8 – Fonctions de hachage cryptographiques

Des usages variés

- en plus du secret, de nos jours, la cryptographie doit assurer l'authentification des messages
- elle doit également en assurer l'intégrité ou même la non-répudiation
- les fonctions de hachage (*hash-function* en anglais) remplissent bien ces rôles en produisant des empreintes de messages
- le calcul d'une empreinte est finalement le moyen le plus simple de détecter la présence d'erreurs de transmission ou de corruption
- elles sont utilisées dans les protocoles de signatures électroniques quand les données à signer sont trop volumineuses pour l'être par d'autres procédés
- certains mots de passe jetables sont une application directe des fonctions de hachage
- finalement, les fonctions de hachage constituent un mécanisme de base de nombreux protocoles cryptographiques

Fonction de hachage

- l'empreinte calculée par une fonction de hachage doit être évoluée, dans le sens où elle est censée identifier le message
- on fait l'analogie avec les empreintes digitales, qui à défaut de servir à reconstituer les caractéristiques de quelqu'un, permette de l'identifier
- d'un point de vue technique, les fonctions de hachage, en tant que codes non injectifs, sont des codes ambigus, compresseurs avec perte (*cf.* Cours 2)
- fonction de hachage :

$$H : \{0,1\}^* \to \{0,1\}^n$$

$$x \mapsto y = H(x)$$

- une fonction de hachage est donc une fonction qui transforme une chaîne de taille quelconque en une chaîne de taille fixe : son empreinte
- $H(x)$ doit être très rapide à calculer à partir de x

Exemples déjà vus
- le bit de parité : si $m = \sigma_1 ... \sigma_n$ alors il vaut $H(m) = (\sum_{i=1}^{n} \sigma_i) \mod 2$
- lors du stockage des mots de passe Unix, les 25 applications de Des constituent un calcul de fonction de hachage

Propriétés par rapport aux collisions

- on parle de collisions entre 2 chaînes x et x' lorsque :

$$\begin{cases} x \neq x' \\ H(x) = H(x') \end{cases}$$

- vu les ensembles de départ et d'arrivée, les collisions sont inévitables
- si y est tel que $y = H(x)$, x est appelé antécédent ou préimage de y
- la qualité de H dépend des 3 propriétés suivantes :
  1. résistance à la préimage : étant donné y, on ne peut pas trouver en un temps raisonnable de x tel que $y = H(x)$
  2. résistance à la seconde préimage : étant donné x, on ne peut pas trouver en un temps raisonnable de $x' \neq x$ tel que $H(x') = H(x)$
  3. résistance aux collisions : on ne peut pas trouver en un temps raisonnable de couples x et x' tels que $H(x) = H(x')$
- une fonction de hachage à sens unique est une fonction de hachage qui vérifie les propriétés 1. et 2.
- une fonction de hachage sera considérée comme cassée lorsqu'il existera un algorithme permettant de trouver des collisions avec une complexité meilleure que l'attaque de Yuval (en $\mathcal{O}(2^{n/2})$ calculs d'empreintes successifs où n est la taille des empreintes)

Paradoxe des anniversaires

- Donnée : $B = (b_1, \ldots, b_k) \in \{1, 2, \ldots, n\}^k$

  Problème : quelle est la probabilité p qu'il existe au moins 2 éléments identiques dans B ?
- par passage à l'événement complémentaire, on a :

$$q = 1 - p = 1\left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right)\ldots\left(1 - \frac{k-1}{n}\right) = \prod_{i=1}^{k-1}\left(1 - \frac{i}{n}\right)$$

- supposons que sur $[0,1]$, $1 - x \approx e^{-x}$:

$$q \approx \prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{-\frac{1}{n}\sum_{i=1}^{k-1} i} = e^{-\frac{(k-1)k}{2n}}$$

  par passage au logarithme :

$$\ln\frac{1}{q} \approx \frac{(k-1)k}{2n} \iff k^2 \approx 2n \ln\frac{1}{q}$$

- si $p > \frac{1}{2}$ alors $k \geq \sqrt{2n\ln 2}$ et donc $k = \mathcal{O}(\sqrt{n})$ car $\sqrt{2\ln 2} = 1,177$
- utilité : dimensionner la longueur n de l'empreinte d'une fonction de hachage pour éviter les collisions

Exemple

$n = 365$, quelle est la valeur de seuil de k pour avoir $p > 1/2$? (*cf.* TD n° 6)

Attaque de Yuval (dite « Attaque des anniversaires »)

- l'idée est de calculer et trier des couples $(x, H(x))$ pour détecter des collisions
- combien faut-il calculer de couples pour trouver 2 empreintes identiques parmi 2^n empreintes possibles avec une probabilité $p > 1/2$?

