CHAPITRE 4

FONCTIONS DE HACHAGE

4.1 Introduction

Une fonction de hachage est une composante fondamentale dans presque tous les systèmes cryptographiques et dans plusieurs applications de sécurité. Les scénarios courants de son utilisation vont de la **réduction de la taille** des données à signer, au contrôle de **l'intégrité** d'un fichier en passant par **l'authentification** des utilisateurs à travers un réseau. Les fonctions de hachage calculent le condensé d'un message binaire, ce qui représente une chaîne de bits courte et de longueur fixe. Pour un message donné d'une longueur arbitraire, le condensé, ou le haché, est considéré comme son empreinte (c'est-à-dire, une représentation unique et compacte du message). Grâce à ces caractéristiques, les applications des fonctions de hachage en cryptographie sont multiples. Dans ce chapitre nous allons introduire les principes et les concepts de base des fonctions de hachage.

4.2 Défnition

Une fonction de hachage **H** est un algorithme déterministe et efficace, qui prend comme entrée une donnée (un message) binaire **M** de taille quelconque, et produit à la sortie un haché **h** de taille fixe (en général entre 128 et 512 bits). Cet haché, appelé aussi "**condensé**" ou "**empreinte**" doit dépendre de tous les bits du message, et il est utilisé comme représentant comprimé de celui-ci.

$$\text{H: } (0,1)^* \rightarrow (0,1)^n$$
$$\text{h=H(M)}$$

4.3 Propriètés des fonctions de hachages

Soit **H** une fonction de hachage, on note **h = H(M)**, alors **M** est appelé **préimage** de h. Les propriètes à vérifier dans une fonction de hachage à utiliser pour la cryptographie sont comme suit:

- **Rapidité** : la valeur de haché se calcule très rapidement.
- **Propriété de compression** : quelque soit la taille du message M, il est facile de calculer le haché H(M), et sa la taille est fixe, elle est généralement inférieur à celle du message ;

- **Non-continuité**: un changement léger dans le message en entrée de la fonction de hachage doit produire une sortie complètement différente.
- **Résistance à la préimage** : étant donné **y**, il est difficile de trouver un **M** tel que **y = H(M)** (fonction à sens unique);
- **Résistance à la deuxième préimage** : étant donné un **h** et un **M** tel que h = H(M), il est très difficile de trouver un autre **M'** $\neq$ **M** tel que **h=H(M')**;
- **Résistance à la collision** : il est très difficile de trouver **M** et **M'** tel que **M** $\neq$ **M'** et **H(M')= H(M)**.

4.4 Notions de sécurité des fonctions de hachage

Dans le domaine de la sécurité informatique, les fonctions de hachage sont utilisées comme brique de base de différents protocoles cryptographiques. La sécurité de ces protocoles est basée essentiellement sur la résistance de la fonction de hachage à différent types d'attaques. Pour pouvoir être utilisée en cryptographie, une fonction de hachage doit résister à 3 types d'attaques: l'attaque sur la pré-image, l'attaque sur la seconde pré-image et l'attaque par collision.

- **Attaque sur la pré-image**: Dans cette attaque, on donne à l'attaquant un haché, et il doit trouver un message qui produit ce haché. Formellement : étant donné une valeur **h** de l'ensemble des empreintes, trouver un message **M** correspondant, tel que H(M) = h. La résistance à une telle attaque veut dire que la fonction est non-inversible.
- **Attaque sur la seconde pré-image**. Dans une attaque sur la seconde pré-image, on donne un message à l'attaquant, et il doit trouver un autre produisant le même haché que le premier. Formellement: étant donné un message **M** et son haché **h** = H(M), trouver **M'** différent de M tel que H(M') = h.
- **Attaque par collision**. Dans ce type d'attaques, on demande à l'attaquant de trouver deux messages arbitraires ayant la même empreinte. Formellement : trouver deux messages quelconques différents **M** et **M'** tels que H(M') = H(m).

4.5 Conception des fonctions de hachage

La plupart des fonctions de hachage sont basées sur l'itération d'une fonction de compression qui traite un nombre fixe de bits. Le message à hacher est divisé en blocs d'une certaine longueur où le dernier bloc est éventuellement complété avec des bits supplémentaires. Ce concept a été présenté pour la première fois par Michael O. Rabin en 1978. Ainsi, une fonction de hachage peut être composée de deux processus différents.

