Adapter les définitions 27, 30 à ce nouveau contexte.*
2. *Redéfinir la sécurité contre les attaques à clairs choisis (définition 34) par rapport aux définitions de la question précédente.*
3. *Prouver la sécurité de $\Pi_{flux,sync}$ (figure 2.9) par rapport à la définition précédente.*

Utilisation asynchrone. Pour pouvoir utiliser les chiffrements de flux de manière plus standard (c-à-d avec une clef statique), on va devoir étendre notre définition. On regardera ça plus en détails avec l'exercice 20.

7. Avec le risque si cet aléa n'est pas choisi assez uniformément d'une faille de sécurité (puisque les prémisses de notre preuve ne sera alors plus valide).

$\Pi_{\text{flux,sync}}.\texttt{KeyGen}\left(1^\lambda\right)$:	$\Pi_{\text{flux,sync}}.\texttt{Enc}(sk, m)$:	$\Pi_{\text{flux,sync}}.\texttt{Dec}(sk, c)$:
$s \xleftarrow{\$} \{0,1\}^\lambda$	$\eta_0 := sk$	$\eta_0 := sk$
Retourner $(\texttt{Init}(s))$	Pour $i \in [\![1; \lvert m\rvert]\!]$:	Pour $i \in [\![1; \lvert c\rvert]\!]$:
	$(\eta_i, b_i) := \texttt{NextBit}\,(\eta_{i-1})$	$(\eta_i, b_i) := \texttt{NextBit}\,(\eta_{i-1})$
	Mettre à jour $sk := \eta_m$	Mettre à jour $sk := \eta_m$
	Retourner	Retourner
	$\left(b_1 \oplus m_1, \ldots, b_{\lvert m\rvert} \oplus m_{\lvert m\rvert}\right)$	$\left(b_1 \oplus c_1, \ldots, b_{\lvert c\rvert} \oplus c_{\lvert c\rvert}\right)$

FIGURE 2.9 – Schéma de chiffrement synchrone basé sur un chiffrement de flux

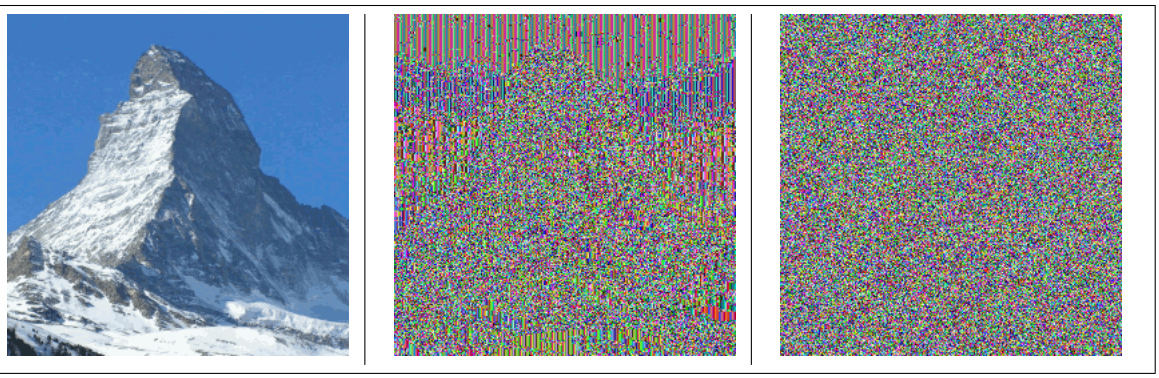

FIGURE 2.10 – La même image en clair, puis chiffrée avec le mode ECB, puis avec le mode CBC (avec DES comme PRP)

2.2.2 Chiffrement par blocs

Dans cette section, on va avoir la même ambition que dans la partie précédente, mais en réutilisant les primitive de la section 2.1.3 : les fonctions pseudo-aléatoires et les permutations pseudo-aléatoires (PPA). Le point commun entre toutes les méthodes de chiffrement par blocs, et de découper les messages en blocs de taille constante (ou dépendant du paramètre de sécurité) puis de générer un chiffré qui est le résultat d'une concaténation de blocs de taille analogue. On pourrait définir formellement ce qu'est un schéma de chiffrement par blocs, mais étant donné que ce n'est pas nécessaire et qu'il y a déjà beaucoup de définitions formelles dans ce cours, on se contentera de décrire formellement les méthodes de chiffrement par blocs les plus connus (et pour ceux qui sont en manque de formalisme, on pourra toujours regarder l'exercice 22). On notera quand dans tous les cas suivants, la génération de la clef consiste à tirer uniformément une clef k de taille λ qui correspond à la clef de la FPA (ou de la PPA) vue dans la définition 35.

Remarque : Ici pour simplifier nos modèles on considère une taille de blocs identique à celle des clefs (en l'occurrence égale au paramètre de sécurité). Dans le cas général, ce n'est pas toujours le cas, par exemple les tailles de blocs pour AES sont identiques pour tous les niveaux de sécurité définis. Cette simplification ne change pas grand chose à nos raisonnements et analyses.

Mode ECB (Electronic Code Book). Ce mode est le plus "naturel". Il consiste tout simplement à appliquer à chaque bloc la PPA F_k (cf la description de l'algorithme de chiffrement sur la figure 2.11 ou sa représentation graphique figure 2.12). Si cette méthode a l'avantage d'être particulièrement simple, et optimisé au bit près en terme de compacité [8], elle a le lourd inconvénient d'être déterministe, et donc de chiffrer toujours de la même manière le même message. Et donc elle ne pourra pas vérifier la sécurité contre des attaques à clairs choisis (cf exercice 16 pour plus de détails). Pire même pour une unique utilisation, un même bloc déjà chiffré sera reconnaissable, ce qui est particulièrement problématique par exemple dans un contexte de chiffrement d'images comme on peut le constater avec la figure 2.10. Plus formellement, on peut montrer que ce mode de chiffrement par bloc ne vérifie pas la sécurité contre l'indiscrétion.

8. On remarque en effet que la taille du chiffré fait exactement la taille du texte clair.

$\underline{\Pi_{\text{ECB}}.\texttt{Enc}_k\,(m_1||\ldots||m_\ell)}$:
Pour $i \in [\![1,\ldots,\ell]\!]$:
$c_i := F_k(m_i)$
Retourner $(c_1||\ldots||c_\ell)$

$\underline{\Pi_{\text{OFB}}.\texttt{Enc}_k\,(m_1||\ldots||m_\ell)}$:
$c_0 \xleftarrow{\$} \{0,1\}^\lambda$
$f_0 := c_0$
Pour $i \in [\![1,\ldots,\ell]\!]$:
$f_i := F_k\,(f_{i-1})$
$c_i := m_i \oplus f_i$
Retourner $(c_0||c_1||\ldots||c_\ell)$

$\underline{\Pi_{\text{CBC}}.\texttt{Enc}_k\,(m_1||\ldots||m_\ell)}$:
$c_0 \xleftarrow{\$} \{0,1\}^\lambda$
Pour $i \in [\![1,\ldots,\ell]\!]$:
$c_i := F_k\,(c_{i-1} \oplus m_i)$
Retourner $(c_0||c_1||\ldots||c_\ell)$

FIGURE 2.11 – Algorithmes de chiffrement par blocs des modes ECB, OFB, et CBC

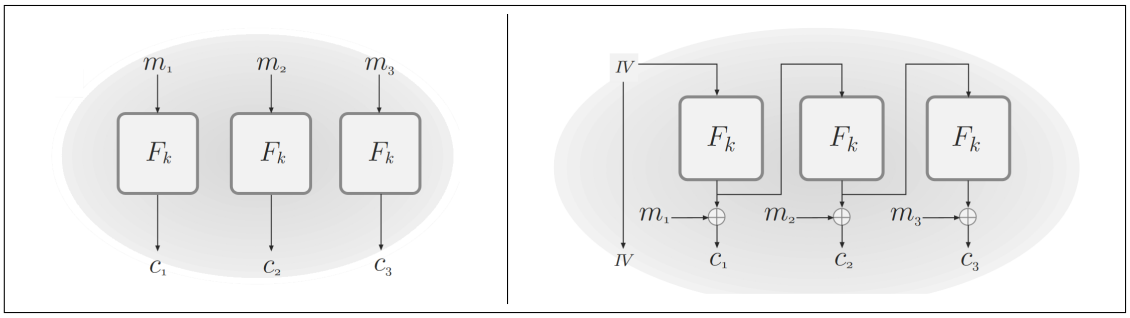

FIGURE 2.12 – Représentation graphique des modes ECB, et OFB

Pour les constructions suivantes, il est acté que l'algorithme de chiffrement ne doit pas être déterministe. L'aléa du chiffrement, qui correspond au r du schéma Π_2 (cf figure 2.1), sera divulgué similairement à la construction Π_2 en constituant le premier bloc du chiffré c_0. Assez souvent, dans la littérature, ce bloc sera noté *IV* pour *Initial Vector* (vecteur d'initialisation dans la langue de Molière). De fait les trois modes présentés ici sont sécurisés contre les attaques à clairs choisis (mais les preuves ne seront pas abordées ici).

Mode OFB (Output FeedBack). Ce mode (cf la description de l'algorithme de chiffrement sur la figure 2.11 ou sa représentation graphique figure 2.12) se rapproche des chiffrement par flux : Il consiste à générer une longue suite de blocs à partir du vecteur d'initialisation c_0 en lui appliquant des itérations de la FPA F_k : $\big(F_k(c_0), F_k^2(c_0), \ldots, F_k^\ell(c_0)\big)$. Puis à xorer cette suite de blocs avec notre message et renvoyer le résultat de ce xor précédé du vecteur d'initialisation.

