Signatures numériques

christina.boura@uvsq.fr

13 avril 2018

Les cryptosystèmes que nous avons vu jusqu'à présent permettent de réaliser principalement deux types d'opérations : le chiffrement de données et l'échange d'une clé entre deux parties. La première opération peut être réalisée en utilisant par exemple les chiffrements par blocs DES et 3DES ou encore les chiffrements à clé publique RSA ou Elgamal. La deuxième opération peut être accomplie à l'aide du protocole Diffie-Hellman. Cependant nous allons voir que ces deux opérations ne nous permettent pas de régler tous les problèmes liés à la sécurité informatique. On introduit alors ici une nouvelle notion, qu'on appelle *signature numérique* dont on explique l'utilité. On argumente ensuite pourquoi ce principe doit être basé sur les algorithmes à clé publique.

1 La notion de la signature numérique

Traditionnellement, on signe un document (acte commercial, contrat, diplôme, ...) pour prouver son authenticité et pour s'engager sur son contenu. Par exemple, tous les diplômes de licence de l'université de Versailles, sont signés un par un par le président de l'université. Un diplôme ainsi signé est considéré comme légitime. La signature du président sur votre diplôme assure votre futur employé sur la légalité de votre diplôme. Même s'il est toujours possible de falsifier une signature manuscrite, on considère que les lois contre la falsification des documents officiaux, mis en place par la plupart des gouvernements, sont assez stricts pour empêcher la tentation. C'est la raison pour laquelle les signatures manuscrites ont un rôle très important dans la vie commerciale et juridique.

Les signatures numériques sont une analogie électronique aux signatures traditionnelles. Leur rôle est de reproduire les caractéristiques principales des signatures manuscrites, c'est-à-dire lier un document à son auteur, rendre l'auteur responsable du contenu en cas de problème juridique et être difficilement imitables. Pour satisfaire toutes ces caractéristiques les algorithmes cryptographiques sont employés.

On commence par analyser une première approche dans laquelle des algorithmes de chiffrement à clé secrète sont utilisés afin de générer une signature numérique. On verra très rapidement le problème principal de cette approche et on expliquera comment ceci peut être résolu en utilisant les cryptosystèmes à clé publique.

2 Le problème de bébés phoques ou : Pourquoi la cryptographie symétrique ne suffit pas

Supposons qu'on veut signer notre bail de location, qui est sous forme numérique. Notre signature doit être difficilement imitable afin d'éviter que n'importe quelle personne puisse l'utiliser pour signer à notre place. L'idée est alors d'échanger avec notre bailleur, à l'aide du protocole Diffie-Hellman, une clé secrète k, et d'employer ensuite k pour signer le bail. L'utilisation d'une clé secrète pour signer un document assure que les seules personnes capables de produire une signature légitime, sont les personnes en détention de cette clé. La prochaine étape est d'employer un algorithme de signature qui prendra en entrée deux données : le bail à signer et la clé secrète k et qui produira en sortie une signature du document, s. Une approche est alors d'utiliser un algorithme symétrique comme algorithme de signature, par exemple le chiffrement par blocs 3DES. On chiffre alors le bail avec 3DES en utilisant la clé k. Le résultat (c'est-à-dire le document chiffré) est la signature. Une fois l'opération terminé, on transmet le document chiffré, accompagné du document en clair à notre propriétaire. Celui-ci, afin de vérifier la signature, chiffre le bail en clair avec la clé partagée k en utilisant 3DES. Si le résultat est identique au document chiffré, alors le propriétaire peut s'assurer que c'est bien nous qui avons chiffré le bail puisque on est les seuls à être en possession de la clé secrète k.

