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 un protocole 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 censée 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 flou 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 de sécurité :

— 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 faisables 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 "naturels" (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 à l'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 définie, 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, et renvoie une une autre donnée.

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, machines 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 infinie.

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, de la manière dont on définit le temps de calcul ou du langage de programmation dans lequel il sera codé. En revanche si on fait varier la taille de l'entrée et qu'on regarde asympototiquement la courbe du temps d'exécution de l'algorithme, alors quel que soit le langage ou modèle ou définition de temps choisi, le temps de calcul sera le même *à une constante multiplicative près.*

Dans ce paradigme, tous les algorithmes linéaires (c'est à dire ayant un temps de calcul linéaire en la taille de l'entrée) 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 [2].

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 [3].

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).

2. La grande majorité des problèmes du monde réel ayant une résolution polynomiale se sont trouvés avoir des algorithmes ayant des temps de calcul bornables par des polynômes de petits degrés (2 ou 3)

3. 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.

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.

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 [4] 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 :* $\Pr\left[\text{Inforg}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \epsilon(t, \lambda)$ [5].

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 [6].

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.

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

5. 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 $[0; 1]^{\mathbb{N}^4}$.

6. 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.

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 $\Pi_{concatou}$:

$\texttt{KeyGen}(1^\lambda) : (sk, vk) \leftarrow \Pi_S.\texttt{KeyGen}(1^\lambda), (sk', vk') \leftarrow \Pi'_S.\texttt{KeyGen}(1^\lambda)$. *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 [7]).

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.*

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

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.*

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 $\left(\frac{1}{P(n)}\right)_{n\in\mathbb{N}}$*[8]*.*

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.*

8. 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.