- Le premier processus est souvent une fonction de compression ou une permutation, permettant de traiter un bloc de données de taille fixe.
- Le deuxième, appelé algorithme d'extension de domaine est un mécanisme reliant les sorties de la fonction de compression interne, d'une itération à une autre pour former le haché final

4.5.1 Aperçu sur les Fonctions de compression

La compression de données est le premier objectif des fonctions de hachage. Une fonction de hachage doit comprimer la donnée en entrée, de telle sorte que la valeur de hachage résultante. Pour cette raison, les fonctions de compression sont au cœur des fonctions de hachage. Leur construction est un problème complexe, mais il existe plusieurs manières pour y parvenir. Nous allons brièvement introduire les constructions les plus importantes:

- **Basées sur des algorithmes de chiffrement par bloc**: la motivation principale de l'utilisation d'un chiffrement par bloc comme fonction de compression est la minimisation des efforts de conception. En outre, si les opérations de chiffrement et de hachage sont utilisées ensemble, l'utilisation des mêmes algorithmes pour le chiffrement et le hachage permettra de réduire les coûts de mise en œuvre. Les premières fonctions de hachage à base de chiffrement par blocs utilisent **DES** comme fonction de compression. La plupart des fonctions de hachage largement utilisées, comme MD4, MD5, SHA-1 et SHA-2, utilisent un algorithme de chiffrement par blocs comme fonctions de compression.

- **Basées sur des algorithmes de chiffrement par flux**: Ce type de constructions a été introduit pour des raisons de vitesse et d'efficacité. La première fonction de hachage a base de chiffrements par flux est la fonction **PANAMA** (1998). les fonction de hachage (MD5, SHA-1, etc.) appliquent un grand nombre de tours sur un bloc, alors que PANAMA effectue un seul "tour" complexe mais parallèle. PANAMA a l'avantage d'être polyvalente, elle peut jouer le rôle d'une fonction de hachage, d'un chiffrement par flux ou encore d'un code d'authentification de messages, Malheureusement, PANAMA a été cryptanalisée par ses concepteurs pour une attaque qui peut générer une collision. RC4-Hash est une autre fonction de hachage de cette famille. Elle est basée sur un algorithme de chiffrement largement utilisé, qui est le RC4. Cette fonction a aussi été cryptanalysée. Les fonctions de hachage basées sur les algorithmes de chiffrement par flux sont en général, plus rapides que celles basées sur les algorithmes de chiffrement par blocs.

- **Basées sur des problèmes mathématiques**: consiste à utiliser des fonctions issues des problèmes mathématiques difficiles, comme le problème de factorisation des grands nombres, logarithmique discret, etc. Pour construire des fonctions de compression, on peut utiliser d'autres problèmes aussi variés que la recherche du plus court vecteur dans un réseau, ou le problème de la résolution de systèmes d'équations quadratiques multi-variées dans un corps fini. Cependant, les principaux inconvénients de cette approche sont la nécessité de posséder une très grande quantité de mémoire ou encore une vitesse d'exécution souvent lente à cause d'opérations algébriques très coûteuses.

- **Basées sur les systèmes dynamiques** Les systèmes dynamiques constituent une bonne approche pour concevoir des fonctions de hachages sûres et rapides à cause de leurs caractéristiques :

  - Sensibilité aux conditions initiales : qui permet d'avoir un bon effet d'avalanche,
  - Comportement chaotique et imprévisible : permet de concevoir des fonctions pseudo-aléatoires avec de bonnes propriétés de sécurité,
  - La simplicité des opérations internes : ce qui implique la rapidité et la simplicité d'implémentation.

4.6 Applications des fonctions de hachage

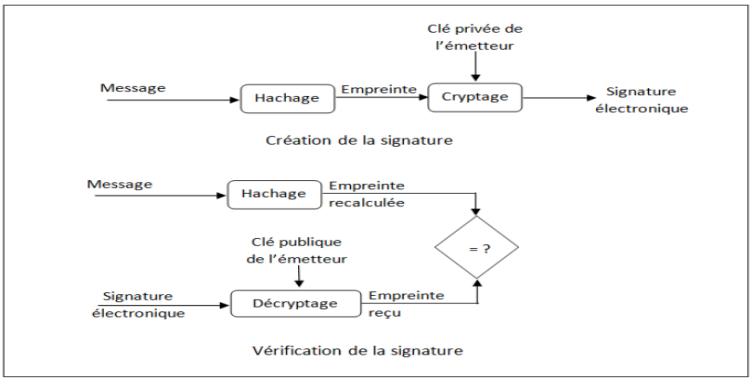
Les fonctions de hachage sont utilisées par de nombreux systèmes cryptographiques, à la fois en cryptographie symétrique et en cryptographie asymétrique. Nous présentons ci-dessous une liste non exhaustive de ces systèmes.