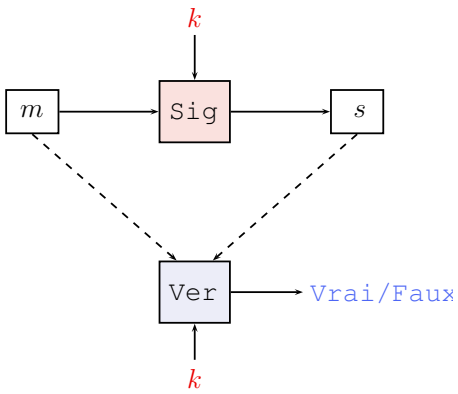


FIGURE 1 – Mécanisme de signature en utilisant une clé partagée k entre deux entités.

Plus formellement, si m est le message qu'on veut signer, on utilise une clé secrète k et un algorithme symétrique `Sig`. Cet algorithme prend en entrée le message m et la clé k et produit une signature s. La personne qui veut vérifier la signature doit être en possession de la clé secrète k. Cette personne reçoit le message m et la signature s. Elle applique ensuite un algorithme `Ver` qui prend en entrée m, s et k et qui donne en sortie la réponse "vrai" si la signature est valide et "faux" si elle ne l'est pas. Dans notre scénario, l'algorithme `Ver` est juste l'algorithme `Sig` avec de plus un test qui vérifie si le résultat de `Sig` appliqué à m et k est égal à s ou pas. Cette situation est illustrée dans la figure 1.

On verra maintenant avec un exemple que cette approche présente un problème fondamental : Elle ne protège pas la personne qui signe et la personne qui vérifie, l'un contre l'autre.

2.1 Problème de l'approche

Marc travaille dans une petite entreprise à Versailles, mais son salaire ne lui suffit pas pour acheter la voiture de ses rêves, une voiture de course très chère. Il envisage alors sérieusement de trouver un nouveau travail et il est même prêt à aller travailler très loin pour une courte période. Un ami de Marc, d'éthique douteuse, vivant aux États-Unis, lui dit qu'une grande entreprise pétrolière américaine, *Petrol King* cherche à embaucher des personnes pour une tache très particulière : tuer des bébés phoques vivant sur la banquise de l'Alaska. Ces derniers, à cause de leur petite taille ont tendance à se coincer dans les moteurs des pétroliers, créant pleins de problèmes techniques à ceux-ci et faisant perdre des millions de dollars à Petrol King. L'entreprise n'arrive pas à recruter des gens à cause du climat très hostile de l'Alaska et la cruauté de la tache demandée. L'ami de Marc lui précise que ce travail serait, bien étendu, particulièrement bien rémunéré. Marc hésite un peu, mais l'envie de s'acheter la belle voiture l'entraîne rapidement à prendre la mauvaise décision et à accepter le poste. Puisque Marc ne peut pas se déplacer physiquement jusqu'à Washington, le siège du géant américain, il signe le contrat à distance. L'entreprise et lui échangent, en utilisant le protocole Diffie-Hellman, une clé k, que Marc utilise pour signer le contrat.

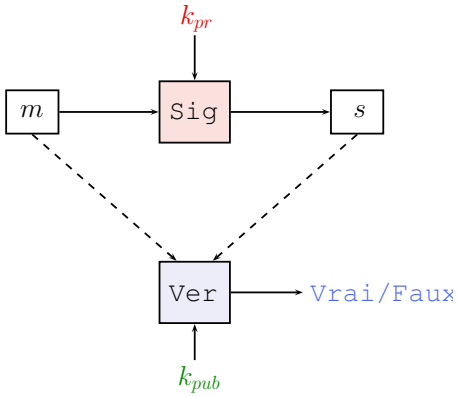
Cependant, la copine de Marc, une militante de Greenpeace, menace de le quitter en apprenant sa décision. Marc est très attaché à sa copine, qui lui prépare des petits mets délicieux. Marc décide alors de renoncer à son contrat. Mais il est trop tard, il est signé. Marc décide alors de tricher, en soutenant qu'il n'a jamais signé de contrat avec Petrol King. Malheureusement pour lui, Petrol King, ne compte pas laisser tomber cette affaire et entraîne Marc en justice. L'avocat de Petrol King, présente au juge le contrat signé. Il soutient que c'est forcement Marc qui a signé le contrat, car il est le seul à être en possession de la clé secrète k. Cependant, l'avocat de Marc, qui connaît très bien son boulot et a pris des cours de cryptographie à l'Université de Versailles, va expliquer au juge que Petrol King est également en possession de la clé k et c'est donc Petrol King qui a voulu entraîner Marc dans cette affaire douteuse, après avoir essayé longtemps d'embaucher quelqu'un pour ce poste odieux. Devant cette situation, il est impossible pour le juge de décider qui parmi Marc ou Petrol King a signé le contrat.