  (*hypothèse supplémentaire* : les images par H suivent une distribution uniforme)
- en considérant k entrées, on a plus d'1 chance sur 2 d'avoir une collision avec k en $O(\sqrt{2^n})$, soit, en passant au logarithme :

n	50	100	150	200
$\log_2 k$	25	50	75	100

- en effet, en calculant plus de $2^{n/2}$ empreintes, on a une collision avec une probabilité strictement supérieure à $1/2$
- comme on demande que H soit résistante aux collisions, on choisit n pour que le calcul de $2^{n/2}$ images par H soit irréaliste : à ce jour, $n \geq 128$ voire même $n \geq 160$.

Utilisation des fonctions de hachage

Intégrité des données

- on calcule l'empreinte d'une donnée
- on transmet la donnée et son empreinte
- à réception, on recalcule l'empreinte de la donnée que l'on compare à l'empreinte reçue
- cette technique est utilisée contre les virus informatiques ou dans la distribution de logiciels
- on nomme cette utilisation MDC (*Modification Detection Code* ou *Manipulation Detection Code*)

Authentification des données

- si Alice et Bob partagent une clef secrète et une fonction de hachage H, ils peuvent se servir de leur clef comme valeur initiale de H
- si Alice envoie à Bob un message m et son empreinte $H(m)$, à réception, Bob peut vérifier que le message provient bien d'Alice
- les MAC (*Message Authentification Code*) gèrent ainsi l'intégrité des données et l'authentification de leur source

Construction de Merkle-Damgård

- c'est une des constructions de fonctions de hachage les plus répandues
- elle utilise la fonction de compression h suivante :

$$h : \{0,1\}^b \times \{0,1\}^n \longrightarrow \{0,1\}^n$$

- le message M est découpé en blocs de b bits $M_1, \ldots, M_k$

  (il y a bourrage - *padding* - au besoin)
- à partir de IV (*Initial Value*) de n bits, on itère h :

$$H(M) = h(M_k, h(\ldots h(M_2, h(M_1, \mathrm{IV}))))$$

- $H(M)$ est de taille n, IV est une chaîne de taille n fixée par l'implantation ou par l'algorithme (*cf.* TP 8)

Théorème : si h est résistante aux collisions, alors H l'est aussi

- une autre construction des fonctions h_i pourrait être la suivante :

$$h_i = E_{g(h_{i-1})}(M_i) \oplus h_{i-1} \oplus M_i$$

  avec E_k une fonction de chiffrement à clef secrète et g une fonction d'adaptation à la taille de clef

MD5

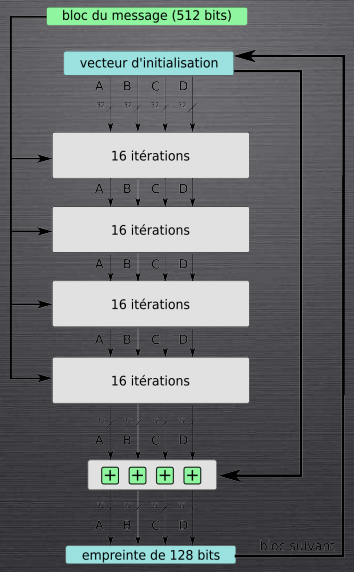
- MD5 (*Message Digest 5*) a été introduit en 1991 par R. Rivest
- c'est une amélioration de MD4 datant de 1990, *cassé* puis abandonné
- en 1996, une faille est trouvée dans la mesure où l'on peut construire systématiquement 2 antécédents ayant la même empreinte
- considéré dès lors comme non sûr, il a été remplacé par SHA-1
- la taille de l'empreinte MD5 est de 128 bits donc la recherche de collisions par force brute est de 2^{64} calculs d'empreinte ...
  mais on y arrive de nos jours en 2^{42} voire 2^{30} calculs
- en 2004, il a été définitivement *cassé* par des chercheurs chinois sans recours à la recherche exhaustive
- toujours utilisé pour vérifier l'intégrité d'un logiciel à télécharger (`HashCalc`, `md5sum`)
- avec un sel, toujours utilisé pour coder des mots de passe (GNU-LINUX, CISCO)

MD5 : L'ALGORITHME

- la fonction de compression de MD5 utilise des blocs de 512 bits pour produire une empreinte de 128 bits
- chaque bloc est découpé en 16 sous-blocs de 32 bits $(M_i)_{0 \leq i \leq 15}$
- MD5 utilise 64 constantes fixées $(K_j)_{0 \leq j \leq 63}$
- l'algorithme travaille avec un état interne sur 128 bits lui-même divisé en 4 mots de 32 bits : A, B, C et D (initialisés avec des constantes)
- 64 *tours* sont effectués, répartis en 4 rondes de 16 opérations
- l'algorithme incorpore les blocs provenant du message à hacher, ces blocs vont ainsi modifier l'état interne
- les 16 opérations similaires sont basées sur une fonction F qui varie selon la ronde, une addition $\mod 2^{32}$ et une rotation vers la gauche
- les 4 fonctions (certaines non-linéaires) disponibles sont :

