ÉPREUVE COMMUNE DE TIPE 2008 - Partie D

TITRE :

Les Fonctions de Hachage

Temps de préparation : ……..……2 h 15 minutes
Temps de présentation devant le jury : …….10 minutes
Entretien avec le jury : ……..10 minutes

GUIDE POUR LE CANDIDAT :

Le dossier ci-joint comporte au total : 17 pages

Document principal (13 pages, dont celle-ci)
Documents complémentaires (4 pages) : annexe et glossaire

Travail **suggéré** au candidat :

Le candidat peut, au choix, proposer une synthèse du texte, ou mettre en avant un algorithme, une démonstration. Dans tous les cas il veillera à bien comprendre ce qu'il expose et cherchera à apporter un éclairage personnel au texte.

Attention : s*i le candidat préfère effectuer un autre travail sur le dossier, il lui est* ***expressément recommandé*** *d'en informer le jury avant de commencer l'exposé.*

CONSEILS GENERAUX POUR LA PREPARATION DE L'EPREUVE :

* Lisez le dossier en entier dans un temps raisonnable.
* Réservez du temps pour préparer l'exposé devant le jury.

- Vous pouvez écrire sur le présent dossier, le surligner, le découper … mais tout sera à remettre au jury en fin d'oral.
- En fin de préparation, rassemblez et ordonnez soigneusement TOUS les documents (transparents, *etc.*) dont vous comptez vous servir pendant l'oral, ainsi que le dossier, les transparents et les brouillons utilisés pendant la préparation. En entrant dans la salle d'oral, vous devez être prêts à débuter votre exposé.
- A la fin de l'oral, vous devez remettre au jury le présent dossier, les transparents et les brouillons utilisés pour cette partie de l'oral, ainsi que TOUS les transparents et autres documents présentés pendant votre prestation.

I Introduction

Une fonction de hachage est une méthode permettant de caractériser une information, une donnée. En faisant subir une suite de traitements reproductibles à une entrée, elle génère une empreinte servant à identifier la donnée initiale. De telles fonctions datent de la fin des années 1980 (algorithme MD2) mais l'idée est plus ancienne, et a germé dès l'apparition des codes correcteurs d'erreurs (théorie de l'information).

Une fonction de hachage prend donc en entrée un message de taille quelconque, applique une série de transformations et réduit ces données. On obtient à la sortie une chaîne de caractères hexadécimaux, le condensé, qui résume en quelque sorte le fichier. Cette sortie a une taille fixe qui varie selon les algorithmes (128 bits pour MD5 et 160 bits pour SHA-1).

Ces fonctions sont très utilisées en informatique et en cryptographie.On les rencontre en navigant sur le Web : les auteurs de logiciels proposent souvent des empreintes sur les pages dédiées aux téléchargements (des fichiers portant l'extension *md5* ou *sha1*, qui contiennent la valeur hachée dudit programme). En comparant l'empreinte de la version téléchargée avec l'empreinte disponible sur le site, l'utilisateur peut s'assurer que sa version n'a pas été corrompue (erreurs de transmission, virus, etc.)

Enfin, une fonction de hachage *cryptographique* doit aussi satisfaire d'autres contraintes. La première stipule qu'il doit être très difficile de retrouver ou générer un texte à partir de l'empreinte (on parle alors de fonction à sens unique). Par très difficile, on entend que même avec une armée de machines dédiées à cette tâche, il sera impossible d'effectuer une telle extraction en un temps raisonnable.

Une autre caractéristique d'une fonction de hachage cryptographique concerne le comportement de l'empreinte selon le fichier en entrée. La moindre modification dans ce fichier doit engendrer un condensé totalement différent. En outre, le hachage doit rendre impossible la création d'un fichier qui donne la même empreinte qu'un autre préalablement fixé. Cette dernière contrainte est cruciale pour assurer l'intégrité d'un fichier : si elle n'était pas respectée, on pourrait facilement générer un fichier corrompu mais valide aux yeux de l'utilisateur.

Ce texte a pour but de formaliser ces idées, d'expliquer comment sont construites de telles fonctions, et de donner des exemples concrets d'utilisation de ces dernières.

II Fonctions de hachage et de compression

1 Définitions

DÉFINITION 1. *Soit Σ un alphabet. On définit une* fonction de hachage *comme une application*

$$h : \Sigma^* \longrightarrow \Sigma^n, n \in \mathbb{N}$$

Les fonctions de hachage associent donc, à des chaînes de caractères de longueur quelconque, d'autres chaînes de longueur fixe. Remarquons dès à présent qu'elles ne peuvent être injectives.

EXEMPLE 1. Par exemple, l'application associant[1] $b_1 \oplus b_2 \oplus \ldots \oplus b_n$ au mot (binaire) $b_1 b_2 \ldots b_n$ dans $\{0;1\}^*$ est une fonction de hachage. Elle envoie le mot 01101 en 1.

Plus généralement, elle transforme une chaîne de caractères b en 1 si le nombre de 1 dans b est impair (et sinon en 0).

Une des manières de fabriquer des fonctions de hachage est d'utiliser des *fonctions de compression*.

DÉFINITION 2. *Par définition, une fonction de compression est une application*

$$h : \Sigma^m \longrightarrow \Sigma^n, m, n \in \mathbb{N}, m > n$$

En d'autres termes, les fonctions de compression remplacent une chaîne de caractères de longueur fixée par une chaîne de caractères plus courte, elle aussi de longueur fixée.