Pour déchiffrer, il suffit de générer la même suite de blocs à partir du vecteur d'initialisation puis de la xorer avec le reste du chiffré pour retrouver le message en clair.

Il a l'avantage d'être résilient aux erreurs de transmission : En effet, si un bloc du chiffré distinct du vecteur d'initialisation est altéré (par du bruit électromagnétique par exemple) l'essentiel du message en clair est obtenu sans erreur après déchiffrement. Dans certains contextes (par exemple pour une vidéo où une erreur d'un pixel ne sera même pas détectée par le spectateur), c'est une propriété très intéressante.

En revanche il ne permet pas de déchiffrer des morceaux arbitraires d'un même message de manière efficace (on doit toujours recalculer les itérés de F_k à partir du premier bloc).

On remarque également qu'on peut précalculer des "proto-chiffrés" sans connaître le message, ce qui dans certains contextes peut avoir un intérêt [9].

Mode CBC (Cipher Block Chaining). Pour ce mode, on a besoin d'une PPA : On considère le vecteur d'initialisation comme un premier bloc. Puis on xor chaque bloc du message avec le résultat du chiffrement du bloc précédent avant d'en calculer l'image par le PPA F_k (cf figure 2.11). On laisse en exercice au lecteur la rédaction de l'algorithme de déchiffrement.

On remarque que ce mode est strictement moins bien que le précédent en terme de résilience au bruit.

9. On rappelle que le xor est une opération très efficace numériquement (une seule opération assembleur dans la plupart des architectures des ordinateurs modernes), et qu'en revanche l'évaluation de F_k peut être en pratique un peu lourde.

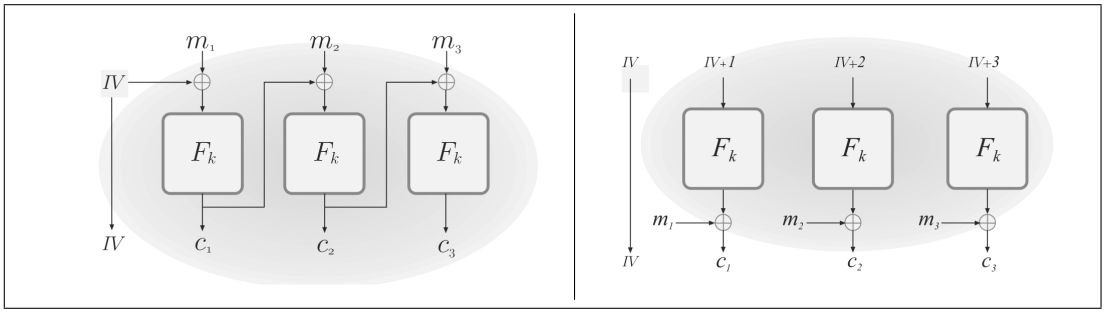

FIGURE 2.13 – Représentation graphique des modes CBC, et CTR

Mode CTR (Compteur). Comme pour le mode OFB, ce mode peut également être vu comme d'inspiration "chiffrement de flux". Mais pour calculer notre flux pseudo-aléatoire, au lieu de calculer des itérés de F_k on calcule simplement l'image par F_k des ℓ entiers consécutifs à partir du vecteur d'initialisation.

En plus d'être précalculable et résilient au bruit, cette méthode à l'avantage d'être parallélisable : on peut calculer simultanément les images par F_k alors que pour le mode OFB, il fallait les calculer les uns après les autres.

On remarque également qu'on peut faire des déchiffrements partiels de messages de manière efficace.

2.3 Exercices

Exercice 11. *On suppose dans cet exercice l'existence d'un système de chiffrement correct* Π *sur l'ensemble des message* $\mathcal{M} = \{0,1\}^*$. *On pose* $\forall N \in \mathbb{N} : \mathcal{M}_N = \cup_{i=0}^{N} \{0,1\}^i$.

1. *Montrer que* $\forall k \in \{\Pi.\texttt{KeyGen}()\}, \forall m_0, m_1 \in \mathcal{M} : \{\Pi.\texttt{Enc}_k(m_0)\} \cap \{\Pi.\texttt{Enc}_k(m_1)\} = \emptyset$.
2. *En déduire que pour* n *assez grand,* $p_n := \Pr_{m \xleftarrow{\$} \mathcal{M}_n}\left[|\Pi.\texttt{Enc}_k(m)| \geq \frac{n}{2}\right] > \frac{3}{4}$.
3. *Montrer qu'il existe un* $d \in \mathbb{N}[X]$, *tel que* $\forall \lambda \in \mathbb{N} : \max_{k \in \texttt{KeyGen}(1^\lambda)} |\{\Pi.\texttt{Enc}_k(0)\} \leq d(\lambda)$.
4. *Déduire des deux questions précédentes que* Π *ne peut pas être sécurisé contre les attaques à clairs choisis.*
5. *Transformer la condition de victoire du jeu de sécurité à clairs choisis pour pouvoir théoriquement couvrir le cas où* $\mathcal{M} = \{0,1\}^*$.

Exercice 12 (Définitions naïves (1))**.** *On constate qu'en pratique, les cryptanalystes essaient de retrouver la clef à partir de couples (messages,cryptogrammes) où les messages sont choisis, et ne considère pas les jeux d'indistingabilité.*

1. *Définir une définition de sécurité qui colle plus à ces attaques.*
2. *Montrer que dans le cas d'un schéma correct, la sécurité contre les attaques à clairs choisis implique la sécurité définit ci-dessus.*
3. *Construire un schéma trivial qui satisfait la sécurité définit dans cette exercice, mais qui n'a aucun intérêt en pratique.*

Exercice 13 (Définitions naïves (2))**.**

1. *On peut commencer par considérer la définition suivante qui nous dit qu'en moyenne il est difficile de retrouver intégralement le message chiffré :*

   Définition 39. *On dit qu'un schéma* Π *est sécurisé naïvement si pour tout adversaire probabiliste polynomialement borné* $\mathcal{A}$ *:*

   $$\Pr[\text{PrivK}^{\text{naïf}}_{\mathcal{A},\Pi}(\lambda) = 1] = neg\,(\lambda)$$

   dans lequel $\text{PrivK}^{\text{naïf}}$ *fait référence au jeu défini dans la figure 2.14.*

   Quelle est la condition implicite sur $|\mathcal{M}|$ *pour que la définition soit opérante ?*

2. *On suppose que* $\mathcal{M}$ *est une constante indépendante du paramètre de sécurité. Définir alors un adversaire qui rend tout système non sécurisé par rapport à la définition 39.*

$\underline{\text{PrivK}^{\text{naïf}}_{\mathcal{A},\Pi}(\lambda)}$:	$\underline{\text{PrivK}^{\text{eavd}}_{\mathcal{A},\Pi}(\lambda)}$:
$k \leftarrow \Pi.\texttt{KeyGen}\left(1^\lambda\right)$	$k \leftarrow \Pi.\texttt{KeyGen}(1^\lambda)$
$m \xleftarrow{\$} \mathcal{M}$	$(d_0, d_1, \eta) \leftarrow \mathcal{A}_1\left(1^\lambda\right)$
$m' \leftarrow \mathcal{A}\left(\Pi.\texttt{Enc}_k(m)\right)$	$b \xleftarrow{\$} \{0,1\}$
Retourner $m \stackrel{?}{=} m'$	$m \xleftarrow{d_b} \mathcal{M}$
	$b' \leftarrow \mathcal{A}_2\left(\Pi.\texttt{Enc}_k\left(m\right), \eta\right)$
	Retourner $b \stackrel{?}{=} b'$

FIGURE 2.14 – Jeu de confidentialité naïf, et jeu d'indistingabilité des distributions

$\underline{\text{PrivK}^{\text{LR-cpa}}_{\mathcal{A},\Pi}(\lambda)}$:	$\underline{LR_{k,b}(m_0, m_1)}$:
$k \leftarrow \Pi.\texttt{KeyGen}\left(1^\lambda\right)$	Si $\lvert m_0\rvert \neq \lvert m_1\rvert$: Retourner $\perp$
$b \xleftarrow{\$} \{0,1\}$	Sinon : Retourner $\Pi.\texttt{Enc}_k(m_b)$
$b' \leftarrow \mathcal{A}^{LR_{k,b}(\cdot,\cdot)}\left(1^\lambda\right)$	
Retourner $b \stackrel{?}{=} b'$	

FIGURE 2.15 – Jeu d'indistingabilité Gauche ou Droite, et le schéma Π_2

3. *On considère alors cette nouvelle définition :*

   Définition 40. *On dit qu'un schéma* Π *est sécurisé naïvement si pour tout adversaire probabiliste polynomialement borné* $\mathcal{A}$ *:*

   $$\left|\Pr[\text{PrivK}^{\text{naïf}}_{\mathcal{A},\Pi}(\lambda) = 1] - \frac{1}{|\mathcal{M}|}\right| = neg(\lambda)$$

   dans lequel $\text{PrivK}^{\text{naïf}}$ *fait référence au jeu défini dans la figure 2.14.*

4. *Construire un schéma* Π*, qui vérifie cette propriété, mais ne vérifie pas la sécurité contre l'indiscrétion.*

5. *On définit cette nouvelle propriété de sécurité :*

   Définition 41. *On dit qu'un schéma* Π *est sécurisé naïvement si pour tout adversaire probabiliste polynomialement borné* $\mathcal{A}$ *:*

   $$\Pr\left[\text{PrivK}^{\text{eavd}}_{\mathcal{A},\Pi}(\lambda) = 1\right] = neg(\lambda)$$

   dans lequel $\text{PrivK}^{\text{eavd}}$ *fait référence au jeu défini dans la figure 2.14.*

   Quel est l'implicite sur (d_0, d_1) *? Est-ce un problème en pratique selon vous ?*

6. *Montrer que cette définition est équivalente à la sécurité contre l'indiscrétion (définition 31).*

Exercice 14 (Paradigme Gauche ou Droite)**.**

Définition 42. *Soit* Π*, un schéma de chiffrement. On dit que le schéma* Π *est sécurisé contre les attaques à clairs choisis si et seulement si pour tout PPT* $\mathcal{A}$*, la valeur*

$$Adv^{\text{cpa}}_{\mathcal{A},\Pi}(\lambda) := \left|\Pr[\text{PrivK}^{\text{LR-cpa}}_{\mathcal{A},\Pi}(\lambda) = 1] - \frac{1}{2}\right|$$

est négligeable en λ*.*

Montrer que cette définition est équivalente à la définition 34.