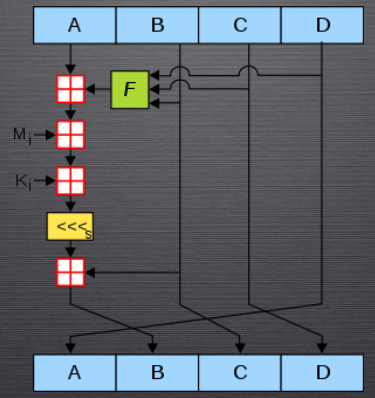
$$F = (B \wedge C) \vee (\neg B \wedge D)$$
$$F = (D \wedge B) \vee (\neg D \wedge C)$$
$$F = B \oplus C \oplus D$$
$$F = C \oplus (B \vee \neg D)$$

MD5 : SCHÉMA GÉNÉRAL



(figure Wikipédia)

MD5 : UNE OPÉRATION



(figure Wikipédia)

SHA-1 et SHA-256

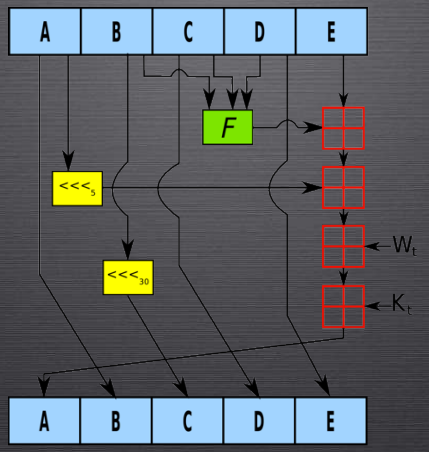
- SHA-1 (*Secure Hash Algorithm*) a été publié en 1995 par le NIST
- *a posteriori*, il s'est avéré être un bon générateur de nombre pseudo-aléatoires ...
- sa fonction de compression utilise des blocs de 512 bits pour produire un empreinte de 160 bits
- la recherche de collisions par force brute a une complexité en $\mathcal{O}(2^{80})$
- des collisions ont été obtenues en $\mathcal{O}(2^{63})$ mais l'attaque reste difficile
- SHA-1 est considéré comme théoriquement *cassé*, mais du fait de sa popularité et de la difficulté de l'attaque, il est encore largement répandu
- SHA-256, publié en 2000, est une variante plus robuste, avec une taille d'empreinte de 256 bits
- il a inspiré le chiffrement à clef secrète SHACAL-2, sans attaque efficace connue à ce jour

SHA-1

- sa fonction de compression utilise des blocs de 512 bits pour produire une empreinte de 160 bits
- comme pour MD5, chaque bloc B est découpé en 16 sous-blocs de 32 bits
- ils sont étendus en 80 nouveaux blocs $(W_j)_{0\leq j\leq 79}$
- 4 constantes $(K_j)_{0\leq j\leq 3}$ sont utilisées
- l'algoritme travaille sur un état de 160 bits subdivisés en 5 mots A, B, C, D et E de 32 bits chacun (initialisés avec des constantes)
- 80 tours sont effectués
- SHA-1 utilise une succession de fonctions logiques $f_0, f_1, \ldots, f_{79}$
- chaque fonction f_t où $0 \leq t \leq 79$ travaille sur 3 mots de 32 bits x, y et z, et génère un mot de 32 bits en sortie :

$$f_t(x,y,z) = \begin{cases} \text{Ch}(x,y,z) = (x \wedge y) \vee (\neg x \wedge z) & \text{si } 0 \leq t < 20 \\ \text{Parity}(x,y,z) = x \oplus y \oplus z & \text{si } 20 \leq t < 40 \\ \text{Maj}(x,y,z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & \text{si } 40 \leq t < 60 \\ \text{Parity}(x,y,z) = x \oplus y \oplus z & \text{si } 60 \leq t < 80 \end{cases}$$

SHA-1 : une opération



(figure Wikipédia)

Whirlpool

- A l'heure actuelle, on préconise SHA-256, RIPEMD-160 (européen) ou Whirlpool
- Whirlpool (du nom de la galaxie $M-51$) est une fonction de hachage recommandée par le projet NESSIE en 2004
- elle est basée sur la 2e construction donnée
- le chiffrement sous-jacent est W, une variante de Rijndael
- la fonction de compression comporte 10 tours et utilise des blocs de 512 bits pour produire une empreinte de 512 bits

SHA-3 alias Keccak

- Keccak est une fonction de hachage inventée par G. Bertoni, J. Daemen, M. Peeters, et G. Van Assche
- proposé à la compétition SHA-3, Keccak a été choisi par le NIST en octobre 2012
- ce n'est pas forcément pour remplacer SHA-2 sans attaque significative ni SHA-1, le but était de faire émerger une fonction de hachage différente dans sa conception
- SHA-3 utilise une construction *en éponge* totalement nouvelle !
- la taille des empreintes y est variable, de 224 à 512 bits.