Cette situation peut paraître complètement irréelle et très peu probable, mais ceci n'est pas exactement comme ça. Dans plusieurs situations il est important de pouvoir prouver à une tiers personne (par exemple un juge) qui est le détenteur de la signature. Plus précisément, le juge doit pouvoir conclure de façon sure qui a signé le message, même si toutes les parties sont malhonnêtes. Ceci ne peut pas être possible avec

l'utilisation d'un cryptosystème symétrique, car l'utilisation d'un tel système implique que plus d'une personne connaît la clé secrète qui a permis de générer la signature. Pour pouvoir faire une conclusion sûre, la clé qui sert à signer doit être exclusivement détenue par une seule personne. La solution à ce problème se trouve alors dans l'utilisation de la cryptographie à clé publique.

3 Générer des signatures avec des algorithmes à clé publique

La procédure pour générer une signature numérique est la même que celle que nous venons de voir, à un détail près : la clé qui sert à générer la signature est la clé privée de la personne qui signe, tandis que la clé qui sert à vérifier la signature est la clé publique de cette personne. Cette procédure est illustrée dans l'image suivante.



Imaginons que Bob veut signer un message m. Il utilise pour ceci un algorithme de signature `Sig` qui prend en entrée le message m et la clé privée k_{pr} de Bob. Puisque Bob garde sa clé privée précieusement secrète, seulement Bob peut signer un message avec cette clé. Une fois que la signature s est produite, Bob concatène à cette signature le message m et envoie le couple (m, s) à Alice. Il est important de noter qu'une signature numérique seule n'a aucune utilité si elle n'est pas accompagné du message. Une signature numérique sans le message est équivalente à une signature manuscrite sur un bout de papier qui n'est pas accompagné par le contrat ou le document à signer.

La signature numérique est dans la pratique un grand nombre entier, constitué par exemple de 1024 ou 2048 bits. Alice qui reçoit cette signature doit être en mesure de vérifier si la signature est valide ou pas. Pour cela, une fonction de vérification est indispensable. Une telle fonction prend en entrée m et s, mais aussi la clé publique de Bob, afin de pouvoir lier la signature à Bob. La fonction de vérification n'a que deux sorties possibles : si m a été signé avec la clé privée liée à la clé publique qui possède Alice, alors la fonction renvoie "vrai", sinon elle renvoie "faux".

On peut maintenant voir que la propriété principale des signatures numériques est vérifiée. On peut remonter sans ambiguïté à la personne qui a produit un message signé, puisque une signature valide ne peut pas être produite qu'avec la clé privée unique de cette personne. De cette façon on peut prouver que la personne qui a signé le message est celle qui possède la bonne clé privée. Ce type de preuve est tout à fait légitime et est en général accepté au tribunal par un juge.

Il faut ici noter que ce protocole n'assure pas la confidentialité du message puisque ceci est envoyé en clair. Si on souhaite rendre le message secret, on peut en parallèle le chiffrer (voir TD).

4 Services de sécurité

On discute dans cette partie les *services de sécurité* qui peuvent être atteints avec l'utilisation des signatures numériques. Les *services de sécurité* sont les objectives qu'un système de sécurité peut assurer. Les services les plus importants sont :

Confidentialité : L'information reste secrète pour toutes les personnes qui ne sont pas autorisés à l'avoir.

Intégrité : Le message n'a pas été modifié pendant le transfert.

Authenticité du message : L'origine du message est authentique.

Non-répudiation : L'expéditeur du message ne peut pas nier la création du message.