4.6.1 Code d'authentification de message

Les codes d'authentification de message (**MAC**) sont des empreintes d'une donnée qui dépendent d'une clé secrète **k**, et qui ne peuvent être calculées qu'en connaissant **k**. En ce sens, ils peuvent être vus comme des fonctions de hachage à clé secrète. Un MAC doit être difficile à forger c-à-dire qu'un attaquant ne doit pas pouvoir calculer de MAC sans connaître la clé. Une façon simple pour construire un MAC est de faire rentrer le message et la clé dans une fonction de hachage : $MAC_k(M) = F(k|M)$ elle est sûre si la fonction de hachage se comporte bien comme une fonction aléatoire, puisqu'on ne peut pas prévoir la valeur de la fonction en un point en connaissant seulement sa valeur en d'autres points.

4.6.2 Signature électronique (Hash-and-Sign)

Les schémas de signature sont sans doute l'application la plus importante des fonctions de hachage. Ils permettent à un utilisateur de signer un message à l'aide de **sa clé privée**. Chacun peut vérifier la validité de cette signature grâce à **la clé publique** correspondante. En pratique, au lieu d'appliquer un schéma de signature directement à un long message, on applique la signature à un haché du message. Ainsi, l'opération de signature est faite sur un identifiant de petite taille et elle sera moins coûteuse. Pour pouvoir signer des données de taille quelconque, la solution la plus répandue consiste donc à leur appliquer une fonction de hachage, puis d'appliquer la fonction de signature **S** sur l'empreinte obtenue (voir la Figure). La signature d'un message **M** est alors **S(H(M))**.



4.6.3 Génération de nombres pseudo-aléatoires

Les propriétés statistiques des fonctions de hachage et leur caractère à sens unique peuvent être exploités dans des contextes de génération de nombres aléatoires. En effet, de nombreux mécanismes cryptographiques nécessitent la génération de nombres aléatoires : génération de clés, signature électronique, chiffrement asymétrique, génération de valeurs d'initialisation pour le chiffrement symétrique...Les nombres ainsi générés doivent être tirés uniformément et être imprédictibles pour un attaquant.Les nombres aléatoires sont générés à partir de secrets stockés en mémoire et/ou de phénomènes physiques aléatoires. l'utilisation de fonctions de hachage permet d'obtenir des suites de bits indépendants et uniformément distribués. D'autre part, le caractère non inversible des fonctions de

hachage permet de dériver des nombres pseudo-aléatoires à partir de secrets sans les compromettre. Ces fonctions permettent également de mettre à jour les valeurs secrètes, afin que leur compromission éventuelle n'affecte pas leurs valeurs passées.

4.6.4 Stockage de mots de passe

Les fonctions de hachage sont utilisées pour éviter de stocker des mots de passe en clair. Ainsi, quand on se connecte sur un ordinateur, la machine calcule un haché du mot de passe, et le compare au haché préalablement connu. Ceci permet d'éviter de stocker le mot de passe en clair, et si la machine est compromise, l'attaquant ne pourra pas retrouver les mots de passe des utilisateurs (voir la Figure). Cette méthode semble sûre puisqu'une fonction de hachage est à sens unique. Ainsi, même si un attaquant mettait la main sur les hachés des mots de passe stockées sur la machine , il lui serait pratiquement impossible de retrouver le mot de passe d'accès

4.6.5 Protection des fichiers

Une des façons d'utiliser une fonction de hachage est de considérer l'empreinte d'un document comme un identifiant unique. En effet, une bonne fonction de hachage est résistante aux collisions, i.e. on ne peut pas trouver deux messages ayant la même empreinte. On peut utiliser ces identifiants pour vérifier l'intégrité d'un document, ou pour identifier un document si l'empreinte est mieux protégée que le document lui-même. Ceci sert notamment dans certains protocoles de téléchargement pair-à-pair : on obtient le haché du document depuis un serveur central, et le haché sert à vérifier les données reçues depuis les pairs.

4.6.5.1 Authentification par défi/réponse

Les MAC sont souvent utilisés pour construire des protocoles d'authentification. Dans un protocole d'authentification simple, un client veut s'identifier auprès d'un serveur avec qui il partage un mot de passe. Le serveur envoie un message aléatoire appelé défi ou challenge au client, et le client répond avec un MAC du défi, en utilisant le mot de passe comme clé. Cela ne révèle pas d'information utile sur la clé à un adversaire, mais le serveur peut vérifier que le calcul est correct et donc identifier l'utilisateur.