Problèmes liés à l'identification

- lors d'un accès distant à un système sécurisé, l'identification est nécessaire
- l'utilisateur s'identifie habituellement au moyen du couple *identifiant/mot de passe*

- un pirate peut découvrir le mot de passe de l'utilisateur de 3 manières différentes :
  1. par accès direct aux informations contenues au sein du système *e.g* au fichier des mots de passe
  2. par espionnage de la ligne de connexion au moment où l'utilisateur se connecte ou par un programme d'espionnage sur l'ordinateur de l'utilisateur
  3. par le fait de retrouver le mot de passe de l'utilisateur si ce dernier en a choisi un qu'il est facile de deviner

Mots de passe jetables

- la solution la plus courante consiste à conserver les mots de passe chiffrés
- à terme, ce sont les systèmes biométriques qui risquent de l'emporter (empreintes vocales, digitales ou rétiniennes, reconnaissance faciale ...)

- mais une autre solution est de faire varier le mot de passe à chaque connexion : c'est l'idée des mots de passe jetables (*One Time Passwords*)
- ils sont utilisés dans `S/Key`, `Opie` ou encore `LogDaemon`

- le service de contrôle d'accès se fait au moyen de fonctions à sens unique en général ou de hachage cryptographique en particulier
- cette application des fonctions de hachage a été inventée par L. Lamport

OTP : fonctionnement

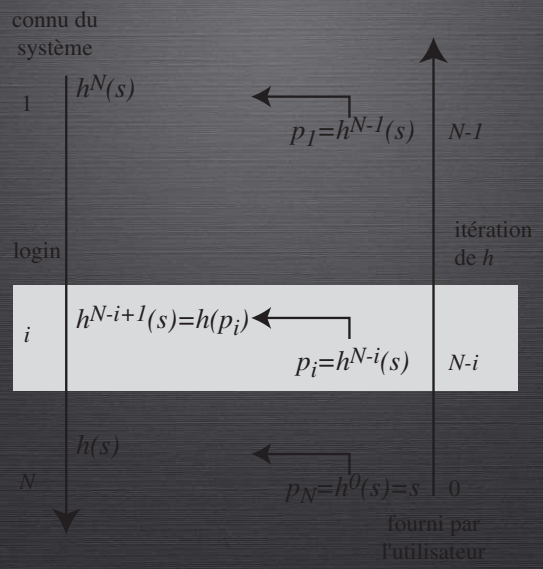
- le système et l'utilisateur partage une fonction de hachage cryptographique h et un nombre maximal N de connexions
- l'utilisateur a un secret initial s et fournit $h^N(s)$ au système
- le i^e mot de passe p_i est égal à $h^{N-i}(s)$
- ainsi, la suite des N mots de passe que l'utilisateur fournit pour se connecter est :

$$h^{N-1}(s),\ h^{N-2}(s),\ \ldots,\ h(h(h(s))),\ h(h(s)),\ h(s),\ s$$

- côté système, l'identification se fera précisément sur les valeurs :

$$h^N(s), h^{N-1}(s), \ldots, h(h(h(s))), h(h(s)), h(s)$$

OTP : SCHÉMA



OTP : FONCTIONNEMENT (SUITE)

- pour la i^e connexion d'une série de N connexions, le système connaît $h^{N-i+1}(s)$ et le numéro i de la connexion
- à la demande de connexion, le système envoie à l'utilisateur un défi sous la forme du numéro i de connexion
- l'utilisateur dispose de la fonction h, du secret initial s, du nombre maximal de connexions N et du numéro i de la connexion courante
  - il peut alors calculer son mot de passe : $p_i = h^{N-i}(s)$ qu'il transmet au système
- le système, à réception de p_i calcule $h(p_i) = h(h^{N-i}(s)) = h^{N-i+1}(s)$ qu'il connaît de la connexion précédente
- s'il y a égalité, il accepte la connexion et mémorise la valeur p_i reçue de l'utilisateur pour la prochaine connexion

autrement dit, chacun des mots de passe est la valeur requise par le système pour la connexion suivante

A suivre ...