On peut alors voir que la signature de garantit pas la confidentialité du message, mais ceci n'est pas un problème, puisque "cacher" le message des regards indiscrets n'est pas forcement le but dans ce cas. De l'autre côté, les services d'intégrité et d'authenticité du message sont bien assurés. On peut aussi noter que l'intégrité est un service qui n'est pas assuré par les signatures manuscrites. En effet, la présence d'une signature manuscrite en bas du document, n'empêche pas quelqu'un de modifier de façon subtile le contenu du message. Finalement, la non-répudiation est également atteinte avec une signature numérique basée sur la cryptographie à clé publique. Ceci n'est pas le cas, comme on vient de le voir, d'une signature numérique basée sur la cryptographie symétrique. Assurer la non-répudiation est une des raison principales pour laquelle on utilise la cryptographie à clé publique pour réaliser une signature à la place d'un algorithme de la cryptographie symétrique.

Différentes applications ont besoin de services de sécurité différents. Par exemple, pour une boîte mail d'usage privée, les trois premiers services sont désirables, tandis que dans le cas d'une boîte mail d'une société pouvoir assurer la non-répudiation peut être extrêmement utile. Dans une autre situation, afin de sécuriser les mises à jour logiciel d'un téléphone portable, les objectives cruciales sont l'intégrité et l'authenticité puisque le fabricant veut assurer que seulement les mises à jour autorisés sont exécutés sur le dispositif. Il faut noter ici que l'authenticité du message implique toujours l'intégrité, tandis que l'inverse n'est pas vrai. De façon générale, déterminer les services de sécurité désirés pour un système donné, est une tache qui dépend principalement de l'usage qu'on veut faire au système.

Les quatre services de sécurité décrits ici peuvent être atteints avec divers algorithmes cryptographiques. Pour la confidentialité, on utilise principalement les algorithmes symétriques comme les algorithmes de chiffrement par blocs ou les algorithmes de chiffrement à flot. L'intégrité et l'authenticité du message peuvent être atteints à l'aide des signatures numériques ou des codes d'authentification de message ou MAC (pas vu pendant ce cours) et la non-répudiation peut être atteinte avec les signatures électroniques.

5 La signature RSA

La signature RSA est basée sur le cryptosystème RSA. La sécurité de ce dernier repose sur la difficulté de factoriser le produit de deux grands nombres premiers. Le schéma de signature a été décrit en 1978 et reste le schéma de signature le plus utilisé dans la pratique.

Supposons que Bob veut envoyer un message signé m à Alice. Bob possède déjà un couple de clés RSA, une clé publique $k_{pub} = (n, e)$ et une clé privée $k_{pr} = d$. On peut supposer que le message que Bob veut signer est un entier entre 1 et $n-1$. Comme on peut le voir dans la figure 2, Bob calcule la signature numérique s du message m, en chiffrant m avec sa clé privée k_{prB}. Puisque Bob est la seule personne en possession de sa clé privée, cette procédure garantie automatiquement que Bob est l'auteur du message signé. Bob concatène la signature s au message m et envoie le couple (m, s) à Alice. Alice reçoit le message signé et le déchiffre en utilisant la procédure RSA avec la clé publique de Bob, k_{pubB}. Cette procédure produit un message m'. Si m égale m', Alice sait que la personne qui a produit la signature était en possession de la clé privée de Bob et comme Bob est la seule personne à avoir accès à cette clé, alors elle sait que c'est Bob qui a signé le message. Ceci assure l'authenticité du message. En parallèle, Alice peut s'assurer que le message n'a pas été altéré pendant le transfert ce qui assure l'intégrité.

On montre maintenant que ce protocole est correct, c'est-à-dire que le processus de vérification retourne "vrai" si le message et la signature n'ont pas été altérés pendant la transmission. On commence par l'opération de vérification $s^e \mod n$:

$$s^e = (m^d)^e = m^{de} \equiv m \mod n.$$

D'après la relation mathématique entre la clé publique et la clé privée

$$de \equiv 1 \mod \phi(n),$$

donc élever n'importe quel entier $m \in \mathbb{Z}_n$ à la puissance de, résulte en l'entier m.