Exercice 15 (Sécurité Sémantique)**.**

Définition 43. *Soit* $\ell \in \mathbb{N}$*, on pose* $\mathcal{M} = \{0,1\}^\ell$*. Soit* Π*, un sysème de chiffrement sur* $\mathcal{M}$*. Alors si pour tout* $i \in \{1, \ldots, \ell\}$*, pour tout PPT* $\mathcal{A}$ *:*

$$\Pr\left[\mathcal{A}\left(1^\lambda, \Pi.\texttt{Enc}_k(m)\right) = m^i\right] \leq \frac{1}{2} + neg(\lambda),$$

On dit que le schéma est sémantiquement sécurisé.

Montrer que cette définition est équivalente à la sécurité contre l'indiscrétion (définition 31).

Exercice 16 (Chiffrement déterministe)**.** *Soit* Π*, un système de chiffrement correct.*

1. *Montrer que si* $\Pi.\texttt{KeyGen}$ *est déterministe, alors* Π *ne vérifie pas la sécurité contre l'indiscrétion.*
2. *On suppose que* $\forall\lambda \in \mathbb{N}$ *:* $\Pi.\texttt{KeyGen}$ *utilise au plus* N_λ *bits d'aléa. Montrer que si* $N_\lambda = O(\log(\lambda))$*, alors* Π *ne vérifie pas la sécurité contre l'indiscrétion.*
3. *Montrer que si* $\Pi.\texttt{Enc}$ *est déterministe, alors* Π *ne vérifie pas la sécurité contre les attaques à clairs choisis.*
4. *On suppose que* $\forall\lambda \in \mathbb{N}, \forall k \in \Pi.\texttt{KeyGen}\left(1^\lambda\right)$ *:* $\Pi.\texttt{Enc}_k$ *utilise au plus* N_λ *bits d'aléa. Montrer que si* $N_\lambda = O(\log(\lambda))$*, alors* Π *ne vérifie pas la sécurité contre les attaques à clairs choisis.*

Exercice 17 (Générateurs pseudo-aléatoires)**.** *Soit* G, G'*, deux générateurs pseudo-aléatoires respectivement de paramètres d'expansion* ℓ, ℓ'*. Dire dans les cas suivants si* G_i *est nécessairement un GPA, avec une preuve dans le cas positif et un contre-exemple dans le cas négatif.*

1. $\forall s \in \{0,1\}^\lambda : G_1(s_1, \ldots, s_\lambda) := G\left(s_1, \ldots, s_{\lfloor \frac{\lambda}{2} \rceil}\right)$.
2. $\forall s \in \{0,1\}^\lambda : G_2(s) := G(s)||G'(s)$.
3. $\forall s \in \{0,1\}^\lambda : G_3(s) := G\left(s_1, \ldots, s_{\lfloor \frac{\lambda}{2} \rfloor}\right)||G\left(s_{\lceil \frac{\lambda}{2} \rceil}, \ldots, s_\lambda\right)$.
4. $\forall s \in \{0,1\}^\lambda : G_4(s) := G\left(0^\lambda||s\right)$.

Exercice 18 (FPA)**.** *Soit* F, F'*, deux fonctions pseudo-aléatoires. Dire dans les cas suivants si* $F^{(i)}$ *est nécessairement une FPA, avec une preuve dans le cas positif et un contre-exemple dans le cas négatif.*

1. $\forall k, s \in \{0,1\}^\lambda : F_k^{(1)}(s) := F_k(0||s)||F_k(0||s)$.
2. $\forall k, s \in \{0,1\}^\lambda : F_k^{(2)}(s) := F_k(0||s)||F_k(1||s)$.
3. $\forall k, s \in \{0,1\}^\lambda : F_k^{(3)}(s) := F_k(0||s)||F_k(s||1)$.

Exercice 19 (PPA)**.** *Montrer qu'une PPA est une FPA.*

Exercice 20 (Schéma de chiffrement asynchrone basé sur le chiffrement de flux)**.** *Dans cet exercice, on va essayer d'adapter nos définitions de chiffrement de flux afin de pouvoir utiliser des clefs statiques.*

Définition 44. *Un chiffrement de flux asynchrone est défini par deux fonctions déterministes polynomialement bornées :*

— `Init` *: Prend en entrée une graine (une chaîne de bits de la taille du paramètre de sécurité), un vecteur d'initialisation et renvoie un état.*

— `NextBit` *: Prend en entrée un état, et renvoie un bit et un nouvel état.*

1. *Adapter la définition de sécurité 38 à cette nouvelle définition.*
2. *L'idée morale est donc de garder toujours la même graine comme clef secrète, et de varier le IV à chaque chiffrement comme on utilise le r pour le schéma* Π_2*. Formaliser cela en écrivant un schéma de chiffrement compatible avec la définition 30.*
3. *Montrer que ce schéma est sécurisé contre les attaques à clairs choisis.*

Exercice 21 (Lien entre GPA, chiffrement par flux, et FPA)**.**

1. *Construire un GPA de facteur d'expansion arbitraire* ℓ*, à partir d'un chiffrement de flux.*
2. *Construire un chiffrement de Flux à partir d'une FPA.*

Exercice 22 (Formalisme du chiffrement par blocs)**.**

1. *On peut caractériser un chiffrement par blocs deux fonctions* F_1, F_2*. Avec* F_1 *qui prend comme variables* f, c, m*, et* F_2 *qui prend comme variable* c *et* m*. Expliquer comment.*
2. *Donner des conditions nécessaires et suffisantes sur* F_1*,* F_2 *pour savoir si la fonction est parallélisable, précalculable, résiliente aux erreurs, et partiellement déchiffrable.*

Exercice 23 (Mode CFB)**.**

1. *Écrire l'algorithme de déchiffrement, en précisant si on a besoin que* F *soit une FPA ou une PPA.*

$\underline{\Pi_{\text{CFB}}.\texttt{Enc}_k\left(m_1||\ldots||m_\ell\right)}$:
$c_0 \xleftarrow{\$} \{0,1\}^\lambda$
Pour $i \in [\![1,\ldots,\ell]\!]$:
$c_i := F_k\left(c_{i-1}\right) \oplus m_i$
Retourner $(c_0||c_1||\ldots||c_\ell)$

FIGURE 2.16 – Algorithme de chiffrement par blocs du mode CFB

2. *Décrire les intérêts et les inconvénients de ce système (résilience au bruit, déchiffrement partiel, parallélisation du déchiffrement etc).*
3. *Montrer que si F est une FPA, alors Π_{CFB} est sécurisé contre les attaques à clairs choisis.*

Exercice 24 (Attaques à chiffrés choisis)**.**

1. *Prouver que Π_2 n'est pas sécurisé contre les attaques à chiffrés choisis.*
2. *Prouver que les modes de chiffrement par blocs vus dans la partie 2.2.2 ne sont pas sécurisés contre les attaques à chiffrés choisis.*

Exercice 25 (Fonctions pseudo-aléatoires faibles)**.**

Définition 45. *On dit qu'une fonction est faiblement pseudo-aléatoire, si et seulement si :*

$$\left| \Pr_{k\xleftarrow{\$}\{0,1\}^\lambda}\left[\mathcal{A}^{F_k^{\$}}\left(1^\lambda\right)=1\right] - \Pr_{f\xleftarrow{\$}\{0,1\}^\lambda}\left[\mathcal{A}^{f^{\$}}\left(1^\lambda\right)=1\right]\right| \leq neg(\lambda).$$

Avec pour toute fonction G à valeur dans $\{0,1\}^\lambda$, $G^{\$}()$, la fonction qui renvoie $(x, G(x))$ avec x tiré uniformément aléatoire dans $\{0,1\}^\lambda$.

1. *Montrer qu'une FPA est une FPA faible.*
2. *Construire une fonction F qui est une FPA faible, mais pas une FPA.*
3. *Dire sans justification pour les constructions $\Pi_2, \Pi_{\text{ECB}}, \Pi_{\text{OFB}}, \Pi_{\text{CBC}}, \Pi_{\text{CFB}}, \Pi_{\text{CTR}}$ quelles propriétés de sécurité on peut atteindre en supposant seulement que F est faiblement pseudo aléatoire.*
4. *Montrer pour chacun des schémas qu'on ne peut pas atteindre mieux en construisant des attaques pour les propriétés de sécurité plus fortes en considérant que la fonction utilisée est F construite à la question 2.*

Exercice 26 (Chiffrement à nuf)**.** *Le but des systèmes de chiffrement à nuf est de ne pas recourir à de l'aléa pour chiffrer mais d'utiliser un nuf, c'est à dire une donnée utilisée une seule fois (nuf pour nombre utilisée une fois)*[10]*. On appelle l'ensemble des nuf $\mathcal{N}$.*

Définition 46. *Un schéma de chiffrement à nuf est un triplet d'algorithmes DPT :* $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$.
$\texttt{KeyGen} : \left\{\{0,1\}^\lambda\right\}_{\lambda\in\mathbb{N}} \to \mathcal{K}$
$\texttt{Enc} : \mathcal{K}\times\mathcal{N}\times\mathcal{M}\to\mathcal{C}$
$\texttt{Dec} : \mathcal{K}\times\mathcal{N}\times\mathcal{C}\to\mathcal{M}$.

