Introduction à la cryptographie
Fonctions de hachage

1 Définition

Lorsque vous téléchargez des fichiers volumineux sur internet, il existe un moyen, certes pas infaillible, de vérifier qu'ils n'ont pas été altérés lors du transfert. Cela consiste à appliquer une fonction h au contenu initial c, d'envoyer $h(c)$ à l'internaute qui vérifiera de son côté que le fichier téléchargé c' possède la même empreinte $h(c')$. Si $h(c)$ est différent de $h(c')$, alors les données échangées ont, à priori, subi des altérations lors de la réception. A l'inverse, si $h(c) = h(c')$, il est très probable que c' soit intact. Évidemment, pour que le concept soit intéressant, il faut que $h(c)$ soit petit, en terme d'espace mémoire, vis à vis de c. Par exemple, pour un fichier de quelques méga octets, on veut une empreinte de quelques centaines de bits.
Ce type de fonction est appelé fonction de hachage. Formellement, on pose h comme une fonction de $\{0,1\}^*$ à $\{0,1\}^N$ où $\{0,1\}^*$ désigne une chaine de longueur quelconque et $\{0,1\}^N$ une chaine de longueur N.

À travers cette définition, il est évident que certains messages auront la même empreinte, on parlera alors de collision. Un type de fonction de hachage pourrait être, étant donnée une chaine, d'en donner la taille. Dès lors, un fichier téléchargé qui n'aurait pas la même taille que le contenu initial serait à priori corrompu. Il existe différentes méthodes, utilisées de nos jours, comme les checksums (sommes de contrôles) ou les codes CRC (Contrôle de redondance cyclique).

Les fonctions de hachages ont vocation a être utilisées en cryptographie. Sur internet, les mots de passe ne sont pas stockés en clair dans une base de données, on utilise plutôt leurs "hachés". Ainsi, si une personne malveillante parvient à "dérober" cette base de donnée, les informations qu'elle pourra en tirer sont inutiles... A condition que h vérifie les propriétés suivantes :

- Impossibilité pratique de trouver une pré-image, c'est à dire, trouver M en connaissant $h(M)$.
- Impossibilité pratique de trouver une seconde pré-image, c'est à dire, connaissant $h(M)$ ou même M, de trouver M' tel que $h(M') = h(M)$.
- Impossibilité pratique de trouver deux messages M et M' tel que $h(M) = h(M')$ (résistance aux collisions).

2 Exemples intuitifs

Comme vous le savez, sur un ordinateur, chaque caractère est stocké sur 8 bits, au format ASCII. À chaque caractère, correspond un nombre compris entre 0 et 255. Par exemple, "Bonjour" est encodé comme "66 111 110 106 111 117 114". Une fonction de hachage simple pourrait être une somme de ces valeurs, modulo un nombre donné,

$$66 + 111 + 110 + 106 + 111 + 117 + 114 \bmod 571 \equiv 735 \bmod 571 \equiv 164$$

Coder une telle fonction, prenant en entrée un message d'une taille quelconque et la valeur du modulo. Expliquez pourquoi une telle fonction ne résiste pas aux collisions.

On va améliorer la fonction précédente, en accordant une importance à l'ordre des lettres. L'empreinte de "Bonjour" sera calculée de la façon suivante :

$$66+111*\mathbf{2}+110*\mathbf{4}+106*\mathbf{8}+111*\mathbf{16}+117*\mathbf{32}+114*\mathbf{64} \bmod 571 = 117 \bmod 571$$

Il est un peu plus compliqué de trouver une collision avec cette version améliorée de notre fonction de hachage. Cela dit, en se donnant 30 messages différents, la probabilité que deux aient la même empreinte n'est pas si faible. Prenons le problème à l'envers, et cherchons la probabilité que 30 messages aient tous une empreinte différente. L'empreinte du deuxième message à une probabilité de $\frac{570}{571}$ d'être différente de celle du premier. L'empreinte du troisième message à une probabilité de $\frac{569}{571}$ d'être différente de celles des deux premiers, ainsi de suite... La probabilité que TOUS les messages aient une empreinte différente est donc de $\frac{570}{571} \times \frac{569}{571} \times \cdots \times \frac{542}{571} \approx 0,46$. La probabilité que deux messages aient donc la même empreinte est de fait $1 - 0,46 = 0,54$ soit plus d'une chance sur deux !
Il suffit de générer un corpus $\mathcal{C}$ de 30 messages aléatoires (différents) et de tester pour tous les couples (M, M') (avec $(M \neq M')$) si $h(M) = h(M')$. On a plus d'une chance sur deux pour que cela arrive. Il y a en tout $(30 \times 29)/2 = 435$ couples différents, ce qui est largement de l'ordre du raisonnable pour une recherche exhaustive.
Il apparait alors évident que N (la taille de l'espace d'arrivée de la fonction h) doit être suffisamment conséquent pour éviter ce genre de faiblesse.
Coder une telle "attaque".

3 Une vraie fonction : le MD-2

Le **MD-2** [1] est une fonction de hachage cryptographique, proposée en 1989 par l'un des pères fondateurs de **RSA**, Ronald Rivest. Aujourd'hui,

1. Message Digest

nous en sommes à la version six, le MD-6, mais c'est sa version précédente (MD-6) qui est à l'heure actuelle la plus utilisée. Nous allons implanter une version simplifiée de MD-2.

3.1 Padding

La taille, en octet, du message M doit être un multiple de 16. Il faut donc rajouter des octets au message M. S'il faut rajouter deux octets, on accole deux fois l'octet 00000010 au message. S'il faut rajouter trois octets, on ajoute trois fois l'octet 0000 0011 à M, etc.

3.2 Ajout de la somme de contrôle

Seize octets supplémentaires de somme de contrôle sont normalement ajoutés à M'. On ne le fera pas ici.

3.3 Traitement des blocs

On initialise un buffer X de 48 octets à 0. S est normalement une permutation générée à partir des décimales de π, mais on prendra ici la fonction "identité", $S[t] = t$.

```
/* Traiter chaque bloc de 16 mots. */
Pour i = 0 à N'/16-1 faire
/* Copier le bloc i dans X. */
Pour j = 0 à 15 faire
Régler X[16+j] à M[i*16+j].
Régler X[32+j] à (X[16+j] xor X[j]).
fin /* de la boucle sur j */
Régler t à 0.
/* Faire 18 tours. */
Pour j = 0 à 17 faire
/* Tour j. */
Pour k = 0 à 47 faire
Régler t et X[k] à (X[k] xor S[t]).
fin /* de la boucle sur k */
Régler t à (t+j) modulo 256.
fin /* de la boucle sur j */
fin /* de la boucle sur i */
Retourner X[0..15]
```

On vérifiera la validité du code avec le haché de "Bonjour", qui est, sous forme de code *ASCII*, "48 104 0 32 40 24 0 100 28 80 52 14 10 41 56 64".