On peut voir que dans le cas de la signature, les clés privées et publiques sont interchangées par rapport au chiffrement RSA. Pendant le chiffrement RSA, on chiffre avec la clé publique, tandis que pour le protocole de signature on utilise la clé privée pour l'opération de chiffrement. De la même façon, on utilise la clé publique pour vérifier (déchiffrer) dans le cas de la signature mais on utilise la clé privée pour déchiffrer dans le cadre du chiffrement classique.

On s'intéresse ensuite brièvement à la sécurité de la signature RSA en présentant une attaque connue sous le nom de *forge existentielle*.

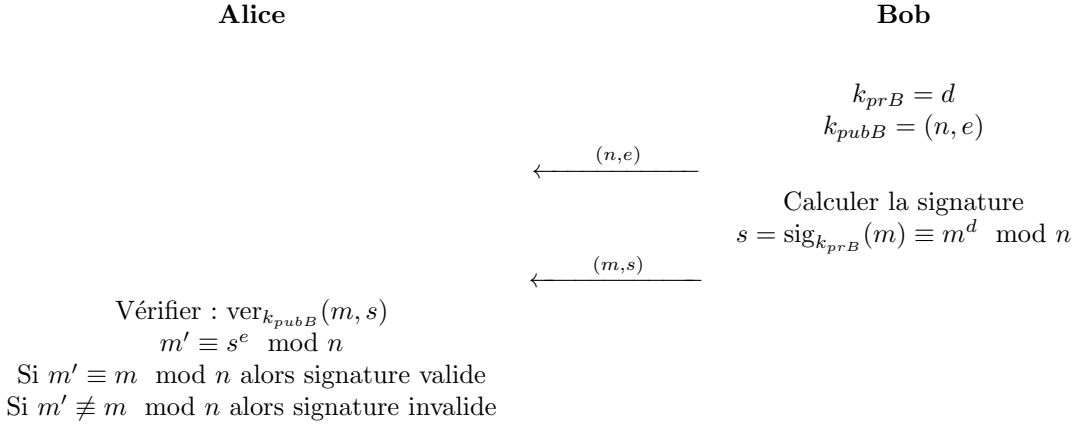


FIGURE 2 – Le protocole de la signature RSA

5.1 Forge existentielle

La forge existentielle est une attaque contre la signature RSA dans laquelle un attaquant, disons Oscar, essaie de tromper la fonction de vérification en générant une signature valide pour un message aléatoire m. Plus précisément, Oscar se fait passer pour Bob d'après Alice et essaie de générer une signature qui sera validée par Alice comme étant authentique. Cette attaque est montrée ci-dessous.