1. *Adapter la définition de sécurité 34 pour les chiffrements à nuf.*
2. *Adapter la définition de sécurité 36 pour les chiffrements à nuf.*
3. *Construire un schéma de chiffrement à nuf cpa-sécurisé et faire la preuve de sécurité.*

10. en anglais, on parle de *nonce* qui vient de *number used once*, nombre utilisé une seule fois en anglais.

Chapitre 3

Authentification de messages

Dans ce chapitre, l'objectif considéré ne va plus être la confidentialité, mais l'authenticité [1] d'une information. On a vu dans le chapitre 1 des modèles de sécurité qui ont cette fonction, et on va donc grandement s'inspirer des modèles de sécurité développés dans ce chapitre.

3.1 CAM : premières définitions et premières constructions

Le contexte, est le suivant des utilisateurs appartenant à une entité commune cherchent à s'assurer de cette appartenance à cette entité lorsqu'ils communiquent. Évidemment, on va supposer que les membres de l'entité partagent un secret commun comme pour les schémas de chiffrement vus dans le chapitre 2. Cette hypothèse qui distingue le contexte des signatures vues dans le chapitre 1 implique qu'on catégorise ce schéma dans la famille de la cryptographie symétrique.

Définition 47 (Code d'Authentification de Messages)**.** *Un code d'authentification de messages est un duo de PPT :* $(\texttt{KeyGen}, \texttt{Mac})$*, avec* $\texttt{Mac}$ *déterministe :*

$\texttt{KeyGen} : \{1^\lambda\}_{\lambda\in\mathbb{N}} \to \mathcal{K}$
$\texttt{Mac} : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$.

L'utilisation est la suivante, un utilisateur qui veut authentifier un message m calcule le code associé $t := \texttt{Mac}(k, m)$ en utilisant la clef secrète k commune à tous les membres de l'entité concernée.

Un utilisateur qui veut vérifier qu'un message m est bien authentique par rapport au code t, en calculant de son côté $t' := \texttt{Mac}(k, m)$, puis en vérifiant si $t \stackrel{?}{=} t'$.

Remarque Comme on le verra dans l'exercice **??**, on peut généraliser la définition en y ajoutant un algorithme de vérification, et en autorisant $\texttt{Mac}$ à être probabiliste (de manière analogue au schéma de signature vu dans le chapitre 1).

Remarque On remarque que vu l'utilisation qu'on a en tête, à partir du moment où $\texttt{Mac}$ est un algorithme déterministe, toute paire d'algorithmes convient, et on a donc pas besoin de définir de propriétés de correction pour ce type de schéma.

On peut s'intéresser directement à la propriété de sécurité des CAM en reprenant ce qui a été fait avec les signatures dans le chapitre 1.

Définition 48 (Inforgeabilité des CAM)**.** *Un CAM est dit inforgeable si et seulement si :*

$$\Pr\left[\text{MacForge}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq neg(\lambda)$$

avec le jeu MacForge *défini dans la figure 3.1.*

On va pouvoir construire assez rapidement un schéma qui vérifie cette propriété de sécurité en réutilisant les FPA vues dans la définition 35. En effet, si il est dur de prédire $F_k(m)$, il est a fortiori encore plus difficile de calculer $F_k(m)$:

1. On peut voir l'utilisation de l'expression "intégrité", mais étant donné l'ambiguïté de cette notion (qui peut aussi faire révérence à la résilience au bruit "naturel"), on parlera plus d'authenticité ici.

$\underline{\text{MacForge}_{\mathcal{A},\Pi}(\lambda)}$:	$\underline{OMac_k(m)}$:	$\underline{\Pi_3}$:
$k \leftarrow \Pi.\texttt{KeyGen}\left(1^\lambda\right)$	$t \leftarrow \Pi.\texttt{Mac}_k(m)$	Soit F_k une fonction pseudo-aléatoire :
$Q \leftarrow \varnothing$	$Q := \{m\} \cup Q$	$\texttt{KeyGen}(1^\lambda)$:
$(m^*, t^*) \leftarrow \mathcal{A}^{OMac_k}\left(1^\lambda\right)$	Retourner t	$k \xleftarrow{\$} \{0,1\}^\lambda$
Retourner $\texttt{Mac}_k\left(m^*\right) \stackrel{?}{=} t^* \wedge m^* \notin Q$		Retourner k
		$\texttt{Mac} : t \xleftarrow{\$} F_k(m)$
		Retourner t

FIGURE 3.1 – Jeu d'inforgeabilité CAM, et un premier schéma

$\underline{\mathcal{B}_{\mathcal{A}}^{O(\cdot)}\left(1^\lambda\right)}$:	$\underline{\mathcal{B}'^{O(\cdot)}_{\mathcal{A}}(1^\lambda)}$:	$\underline{OMac^{O(\cdot)}(m)}$:
$Q \leftarrow \varnothing$	$Q \leftarrow \varnothing$	$Q := \{m\} \cup Q$
$(m^*, t^*) \leftarrow \mathcal{A}^{OMac^{O(\cdot)}(\cdot)}\left(1^\lambda\right)$	$(m^*, t^*) \leftarrow \mathcal{A}^{OMac^{O(\cdot)}(\cdot)}\left(1^\lambda\right)$	Retourner $O(m)$
Si $O\left(m^*\right) \stackrel{?}{=} t^* \wedge m^* \notin Q$	$t^{**} \xleftarrow{\$} \{0,1\}^\lambda$	
Renvoyer 1	Si $t^{**} \stackrel{?}{=} t^* \wedge m^* \notin Q$	
Sinon Renvoyer 0	Renvoyer 1	
	Sinon Renvoyer 0	

FIGURE 3.2 – Attaquants $\mathcal{B}$ et $\mathcal{B}'$ contre le jeu des FPA

Propriété 4. Π_3 *(figure 3.4) est un CAM sécurisé.*

On remarque tout d'abord que $\mathcal{B}^{F_k(\cdot)}\left(1^\lambda\right)$ avec k tiré uniformément dans $\{0,1\}^\lambda$ suit la même distribution que $\text{MacForge}_{\mathcal{A},\Pi}(\lambda)$. Donc

$$\Pr[\text{MacForge}_{\mathcal{A},\Pi}(\lambda) = 1] = \Pr_{k \xleftarrow{\$} \{0,1\}^\lambda}[\mathcal{B}_{\mathcal{A}}^{F_k(\cdot)}\left(1^\lambda\right) = 1]. \tag{3.1}$$

Appelons l'évènement $E := m^* \in Q$. Il est trivial de constater que

$$\Pr_{f \xleftarrow{\$} (\{0,1\}^\lambda)^{\{0,1\}^\lambda}}\left[\mathcal{B}_{\mathcal{A}}^{f(\cdot)}\left(1^\lambda\right) = 1 | E\right] = 0. \tag{3.2}$$

On remarque que lorsqu'on est dans le cas $\bar{E}$, dans l'exécution de $\mathcal{B}^{f(\cdot)}\left(1^\lambda\right)$, on se rend compte que $O(m^*)$ suit une distribution uniforme sur $\{0,1\}^\lambda$ indépendante de m^* et t^*.

$$\begin{aligned} &\Pr_{f \xleftarrow{\$} (\{0,1\}^\lambda)^{\{0,1\}^\lambda}}\left[\mathcal{B}_{\mathcal{A}}^{f(\cdot)}\left(1^\lambda\right) = 1 | \bar{E}\right] \\ =&\Pr_{f \xleftarrow{\$} (\{0,1\}^\lambda)^{\{0,1\}^\lambda}}\left[\mathcal{B}'^{f(\cdot)}_{\mathcal{A}}\left(1^\lambda\right) = 1 | \bar{E}\right] \\ =&\frac{1}{2^\lambda}. \end{aligned} \tag{3.3}$$

On suppose d'abord que $\Pr[\text{MacForge}_{\mathcal{A},\Pi}(\lambda) = 1] > \frac{1}{2^\lambda}$. Comme F est une FPA, on déduit de (3.1), et (3.3) qu'il existe une fonction négligeable $\epsilon(\lambda)$, tel que $\Pr[\text{MacForge}_{\mathcal{A},\Pi}(\lambda) = 1] - \frac{1}{2^\lambda} \leq \epsilon(\lambda)$.