4.6.6 Famille de fonctions de hachage

4.6.6.1 Les fonctions "Messages Digest"

- **MD2**: C'est la première fonction de hachage de la famille MD a être implémenté. Elle a été conçue par Rivest en 1989, ensuite standardisée en 1992. Rogier et al. ont montré en 1997 comment générer des collisions contre la fonction de compression. Plus tard, des attaques en recherche de collisions et de pré-images sur la fonction de hachage complète ont été découvertes. Cette fonction n'est plus utilisée actuellement.

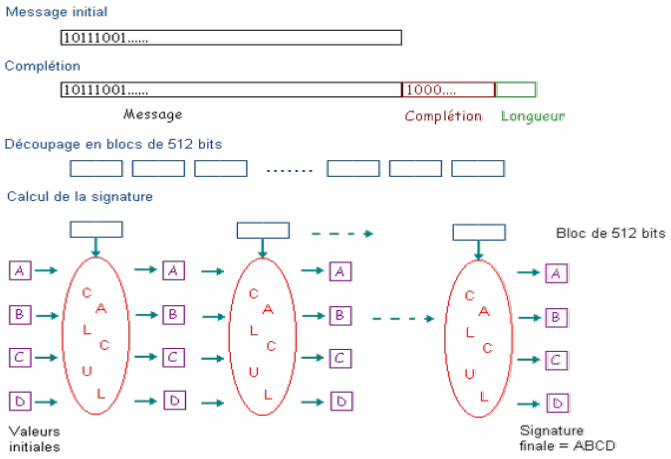
- **MD4**: Cette fonction peut être considérée comme l'origine de base de la famille MD-SHA. En effet, les principes de conception de sa fonction de compression sont largement repris par les fonctions ulterieurs. Cependant, cette fonction n'a été que peu utilisée, car elle a été remplacée par MD5. Elle a été proposée par Rivest en 1990 pour les laboratoires RSA.

- **MD5**: La fonction MD5 a été conçue par Rivest en 1991. Cette fonction a été très utilisée, et elle est toujours disponible aujourd'hui dans plusieurs systèmes et protocoles. Elle représente une amélioration de MD4. Donc la fonction de compression de MD5 possède un tour de plus, par rapport à MD4, de nouvelles fonctions booléennes et des constantes définies pour chaque étape. Les principes généraux restent fixes : des hachés de taille $n = 128$ bits pour un état interne de $r = 4$ registres de $w = 32$ bits chacun, initialisé avec la variable de chaînage d'entrée : les pricipales étapes de la fonction sont:

  - **Etape 1 : Complétion** Le message est constitué de **b** bits $m_1 ... m_b$. On complète le message par un **1**, et suffisamment de **0** pour que le message étendu ait une longueur congruente à 448, modulo 512. Puis on ajoute à ce message la valeur de **b**, codée en binaire sur **64** bits (on a donc b qui peut valoir jusque 264... ce qui est énorme). On obtient donc un message dont la longueur totale est un multiple de **512 bits**. On va travailler itérativement sur chacun des blocs de 512 bits.
  - **Etape 2 : Initialisation** On définit 4 buffers de 32 bits A,B,C et D, initialisés ainsi (les chiffres sont hexadécimaux, ie a=10, b=11...).
    * A=01234567
    * B=89abcdef
    * C=fedcba98
    * D=76543210

    On définit aussi 4 fonctions F,G,H et I, qui prennent des arguments codés sur 32 bits, et renvoie une valeur sur 32 bits, les opérations se faisant bit à bit.
    * F(X,Y,Z) = (X AND Y) OR (not(X) AND Z)
    * G(X,Y,Z) = (X AND Z) OR (Y AND not(Z))
    * H(X,Y,Z) = X xor Y xor Z
    * I(X,Y,Z) = Y xor (X OR not(Z))

    Ce qu'il y a d'important avec ces 4 fonctions et que si les bits de leurs arguments X,Y et Z sont indépendants, les bits du résultat le sont aussi.
  - **Etape 3 : Calcul itératif** Pour chaque bloc de 512 bits du texte, on fait les opérations suivantes:
    * on sauvegarde les valeurs des registres dans AA,BB,CC,DD.
    * on calcule de nouvelles valeurs pour A,B,C,D à partir de leurs anciennes valeurs, à partir des bits du bloc qu'on étudie, et à partir des 4 fonctions F,G,H,I.
    * on fait A=AA+A, B=BB+B, C=CC+C, D=DD+D.
  - **Etape 4: Ecriture du résumé** Le résumé sur 128 bits est obtenu en mettant bout à bout les 4 buffers A,B,C,D de 32 bits.