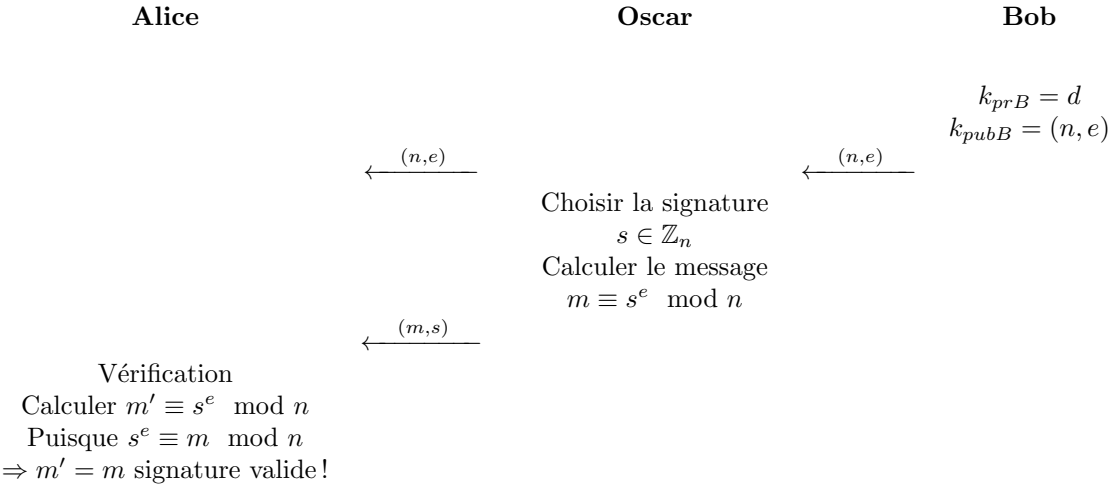


FIGURE 3 – Forge existentielle contre la signature RSA

On peut voir que puisque Alice fait exactement les mêmes calculs qu'Oscar, elle vérifiera toujours la signature comme étant valide. Cette attaque est cependant assez bizarre ; si on observe bien les étapes qu'effectue Oscar, on se rend compte qu'Oscar choisit d'abord la signature et puis calcule le message. En conséquence, il ne peut pas contrôler la sémantique du message m, donc les conséquences de cette attaque dans la pratique sont très minimes. Par exemple, il est très difficile pour Oscar de générer avec cette procédure un message du type "Transfert de 10000 euros sur le compte d'Oscar". Néanmoins, il est assez gênant qu'un processus de vérification automatique ne peut pas reconnaître la forge.

Dans la pratique et afin d'éviter ce genre d'attaques, un processus qui s'appelle rembourrage ou padding est appliqué. Il consiste à imposer une forme particulière aux messages m de façon qu'un message qui ne respecte pas cette forme soit considéré comme illégitime. On peut par exemple imposer que les 124 derniers bits du message à signer pour un module RSA-1024 soient tous égaux à 1. De cette manière un message légitime n'a qu'une chance sur 2^{124} de se produire.

6 Fonctions de hachage

Les fonctions de hachage sont des très importantes primitives cryptographiques et elles sont largement utilisées dans divers protocols. Une fonction de hachage prend en entrée un message de taille quelconque et calcule un *condensé* de taille fixe de ce message. Cette valeur hachée du message peut être vue comme une *empreinte* de celui-ci. Contrairement à tous les autres algorithmes introduits pendant ce cours, les fonctions de hachage n'utilisent pas de clé. Les fonctions de hachage sont utilisées pour les schémas de signature électronique, mais elles ont également beaucoup d'autres utilisations. Elles sont notamment utilisées pour la protection des mots de passe, pour garantir l'intégrité des données ou encore comme générateurs des suites pseudo-aléatoires.

6.1 Signatures électroniques et fonctions de hachage

Les fonctions de hachage sont des composantes indispensables des schémas des signatures électroniques. Dans le cas de la signature RSA, le message à signer ne peut pas dépasser la taille du module n. La taille de ce module en pratique se trouve entre 1024 et 3072 bits, qui se traduit en 128-384 octets seulement. Le problème est que la plupart de messages qu'on veut signer, sont bien plus longs que cela. La question qui se pose est alors la suivante : **Comment calculer efficacement des signatures pour des longs messages ?** Une approche naturelle serait d'utiliser la même idée que pour le mode opératoire ECB pour les chiffrements par blocs : diviser le message m en t blocs m_i de taille plus petite que celle permise par l'algorithme de signature, et signer chaque message indépendamment, comme illustré par la figure 4.

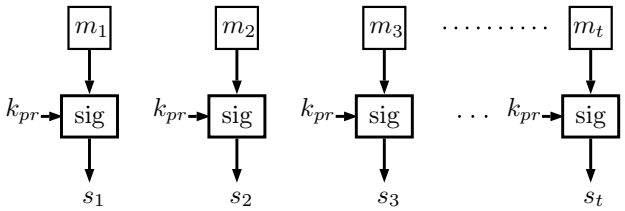


FIGURE 4 – Une première approche pour signer des messages de grande taille. Cette approche présente des problèmes de sécurité et des sérieux désavantages de mise en œuvre.