Donc $\Pr[\text{MacForge}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2^\lambda} + \epsilon(\lambda)$. Ce qu'il fallait démontrer. □

Comme pour la section 2.2, on souhaite à présent gagner en efficacité sans faire de concession de sécurité, on va donc utiliser une technique très similaire en s'inspirant énormément du chiffrement CBC pour la construction CBC_{mac} (figure 3.3) :

Propriété 5. CBC_{mac} *(figure 3.3) est un CAM fortement sécurisé.*

Résultat admis. □

Soit (F_k) une fonction pseudo-aléatoire :
$\underline{\text{CBC}_{\text{mac}}}$:

$\texttt{KeyGen}\left(1^\lambda\right) : k \xleftarrow{\$} \{0,1\}^\lambda$; Retourner k
$\texttt{Mac}(m) : \left(0^\lambda, c_1, \ldots, c_n\right) \leftarrow \Pi_{\text{CBC},F}.\texttt{Enc}_k\left(n, m_1, \ldots, m_n\right)$
avec $c_0 = 0^\lambda$
Retourner c_n

FIGURE 3.3 – CBC-MAC

3.2 Chiffrement authentifié

Dans cette section, on va chercher à définir les propriétés d'authentification sur un système de chiffrement. On parle d'inforgeabilité lorsqu'il est difficile de construire une signature valide, on va donc naturellement étendre la notion au système de chiffrement en parlant d'inforgeabilité lorsque qu'il est difficile de construire un cryptogramme valide, où lorsqu'on dit "cryptogramme valide", on entend que le déchiffrement renvoie un fichier clair lisible (et donc pas une erreur).

Définition 49. *Un schéma de chiffrement* Π *est dit inforgeable (ou non malléable) si et seulement si :*

$$\Pr\left[\text{EncForge}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq neg(\lambda),$$

avec EncForge *(défini figure 3.4).*

A présent on peut combiner cette propriété avec la propriété de sécurité 36 pour dire ce qu'on souhaite :

Définition 50. *Un schéma de chiffrement* Π *est dit authentifié, si il est inforgeable et CCA-sécurisé.*

Pour atteindre cette propriété de sécurité, il paraît judicieux de se reposer sur ce qu'on déjà réussit à obtenir c'est à dire les propriétés 34 et 48. Imaginons donc qu'on a donc un schéma de chiffrement Π_{E}, et un CAM Π_{M}. La manière la plus naïve de combiner ces deux schémas reste la concaténation. C'est à dire le fait de considérer de chiffrer m, en calculant $\left(\Pi_{\text{E}}.\texttt{Enc}_{k_E}\left(m\right), \Pi_{\text{M}}.\texttt{Mac}_{k_M}\left(m\right)\right)$, avec k_E, k_M les clefs de sécurité des deux systèmes utilisés. Évidemment, ce système ne va pas être sécurisé dans le cas général, car le code d'authentification peut théoriquement faire fuiter de l'information (cf exercice **??**). Pour s'assurer que le code d'authentification ne fasse fuiter aucune information, on peut soit le chiffrer avec Π_{E} en même temps que le message (on explorera cette idée avec l'exercice **??**), soit authentifier un chiffré. C'est cette approche qu'on va retenir avec la construction Π_{ETM} (cf figure 3.4). Et comme on va le montrer avec la propriété suivante, on atteindra les objectifs de sécurité maximaux définis jusqu'à présent.

Propriété 6. *Si* Π_{E}*, est CPA-sécurisé, et* Π_{M} *est un schéma de CAM inforgeable. Alors* Π_{ETM} *(cf figure 3.4) est un chiffrement authentifié.*

3.3 Fonctions de hachage et applications

3.3.1 Premières définitions

Dans ce chapitre, on va se concentrer sur une primitive fondamentale : les fonctions de hachage. Le concept de haché doit se comprendre comme celui d'*empreinte cryptographique*. Comme pour toute primitive, il convient de définir formellement de quoi on parle.

Définition 51. *Une fonction de hachage paramétrée :* $(\texttt{Setup}, \texttt{Hash})$*, avec* `Hash` *déterministe,* ℓ *une fonction croissante* $\mathbb{N} \to \mathbb{N}$ *:*

$\texttt{Setup} : \{1^\lambda\}_{\lambda\in\mathbb{N}} \to \mathcal{K}$
$\texttt{Hash} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$.

Si `Hash` *à pour ensemble de départ* $\mathcal{K} \times \{0,1\}^n$ *au lieu de* $\mathcal{K} \times \{0,1\}^*$ *avec* n *une constante, on parle alors de fonction de hachage à entrée bornée.*

$\underline{\text{EncForge}_{\mathcal{A},\Pi}(\lambda)}$:
$k \leftarrow \Pi.\texttt{KeyGen}\left(1^\lambda\right)$
$Q \leftarrow \varnothing$
$(m,t) \leftarrow \mathcal{A}^{\texttt{Enc}_k}(1^\lambda)$
Retourner
$\texttt{Dec}_k(m) \neq \perp \wedge m \notin Q$

$\underline{\texttt{Enc}_k(m)}$:
$c \leftarrow \Pi.\texttt{Enc}_k(m)$
$Q := \{m\} \cup Q$
Retourner c

$\underline{\Pi_{\text{ETM}}}$:
Soit $\Pi_{\text{E}}, \Pi_{\text{M}}$, un schéma de chiffrement, et un CAM.
$\texttt{KeyGen}\left(1^\lambda\right)$:
$(k_E, k_M) \xleftarrow{\$} \{0,1\}^{2\lambda}$
Retourner(k_E, k_M)
$\texttt{Enc}_{k_E,k_M}(m)$:
$c \leftarrow \Pi_{\text{E}}.\texttt{Enc}_{k_E}(m)$
$t \leftarrow \Pi_{\text{M}}.\texttt{Mac}_{k_M}(c)$
Retourner(c,t)
$\texttt{Dec}_{k_E,k_M}((c_1,c_2))$:
Si $\Pi_{\text{M}}.\texttt{Mac}_{k_M}(c_1) \neq c_2$:
Retourner $\perp$
Sinon :
Retourner $\Pi_{\text{E}}.\texttt{Dec}_{k_E}(c_1)$

FIGURE 3.4 – Jeu d'inforgeabilité pour chiffrement, et la construction Chiffrement-puis-CAM

$\underline{\text{Coll}_{\mathcal{A},\Pi_{\text{H}}}(\lambda)}$:
$par \leftarrow \Pi_{\text{H}}.\texttt{Setup}\left(1^\lambda\right)$
$(m_1, m_2) \leftarrow \mathcal{A}\left(par\right)$
Retourner
$\Pi_{\text{H}}.\texttt{Hash}^{par}(m_1) \stackrel{?}{=} \Pi_{\text{H}}.\texttt{Hash}^{par}\left(m_2\right)$

FIGURE 3.5 – Jeu cryptographique de la résistance aux collisions

Le principe d'une empreinte est le suivant, elle doit caractériser son sujet de manière compacte. On peut voir cet aspect de plusieurs manières. Mais la plus naturelle est de penser à une sorte d'"injectivité calculatoire", c'est à dire qu'il est difficile de trouver deux éléments qui ont la même empreinte.

Définition 52. *Une fonction de hachage paramétrée H est dite résistante aux collisions si et seulement si :*

$$\Pr\left[\text{Coll}_{\mathcal{A},\Pi_H}\left(\lambda\right) = 1\right] \leq neg(\lambda),$$

avec Coll *le jeu défini en figure 3.5.*

On remarque une première chose qui différencie nettement les fonctions de hachages de toutes les primitives vues jusqu'à présent : il n'y a *a priori* aucune information secrète du challenger par rapport à l'attaquant. A ce titre, on peut dire qu'il s'agit d'une primitive *encore plus symétrique* que celles vues auparavant (GPA, FPA) où il y avait une asymétrie entre la connaissance des utilisateurs du système (qui possédaient la clef secrète), et un adversaire ignorant de cette information [2]. Pour le dire autrement si les empreintes digitales peuvent être une information sensible, le procédé pour produire des empreintes digitales à partir d'une paire de mains lui n'a rien de confidentiel. Ce qui est important avec cette notion d'empreinte cryptographique, c'est qu'elle puisse caractériser de manière unique (au sens calculatoire et non mathématique) une donnée. On verra par la suite une partie des applications d'une telle fonctionnalité.

3.3.2 Extension du domaine de définition avec la méthode de Merkle-Dåmgard

Tout d'abord, il faut partir du constat que bien souvent, on ne sait pas construire directement des fonctions de hachage à entrée non bornées.

On va donc voir avec la construction de Merkle-Damgård en figure 3.5, illustrée en figure 3.6 comment briser cette limitation.

2. Même si ça reste plus symétrique que dans la cryptologie asymétrique où il y a une asymétrie de l'information des utilisateurs du système.