Description du fonctionnement du MD5

4.6.6.2 Les fonctions "Standard Hash Algorithm"

La famille des fonctions de hachage la plus connue est la SHA (Secure Hash Standard) qui est développée par la NSA et certifiée par le NIST. Les fonctions SHA256 et SHA-512 sont à ce jour, les plus sûres et les plus utilisées. Le dernier membre de cette famille est la fonction Keccak [BDPA09] qui a remporté la compétition de NIST en 2012, et est ainsi devenue la nouveau standard SHA-3 [Bou12].

- **SHA-0** est la première fonction de la famille " Secure Hash Standard ". Elle a été développée en 1993, avant d'être retirée peu après et remplacée par SHA-1. Inspirée des fonctions de la famille MD, la fonction de compression de SHA-0 n'en diffère de celle de MD5, que par l'utilisation d'une nouvelle expansion de message. SHA-0 permet de calculer des empreintes de 160 bits, ce qui permet de résister mieux aux attaques génériques.

- **SHA-1** a été publiée en 1995 pour remplacer la fonction SHA0. Malgré ses vulnérabilités connues qui conduisent à son remplacement progressif par SHA-2, elle reste probablement aujourd'hui la fonction la plus utilisée dans la pratique. Sa fonction de compression permet le traitement de blocs de message de 512 bits et de variables de chaînage de 160 bits.

- **SHA-256** Publiée en 2002, SHA-256 fait partie des derniers membres de la famille SHA. Outre sa taille du haché, elle contient plusieurs nouveautés par rapport à ses prédécesseurs. Par exemple, l'expansion de message est beaucoup plus complexe et corrige les précédentes erreurs de SHA-0 et SHA-1. SHA-256 produit des hachés de n = 256 bits, mais il existe aussi une version 224 bits introduite 2 ans plus tard.

- **SHA-512** a été publiée en même temps que SHA-256 et représente son équivalent pour les processeurs 64 bits, qui vont progressivement remplacer ceux de 32 bits dans les ordinateurs. Les mots traités seront donc de taille 64 bits pour profiter pleinement de cette nouvelle architecture.

  Les autres différences par rapport à SHA-256 concernent la taille de sortie, qui est doublée pour obtenir une fonction fiable sur le long terme, et le nombre d'étapes qui est augmenté. Ainsi, SHA-512 produit des hachés de n = 512 bits, mais une version 384 bits a été aussi introduite en même temps. Comme pour SHA-256, deux registres sont mis à jour durant une étape de SHA-512, et nous notons Ai et Bi ces registres cibles.

4.6.6.3 Les fonctions RIPEMD

- RIPEMD-0 RIPEMD-0 est l'une des primitives recommandées en 1992 à l'issue d'une étude d'un consortium dans le cadre du projet européen RACE Integrity Primitives Evaluation (RIPE) sur les primitives permettant de garantir l'intégrité. Originalement nommée RIPEMD, la fonction de compression se compose de deux branches parallèles, chacune quasiment identique à la fonction de compression de MD4. Les deux lignes parallèles de calcul ne diffèrent que par l'emploi de constantes différentes. Les paramètres de chaque branche sont donc égaux à ceux de MD4, mais l'ordre d'introduction des mots du bloc de message étendu et les longueurs de rotation lors des étapes sont différents de ceux de MD4. Les hachés sont de taille n = 128 bits pour un état interne de 4 registres de 32 bits chacun pour chaque branche.

- RIPEMD-128 En 1996, Hans Dobbertin, Antoon Bosselaers et Bart Preneel ont proposé une version améliorée de RIPEMD-0 pour résister aux premières cryptanalyses de MD4 et de RIPEMD-0. De plus, une version 256 bits a aussi été définie, mais pour une sécurité équivalante à une fonction de 128 bits.

4.6.7 Conclusion

Une fonction de hachage est une composante fondamentale dans un système de sécurité de l'information, et joue un rôle important dans plusieurs applications cryptographiques. Ainsi, elle permet la vérification de l'intégrité des données, l'authentification symétrique de source des messages (MAC), et la validation des signatures numériques.