Cependant cette approche présente quelques sérieux problèmes :

Coût calculatoire très élevé Dans le cas de la signature RSA (même problème pour les autres schémas de signature) les opérations d'exponentiation à effectuer sont des opérations assez coûteuses. Puisque la taille des blocs du message est assez petite, plusieurs blocs doivent être hachés lors de la signature d'un long message. Donc, cette façon de signer des messages longs est en général assez lente. De plus, il ne faut pas oublier que les mêmes opérations vont devoir être effectuées de la part de la personne qui vérifie la signature.

Signatures volumineuses Evidemment, les signatures produites par cette approche ont la même taille que les messages signés. Donc, si un message fait 1 gigaoctet, la signature de ce message ferra la même taille. Comme le message et la signature doivent tous les deux être transmis pour la vérification, 2 gigaoctets seront envoyés dans cet exemple.

Problèmes de sécurité L'approche présentée ci-dessus est susceptible à des nombreuses attaques. Oscar pourrait par exemple enlever certains blocs du message ainsi que les blocs de signature correspondants, re-ordonner les différents blocs ou construire de nouveaux messages et des signatures à partir d'anciens morceaux de messages et de signatures. Même si un attaquant n'est pas capable de manipuler chaque bloc indépendamment, cette méthode n'offre aucune protection pour l'intégralité du message.

En conséquence, pour des raisons de sécurité ainsi que pour des questions de performance, on aimerait avoir une *seule signature de petite taille* pour un message de n'importe quelle longueur. La solution à ce problème est donnée par des *fonctions de hachage*. Si on avait une fonction h capable de calculer une

empreinte de petite taille pour un message $m = (m_1, \ldots, m_t)$, on pourrait effectuer la signature comme montré à la figure 5.

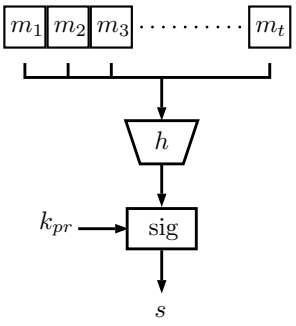


FIGURE 5 – Signature des longs messages à l'aide d'une fonction de hachage h.

Si on suppose qu'on possède une telle fonction, le protocole pour un schéma de signature numérique à l'aide d'une fonction de hachage est décrit à la Figure 6. On suppose ici que Bob veut envoyer un message signé à Alice.

Alice		**Bob**
		$z = h(m)$
		$s = \text{sig}_{k_{prB}}(z)$
	$\xleftarrow{(k_{pubB}, m, z)}$	
$z' = h(m)$		
$ver_{k_{pubB}}(s, z') = \text{vrai/faux}$		

FIGURE 6 – Protocole de signature en utilisant une fonction de hachage

Bob calcule le haché du message m par une fonction de hachage. Ensuite, il signe le haché z à l'aide de sa clé privée, k_{prB}. De l'autre côté, Alice calcule le haché $z' = h(m)$ du message reçu et vérifie la signature avec la clé publique, k_{pubB}, de Bob. On remarque, que la signature ainsi que la vérification s'effectuent tous les deux sur le haché plutôt que sur le message. De cette façon, le haché représente en quelque sorte le message. Pour cette raison, le haché est souvent appelé l'*empreinte* du message.

6.2 Définition formelle

Définition. Une fonction de hachage h est une fonction qui prend en entrée un message de taille aléatoire m, et donne en sortie un condensé de taille fixe, n.

$$\begin{array}{rcl} h : \{0,1\}^* & \to & \{0,1\}^n \\ m & \mapsto & h(m) \end{array}$$

L'*empreinte* d'un message, c'est-à-dire son image par la fonction de hachage, appelé aussi *haché*, sert à la représenter et permet facilement son identification.

La taille en bits des empreintes, n, varie selon la fonction de hachage utilisée. Pour les fonctions de hachage couramment utilisées n varie entre 128 et 512 bits.