$\underline{\Pi_{\mathrm{MD}}}$:
Soit Π_{H}, une fonction de hachage.
$\mathtt{Setup}\left(1^{\lambda}\right)$:
Retourner $\left(\Pi_{\mathrm{H}}.\mathtt{Setup}\left(1^{\lambda}\right)\right)$
$\mathtt{Hash}\left(m_1,\ldots,m_\ell\right)$:
$z_0 := 0^n$
$m_{\ell+1} := \ell$
Pour $i \in \{1,\ldots,\ell+1\}$:
$z_i := \Pi_{\mathrm{H}}.\mathtt{Hash}\left(z_{i-1}||m_i\right)$
Retourner $\left(z_{\ell+1}\right)$

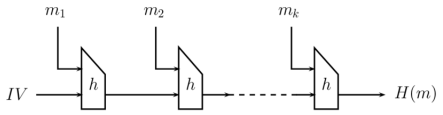

FIGURE 3.6 – Algorithme de Merkle-Dåmgard, et schéma illustrant ladite construction (sans le rajout de la taille à la fin)

$\underline{\mathcal{B}(par)}$:
$((m_1,\ldots,m_\ell),(m_1,\ldots,m'_\ell)) \leftarrow \mathcal{A}(par)$
$z_0 \leftarrow z' \leftarrow 0^n$
$m_{\ell+1} := \ell$
$m'_{\ell+1} := \ell'$
Pour $i \in \{1,\ldots,\ell+1\}$:
$z_i := \Pi_{\mathrm{H}}.\mathtt{Hash}\left(z_{i-1}||m_i\right)$
Pour $i \in \{1,\ldots,\ell'+1\}$:
$z'_i := \Pi_{\mathrm{H}}.\mathtt{Hash}\left(z'_{i-1}||m'_i\right)$
Si $(\ell' \neq \ell)$
Renvoyer $(\ell||z_\ell,\ell'||z_{\ell'})$
Sinon
$r := \min\left(i \in \mathbb{N}\{-\infty\}, z_i \neq z'_i\right)$
Si $r \neq -\infty$
Renvoyer $\left(z_r||m_{r+1}, z'_r||m'_{r+1}\right)$
Sinon $t := \min\left(i \in \mathbb{N}, m_i \neq m'_i\right)$
Renvoyer $\left(m_t||z_{t-1}, m'_t||z_{t-1}\right)$

FIGURE 3.7 – Attaquant $\mathcal{B}$ contre la fonction atomique h utilisant un attaquant $\mathcal{A}$ contre Merkle-Damgard

Théorème 5. *Soit $\left(n\left(\lambda\right)\right)_{\lambda\in\mathbb{N}} \in \mathbb{N}^{\mathbb{N}}$. Soit Π_{H}, une fonction de hachage ayant pour ensemble de départ $\{0,1\}^{2n(\lambda)}$ et ensemble d'arrivée $\{0,1\}^{n(\lambda)}$. Alors Π_{MD} est une fonction de hachage résistante aux collisions, avec pour ensemble de départ $\left(\{0,1\}^{kn(\lambda)}\right)_{k\in\mathbb{N}}$.*

On remarque tout d'abord que l'attaquant $\mathcal{B}$ de la figure 3.7 s'exécute en temps polynomial si c'est le cas de $\mathcal{A}$. On peut faire une disjonction de cas pour analyser ce que renvoie $\mathcal{B}$.

1. $\mathcal{A}$ ne renvoie pas une collision (soit car les images des messages sont distinctes), soit car les messages sont identiques. A ce moment là on ne garantit pas le succès de $\mathcal{B}$.
2. $\mathcal{A}$ renvoie une collision et $\ell \neq \ell'$. A ce moment là, on a $z_\ell = z'_{\ell'}$, et $m'_{\ell'+1} \neq m_{\ell+1}$, on en déduit donc que $(\ell||z_\ell) \neq (\ell'||z_{\ell'})$, et $\Pi_{\mathrm{H}}.\mathtt{Hash}\left(\ell||z_\ell\right) = \Pi_{\mathrm{H}}.\mathtt{Hash}\left(\ell'||z_{\ell'}\right)$. Donc $\mathcal{B}$ renvoie bien une collision pour Π_{H}.
3. $\mathcal{A}$ renvoie une collision et $\ell = \ell'$, et $\exists i \in [\![1;\ell]\!] : z_i = z'_i$. A ce moment là, on a $r > -\infty$, comme $z_{\ell+1} = z'_{\ell+1}$ (car $\mathcal{A}$ a renvoyé une collision pour Π_{MD}), et donc par définition de r : $z_r = z'_r$, et $z_{r+1} = z'_{r+1}$ (comme $r \leq \ell$, ces z's sont bien définis), on a donc : $(z_r||m_{r+1}) \neq \left(z'_r||m'_{r+1}\right)$, et $\Pi_{\mathrm{H}}.\mathtt{Hash}\left(z_r||m_{r+1}\right) = \Pi_{\mathrm{H}}.\mathtt{Hash}\left(z'_r||m'_{r+1}\right)$. Donc $\mathcal{B}$ renvoie bien une collision pour Π_{H}.
4. $\mathcal{A}$ renvoie une collision et $\ell = \ell'$, et $\forall i \in [\![1;\ell]\!] : z_i = z'_i$. A ce moment là, comme $m \neq m'$, on sait que t est bien défini (c'est une entier naturel), et donc par définition de t, on a $(m_t||z_{t-1}) \neq$

$\underline{\Pi_{\text{HTM}}}$:
Soit $\Pi_{\text{H}}, \Pi_{\text{M}}$, une fonction de hachage, et un CAM.
$\texttt{KeyGen}\left(1^{\lambda}\right)$:
$par \leftarrow \Pi_{\text{H}}.\texttt{Setup}(\lambda)$
$k_M \xleftarrow{\$} \Pi_{\text{M}}.\texttt{KeyGen}\left(1^{\lambda}\right)$
Retourner (par, k_M)
$\texttt{Mac}_{par,k_M}(m)$:
$h \leftarrow \Pi_{\text{H}}.\texttt{Hash}^{par}(m)$
$t \leftarrow \Pi_{\text{M}}.\texttt{Mac}_{k_M}(h)$
Retourner(t)

FIGURE 3.8 – Code d'authentification de messages sur $\{0,1\}^*$ basé sur le paradigme Hash-and-MAC

$\underline{\mathcal{B}_1(par)}$:	$\underline{OMac_1(m)}$:	$\underline{\mathcal{B}_2^{OMac}\left(1^{\lambda}\right)}$:	$\underline{OMac_2()}$:
$k_M \leftarrow \Pi_{\text{M}}.\texttt{KeyGen}\left(1^{\lambda}\right)$	$h \leftarrow \Pi_{\text{H}}.\texttt{Hash}^{par}(m)$	$par \leftarrow \Pi_{\text{H}}.\texttt{Setup}\left(\lambda\right)$	$Q := Q \cup \{m\}$
$Q := \emptyset$	$Q := Q \cup \{(m,h)\}$	$(m^*, t^*) \leftarrow \mathcal{A}^{OMac_2(\cdot)}\left(1^{\lambda}\right)$	$h \leftarrow \Pi_{\text{H}}.\texttt{Hash}^{par}(m)$
$(m^*, t^*) \leftarrow \mathcal{A}^{OMac_2(\cdot)}(1^{\lambda})$	$t \leftarrow \Pi_{\text{M}}.\texttt{Mac}(h)$	$h \leftarrow \Pi_{\text{H}}.\texttt{Hash}^{par}(m)$	$t \leftarrow OMac(h)$
$h^* \leftarrow \Pi_{\text{H}}.\texttt{Hash}^{par}(m^*)$	Retourner(t)	Retourner (h, t^*)	Retourner(t)
Si $\exists m' \vert (m', h^*) \in Q$			
Retourner (m^*, m')			

FIGURE 3.9 – Adversaire contre Π_{H}, puis contre Π_{M}, utilisant $\mathcal{A}$

$(m'_t || z_{t-1})$ on en déduit donc que $(\ell || z_\ell) \neq (\ell' || z_{\ell'})$, et $\Pi_{\text{H}}.\texttt{Hash}\left(\ell || z_\ell\right) = \Pi_{\text{H}}.\texttt{Hash}\left(\ell' || z_{\ell'}\right)$. Donc $\mathcal{B}$ renvoie bien une collision pour Π_{H}.

On déduit de cette analyse que la probabilité de succès de $\mathcal{B}$ est égale à celle de $\mathcal{A}$. □

3.3.3 Le paradigme Hash-and-MAC

Ainsi le haché d'une donnée la caractérise d'une manière unique (calculatoirement) et compacte. Il va donc de soit qu'authentifier l'empreinte d'une donnée est équivalent à authentifier la donnée elle même. Et comme l'empreinte est de taille constante, on va donc pouvoir utiliser des CAM qui fonctionnent sur des messages de taille constante (comme par exemple la construction Π_3).

Propriété 7. *Si Π_{H} est une fonction de hachage résistante aux collisions à ensemble d'entrées $\{0,1\}^*$, avec pour ensemble image $\mathcal{M}$, et Π_{M} un code d'authentification de messages inforgeable sur $\mathcal{M}$. Alors Π_{HTM} (défini figure 3.8) est un CAM fortement sécurisé.*

On pose l'évènement $E := \exists m' \vert\, (m', h^*) \in Q$.

$$\begin{aligned}\Pr\left[\text{MacForge}_{\mathcal{A},\Pi_{\text{HTM}}}(\lambda)\right] &= \Pr\left[\text{MacForge}_{\mathcal{A},\Pi_{\text{HTM}}}(\lambda) | E\right]\Pr[E] + \Pr\left[\text{MacForge}_{\mathcal{A},\Pi_{\text{HTM}}}(\lambda) | \bar{E}\right]\Pr[\bar{E}] \quad (3.4)\end{aligned}$$

On remarque que $\text{MacForge}_{\mathcal{A},\Pi_{\text{HTM}}}(\lambda)$ implique que $(m^*, \cdot) \notin Q$, alors $\text{MacForge}_{\mathcal{A},\Pi_{\text{HTM}}}(\lambda) \wedge E \implies m^* \neq m'$. Et dans ce cas, (m^*, m') forme une collision sur Π_{H}^{par}. On en déduit $\Pr\left[\text{MacForge}_{\mathcal{A},\Pi_{\text{HTM}}}(\lambda) | E\right] \leq \Pr\left[\text{Coll}_{\mathcal{B}_1,\Pi_{\text{H}}}(\lambda)\right]$.

On remarque aussi que $\text{MacForge}_{\mathcal{A},\Pi_{\text{HTM}}}(\lambda)$ implique que $\Pi_{\text{M}}.\texttt{Mac}(\Pi_{\text{H}}.\texttt{Hash}(m^*)) = t^*$. Et si on est dans $\neg E$, on en déduit $\text{MacForge}_{\mathcal{B}_2,\Pi_{\text{H}}}(\lambda)$.

De ces deux implications, on peut transformer (3.4) :

$$\begin{aligned}\Pr\left[\text{MacForge}_{\mathcal{A},\Pi_{\text{HTM}}}(\lambda)\right] &\leq \Pr\left[\text{Coll}_{\mathcal{B}_1,\Pi_{\text{H}}}(\lambda)\right]\Pr[E] + \Pr\left[\text{MacForge}_{\mathcal{B}_2,\Pi_{\text{M}}}(\lambda)\right](1 - \Pr[E])\,. \\ &\leq \max\left(\Pr\left[\text{Coll}_{\mathcal{B}_1,\Pi_{\text{H}}}(\lambda)\right], \Pr\left[\text{MacForge}_{\mathcal{B}_2,\Pi_{\text{M}}}(\lambda)\right]\right) = neg(\lambda).\end{aligned}$$

□

Chapitre 4

Cryptographie asymétrique

Dans toutes les constructions vues dans les chapitres précédents (si l'on met de côté les fonctions de hachage), on avait comme hypothèse que les utilisateurs partageaient une clef secrète en commun. On ne s'occupaient pas alors de savoir comment exactement cette clef était partagée (on s'assurait seulement que cette dernière soit raisonnablement petite). Si dans des contextes militaires du XIXème siècle cette hypothèse correspondait au cas d'usage, quand l'on regarde des systèmes d'information moderne comme celui d'internet où des millions puis des milliards d'utilisateurs peuvent potentiellement communiquer entre eux de manière décentralisée sans jamais s'être mis d'accord au préalable, l'hypothèse apparaît trop forte. En particulier le fait que chaque utilisateur ait une clef pour chacun des milliards d'utilisateurs stockée sur sa machine ne paraît pas raisonnable en pratique, en particulier car la distribution de telles clefs est un problème a priori aussi compliqué que celui de communiquer de manière confidentielle. Dans des contextes plus centralisés, où il existe une entité considérée comme un tiers de confiance (c'est à dire qu'on admet que cette entité aura accès à toutes les communications du système en clair) comme le système d'information d'une entreprise, on peut imaginer que ce tiers de confiance ait une clef partagée avec chacun des utilisateurs et que lorsque deux utilisateurs veulent communiquer, ils contactent ce tiers de confiance pour recevoir chacun via un canal sécurisé une clef commune aux deux correspondants. C'est sur ce principe que se base le protocole de *Needham-Schroeder*.

Mais, si on ne veut pas renforcer notre hypothèse d'un tiers de confiance (et donc affaiblir notre modèle de sécurité) On va donc chercher à résoudre un problème a priori impossible : échanger des informations de manière confidentielle sans s'être mis au courant au préalable d'une donnée secrète. Ceci semble impossible car *a priori* un observateur aura accès exactement aux mêmes informations que le destinataire légitime de la donnée sensible. Il est donc nécessaire que les deux correspondants (qu'on appellera dans la suite Alice et Bernard) aient chacun une information secrète (qui sera dans ce contexte inconnu, y compris par son camarade). Ainsi un potentiel adversaire n'aura pas les mêmes informations qu'aucun des correspondants.

4.1 Échange des clefs

Dans cette partie, on va se restreindre à l'objectif de partager une clef commune avec un interlocuteur via un canal non sécurisé sans avoir d'information commune à la base. On a dit qu'il est *structurellement nécessaire* que les deux interlocuteurs aient chacun une clef secrète personnelle. Et d'une manière ou d'une autre la clef commune devra dépendre de ces deux clefs secrètes (sinon on se retrouve dans le cas où un observateur extérieur a autant d'information sur la clef commune qu'un des deux interlocuteurs).

Il va être donc nécessaire que chacun des interlocuteurs envoie un message (dépendant de sa clef secrète) à son camarade, pour que ce dernier puisse calculer la clef commune.

Ses messages peuvent être envoyé de manière séquentielle ou parallèle. Comme, on est dans l'optique de ne pas affaiblir notre modèle a priori, on va supposer que les messages sont envoyés en parallèle (quitte à revenir sur cette hypothèse si l'on se rend compte de blocage).

Plus formellement :

Définition 53. *Un système d'échange de clef est constitué de deux algorithmes PPT :*

$\underline{\text{EtaClef}_{\mathcal{A},\Pi}(\lambda)}$:
$(k_A, m_A) \leftarrow \Pi.\texttt{Gen}(1^\lambda)$
$(k_B, m_B) \leftarrow \Pi.\texttt{Gen}(1^\lambda)$
$sk_0 \leftarrow \Pi.\texttt{KeyGen}(k_B, m_A)$
$sk_1 \xleftarrow{\$} \mathcal{K}_\lambda$
$b \xleftarrow{\$} \{0,1\}$
$b^* \leftarrow \mathcal{A}(m_A, m_B, sk_b)$
Renvoyer $b \overset{?}{=} b^*$

$\underline{\text{DH}_{\mathcal{A}}(\lambda, b)}$:
$(\mathbb{G}, g, p) \leftarrow \texttt{GroupGen}(1^\lambda)$
$x, y, z \xleftarrow{\$} \mathbb{Z}_p$
$g_0 \leftarrow g^{xy}$
$g_1 \leftarrow g^z$
$b^* \leftarrow \mathcal{A}(g, g^x, g^y, g_b)$
Renvoyer b^*

$\underline{\Pi_{\text{DH}}}$:
$\texttt{Gen}(1^\lambda)$:
$(\mathbb{G}, g, p) \leftarrow \texttt{GroupGen}(1^\lambda)$
$x \xleftarrow{\$} \mathbb{Z}_p$
Renvoyer (x, g^x)
$\texttt{KeyGen}(y, h)$:
Renvoyer (h^y)

FIGURE 4.1 – Modèle de sécurité d'établissement de clef, jeu de Diffie-Hellman, et système d'établissement de clef de Diffie-Hellman

1. $\texttt{Gen}\left(1^\lambda\right)$ *qui prend en entrée un paramètre de sécurité en unaire et renvoie une clef et un message* (k, m).
2. $\texttt{KeyGen}(k, m)$ *prend en entrée une clef et un message et renvoie une clef commune* sk.

On remarque qu'ici par simplicité on suppose qu'Alice et Bob utilise le même algorithme (encore une fois on reviendra sur ce principe si on voit que ça nous empêche d'arriver à nos fins).

A présent définissons la propriété de correction :

Définition 54. *On dit que* Π *est un système d'établissement de clef est correct si et seulement pour toutes exécutions :* $(k_A, m_A) \leftarrow \Pi.\texttt{Gen}(1^\lambda)$, $(k_B, m_B) \leftarrow \Pi.\texttt{Gen}\left(1^\lambda\right)$, *on a :*

$$\Pi.\texttt{KeyGen}\left(k_B, m_A\right) = \Pi.\texttt{KeyGen}\left(k_A, m_B\right).$$

On remarque que cette propriété implique que $\Pi.\texttt{KeyGen}$ soit déterministe.

A présent nous pouvons construire notre modèle de sécurité :

Définition 55. *On dit qu'un système d'établissement de clef est sécurisé si et seulement si :*

$$\left|\text{EtaClef}_{\mathcal{A},\Pi}(\lambda) - \frac{1}{2}\right| \leq neg(\lambda),$$

avec EtaClef *défini figure 4.1.*

Pour arriver à l'établissement d'un tel protocole, les primitives cryptographiques vues précédemment ne vont pas suffire. On va avoir besoin d'utiliser des structures algébriques un peu évolué. Dans les années soixante-dix, Diffie et Hellman conçoivent le protocole Π_{DH} (défini figure 4.1) permettant d'atteindre ce but. Leur protocole utilise évidemment des hypothèses calculatoires, en l'occurrence des hypothèses qui concernent des groupes :

Définition 56. *Soit* $\texttt{GroupGen}$ *un algorithme DPT. On dit qu'il s'agit d'un générateur de groupe de Diffie-Hellman si et seulement si :*

1. $\forall \lambda \in \mathbb{N} : \texttt{GroupGen}(1^\lambda)$ *renvoie un triplet* $(\mathbb{G}, g, p)$ *avec* p *un entier premier de taille* λ, $\mathbb{G}$ *un groupe de taille* p, *et* g *un générateur de ce groupe.*
2. $|\text{DH}_{\mathcal{A}}(\lambda, 0) - \text{DH}_{\mathcal{A}}(\lambda, 1)| \leq neg(\lambda)$,

avec DH *défini figure 4.1.*

Cette hypothèse nous permet de déduire :

Théorème 6. *Si* $\texttt{GroupGen}$ *est un générateur de groupe de Diffie-Hellman (56), alors* Π_{DH} *(défini figure 4.1) est un système d'établissement de clef sécurisé.*

Preuve vue en cours □

4.2 Chiffrement asymétrique

En pratique, dans certains contextes, on n'aime pas devoir faire plusieurs tours pour une communication qui se veut à sens unique (par exemple, lors de l'envoi d'un courrier numérique). On se rend compte qu'on peut alors d'inspirer de la philosophie de l'échange de clef, en imaginant que le receveur publie en amont sa "clef publique" m_B de manière à que quiconque veuille communiquer avec lui se crée une paire de clef (k_A, m_A), puis envoie $(m_A, c := \Pi_{\mathrm{MJ}}.\mathtt{Enc}(\mathtt{KeyGen}(k_A, m_B)))$. Le receveur pourra alors calculer $\Pi_{\mathrm{MJ}}.\mathtt{Dec}(\mathtt{KeyGen}(k_B, m_A), c)$.

L'avantage avec cette philosophie, c'est qu'on se rapproche de l'objectif canonique de la cryptographie : envoyer un message de manière confidentielle via un canal non sécurisé. Sauf que cette fois ci contrairement au cas symétrique, on aura même pas supposé le partage commun d'une information confidentielle entre les correspondants.

Étant donnée qu'il y a une asymétrie de l'information entre les usagers (l'envoyeur, et le receveur), on parle de chiffrement *asymétrique*.

Avant de construire un tel schéma formalisons proprement ce que l'on veut obtenir :

Définition 57. *Un système de chiffrement à clef publique est constitué de trois algorithmes PPT :*

1. $\mathtt{KeyGen}\left(1^\lambda\right)$ *prend un paramètre de sécurité en unaire, et renvoie une clef secrète sk et une clef publique pk.*
2. $\mathtt{Enc}(pk, m)$ *prend une clef publique pk, et un message m, et renvoie un cryptogramme c.*
3. $\mathtt{Dec}(sk, c)$ *prend une clef publique pk, et un cryptogramme c, et renvoie un message m.*

La notion de sécurité est analogue à la confidentialité pour le chiffrement symétrique. La différence majeure, est que la clef publique (c'est à dire la partie de la clef que le receveur va diffuser), n'est pas considérée comme une donnée sensible. Ainsi notre adversaire va la prendre en entrée avant même de choisir quelle paire de message il cherche à distinguer :

Définition 58. *Soit* Π*, un schéma de chiffrement à clef publique. On dit que le schéma* Π *est sécurisé contre les attaques à clairs choisis si et seulement si pour tout PPT* $\mathcal{A}$*, la valeur*

$$Adv^{\mathrm{cpa}}_{\mathcal{A},\Pi}(\lambda) := \left|\Pr[\mathrm{PubK}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(\lambda) = 1] - \frac{1}{2}\right|$$

est négligeable en λ*, avec* $\mathrm{PubK}^{\mathrm{cpa}}$ *défini figure 4.2.*

Historiquement, c'est le cryptologue El Gamal qui a transformé l'établissement de clef de Diffie-Hellman en un système de chiffrement à clef publique de la manière évoquée plus haut. On remarquera qu'étant donné qu'ici message et clef sont des éléments de groupes, on généralise notre notion de masque jetable au delà des chaînes de bits munies d'un xor (qui forme un groupe (il s'agit de $(\mathbb{F}_2^n, +)$), à n'importe quelle groupe. Ainsi la deuxième partie du chiffré sera $\left(g^{pk}\right)^r \cdot m$.

Théorème 7. *Si* $\mathtt{GroupGen}$ *est un générateur de groupe de Diffie-Hellman (56), alors le système de chiffrement* Π_{EG} *(défini figure 4.2) est un système de chiffrement à clef publique sécurisé.*

Preuve analogue à celle de Diffie-Hellman. □

4.3 Construction d'un schéma de signatures

A présent que nous sommes beaucoup plus outillés, nous pouvons nous attaquer aux problématiques d'authentification abordées dans le chapitre 1.

Comme on l'a vu pour les codes d'authentification, d'un point de vue calculatoire, le problème d'usurper une identité (c'est à dire de construire un message qui soit considéré comme valide) relève plus de problèmes calculatoires que de problèmes décisionnels. C'est assez logique en fin de compte : casser un système d'authentification *est* un problème calculatoire, tandis que deviner un bit d'information confidentiel (c-à-d 0 ou 1) est un problème décisionnel. Ainsi pour nous faciliter la tâche, et bien que ce ne soit pas obligatoire (après tout pour la construction Π_3 de la figure 3.1, on avait obtenu de l'authentification à partir du problème décisionel de "casser" une FPA), il sera plus pratique dans la suite de considérer le problème calculatoire de Diffie-Hellman défini ci dessous :

$\underline{\mathrm{PubK}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(\lambda)}$:
$(sk, pk) \leftarrow \Pi.\mathtt{KeyGen}\left(1^\lambda\right)$
$(m_0, m_1) \leftarrow \mathcal{A}\left(pk\right)$
$b \xleftarrow{\$} \{0,1\}$
$b^* \leftarrow \mathcal{A}\left(pk\right)$
Renvoyer $b \overset{?}{=} b^*$

$\underline{\Pi_{\mathrm{EG}}}$:
$\mathtt{KeyGen}(1^\lambda)$:
$(\mathbb{G}, g, p) \leftarrow \mathtt{BGroupGen}(1^\lambda)$
$x, x_1, x_2 \xleftarrow{\$} \mathbb{Z}_p$
Renvoyer $(x_1, x_2, (\mathbb{G}, g^x, p, g^x))$
$\mathtt{Enc}(pk, m)$:
$r \xleftarrow{\$} \mathbb{Z}_p$
Renvoyer $(g^r, m \cdot pk^r)$
$\mathtt{Dec}(sk, (c_0, c_1))$:
Renvoyer $\left(c_1 \cdot c_0^{-sk}\right)$

FIGURE 4.2 – Modèle de sécurité contre les attaques à clairs choisis, système de chiffrement public d'El Gamal

$\underline{\mathrm{CDH}_{\mathcal{A}}(\lambda, b)}$:
$(\mathbb{G}, g, p) \leftarrow \mathtt{GroupGen}(1^\lambda)$
$x, y \xleftarrow{\$} \mathbb{Z}_p$
$h \leftarrow \mathcal{A}\left(g, g^x, g^y\right)$
Renvoyer $h \overset{?}{=} g^{xy}$

$\underline{\mathcal{B}(\mathbb{G}, p, g, g_a, g_b, h))}$:
$h' \leftarrow \mathcal{A}(\mathbb{G}, p, g, g_a, g_b)$
Renvoyer $h \overset{?}{=} h'$

$\underline{q - \mathrm{SCDH}_{\mathcal{A}}(\lambda, b)}$:
$(\mathbb{G}, g, p) \leftarrow \mathtt{BGroupGen}(1^\lambda)$
$x \xleftarrow{\$} \mathbb{Z}_p$
$(r \in \mathbb{Z}_p, h) \leftarrow \mathcal{A}\left(g^x, \cdot, g^{x^q}\right)$
Renvoyer $h \overset{?}{=} g^{\frac{1}{x+r}}$

FIGURE 4.3 – Jeu du problème de Diffie-Hellman calculatoire, adversaire contre ledit problème, problème de Diffie-Hellman fort

Définition 59. *Soit* `GroupGen` *un algorithme DPT. On dit qu'il s'agit d'un générateur de groupe de Diffie-Hellman calculatoire si et seulement si :*

1. $\forall \lambda \in \mathbb{N}$: $\mathtt{GroupGen}(1^\lambda)$ *renvoie un triplet* $(\mathbb{G}, g, p)$ *avec* p *un entier premier de taille* λ, $\mathbb{G}$ *un groupe de taille* p, *et* g *un générateur de ce groupe.*
2. $\mathrm{CDH}_{\mathcal{A}}(\lambda) \leq neg(\lambda)$,

avec CDH *défini figure 4.3.*

Définition 60. *Soit* `BGroupGen` *un algorithme DPT. On dit qu'il s'agit d'un générateur de groupe bilinéaire de Diffie-Hellman calculatoirement q-fort si et seulement si :*

1. $\forall \lambda \in \mathbb{N}$: $\mathtt{GroupGen}(1^\lambda)$ *renvoie un triplet* $(\mathbb{G}, g, p)$ *avec* p *un entier premier de taille* λ, $\mathbb{G}$ *un groupe de taille* p, *et* g *un générateur de ce groupe.*
2. $\mathrm{CDHfort}_{\mathcal{A}}(\lambda) \leq neg(\lambda)$,

avec CDH *défini figure 4.3.*

Pour faire le lien avec l'hypothèse vue précédemment, on pourra montrer cette propriété :

Propriété 8. *Si* `GroupGen` *est un générateur de Diffie-Hellman, alors c'est un générateur de Diffie-Hellman calculatoire.*

Preuve reposant sur l'adversaire $\mathcal{B}$ de la figure 4.3 vue en cours. □

Chapitre 5

Le Modèle de l'oracle aléatoire

5.1 Preuve à divulgation nulle de connaissance

Définition 61. *Soit $\mathcal{L}$ un langage. Une paire de Machines de Turing polynomiales* $(\texttt{Prove}, \texttt{Verify})$ *est un système de preuve interactif pour un langage $\mathcal{L}$ si* `Prove` *est une machine de Turing déterministe et que :*

$$\langle \texttt{Prove}(w, x, r), \texttt{Verify}(x) \rangle = 1 \text{ } \textit{(Correction)}$$

Si en plus, il existe une fonction `Extract` *tel que pour tout x, si il existe un algorithme* $\texttt{Prove}_x$ *déterministe, tel que* $\langle \texttt{Prove}_x, T(x) \rangle = 1$*, alors* $\texttt{Extract}^{\texttt{Prove}_x}$ *renvoie x avec une probabilité non nulle. On dit que le système est extractable.*

Posons $\Pi(w, x)$ la transcription du protocole. On dit que le système est à parfaitement à divulgation nulle, si il existe un algorithme polynomial `Sim` *tel que,* $\langle \texttt{Prove}(w, x, \cdot), \texttt{Verify}(x) \rangle$ *suit la même distribution que* `Sim`*. On définit de la même manière la notion de calculatoirement à divulgation nulle, et statistiquement à divulgation nulle.*

5.2 Attaque à chiffré choisis