EXEMPLE 2. Un exemple d'application de compression est l'application qui envoie le mot $b_1 b_2 \ldots b_m \in \{0,1\}^m$ en $b_1 \oplus b_2 \oplus \ldots \oplus b_m$, quand $m > 1$.

Les fonctions de hachage et de compression peuvent être utilisées dans de nombreux contextes, tel que la construction de dictionnaires. Mais ce qui nous intéresse ici est leur rôle, important, en cryptographie.

Quand de telles fonctions sont utilisées en cryptographie, il faut s'assurer qu'un certain nombre de propriétés, garantissant leur sécurité, soient satisfaites. Décrivons ces propriétés...

[1] $\oplus$ désigne le *ou exclusif booléen*

2 Les propriétés exigées

h représente, dans ce qui suit, soit une fonction de hachage ($h : \Sigma^* \longrightarrow \Sigma^n$), soit une fonction de compression ($h : \Sigma^m \longrightarrow \Sigma^n$). Nous noterons D l'ensemble de départ de h.

Remarquons pour commencer que si h doit être utilisée en cryptographie, alors l'élément $h(x)$ devra être facile à calculer, pour tout x de D. Supposons que tel est le cas.

DÉFINITION 3. *h est dite* fonction à sens unique *s'il est infaisable de déterminer son inverse (autrement dit, si le calcul de x tel que $h(x) = s$, à partir de s, est impossible).*$\diamondsuit$

Il reste cependant à définir ce que signifie *infaisable* dans notre contexte. Cette idée est compliquée à mettre en forme : pour bien faire, il faudrait user des concepts de la théorie de la complexité, qui sont difficiles. Nous n'en donnerons ici qu'une idée intuitive.

DÉFINITION 4. *Nous dirons qu'une fonction h est* à sens unique *lorsque les programmes qui essaient, à partir d'une donnée s, de calculer x tel que $h(x) = s$, échouent* presque toujours*, par manque de temps, ou d'espace mémoire.* $\diamondsuit$

REMARQUE 1. En toute rigueur, on ne connaît aucune fonction à sens unique et on ne sait même pas s'il en existe. Cependant, il existe des fonctions faciles à évaluer, mais dont aucun algorithme efficace d'inversion n'est connu. À défaut de mieux, on les utilise comme fonction à sens unique.

EXEMPLE 3. Soit p un nombre premier de 1024 bits (choisi au hasard), et g une racine primitive mod p. Alors la fonction

$$\{0, 1, 2, \ldots, p-1\} \longrightarrow \{1, 2, \ldots, p-1\}$$

définie par

$$x \longmapsto g^x \bmod p$$

est facile à calculer par exponentiation rapide, mais on ne sait pas déterminer son inverse efficacement[2]. Cette fonction peut donc servir de fonction à sens unique.

---

[2] Il est très difficile de calculer le logarithme discret x en base g de $X = g^x \bmod p$.

3 Les collisions

DÉFINITION 5. *On appelle* collision *de h tout couple (x, x') de D^2 tel que $x \neq x'$ et $h(x) = h(x')$.* ♢

EXEMPLE 4. Une collision de l'exemple 1 est (111,100), ou tout couple formé de deux chaînes de caractères distinctes, ayant toutes les deux, ou n'ayant ni l'une ni l'autre, un nombre pair de 1.

---

4 Fonctions faiblement résistantes aux collisions

DÉFINITION 6. *h est dite* faiblement résistante aux collisions *si pour tout $x \in D$, il est infaisable (en pratique, donc) de trouver x' tel que (x, x') soit une collision de h.*♢

Il existe des situations où il est nécessaire qu'une fonction soit faiblement résistante aux collisions, comme dans l'exemple suivant...

EXEMPLE 5. Supposons qu'Alice souhaite protéger un algorithme de chiffrement x enregistré sur son disque dur.

A l'aide d'une fonction de hachage $h : \Sigma^* \longrightarrow \Sigma^n$, elle calcule la valeur hachée $y = h(x)$, qu'elle stocke sur sa clé USB.

Plus tard, Alice retourne sur son ordinateur et, avant d'utiliser son programme de chiffrement, elle regarde s'il n'a pas été changé. Pour cela, Alice vérifie que la valeur hachée actuelle du programme n'a pas changé.

En toute rigueur, ce test n'est valable qu'à partir du moment où la fonction de hachage h est bien faiblement résistante aux collisions. Sans quoi, un adversaire pourrait remplacer le programme x par un autre ayant la même valeur hachée.

Cet exemple illustre une utilisation habituelle des fonctions de hachage résistantes aux collisions : l'intégrité d'un document donné est réduite à celle de son résumé (chaîne de caractères beaucoup plus petite).

5 Résistance forte aux collisions

DÉFINITION 7. *On dit que la fonction de hachage h est* fortement résistante aux collisions *si le calcul d'une quelconque collision (x, x') de h est infaisable.* ♢

Il existe des situations pour lesquelles l'utilisation de fonctions de hachage fortement résistantes aux collisions s'impose (signatures électroniques, *etc.*). On peut montrer que...

PROPRIÉTÉ I : Les fonctions de hachage résistantes aux collisions sont des fonctions à sens unique.

III L'attaque dite « des anniversaires »

Soit h une fonction de hachage de Σ^* dans Σ^n.

Nous décrivons, dans cette section, une attaque simple à la résistance forte aux collisions d'une fonction de hachage, appelée l'*attaque des anniversaires*.

L'attaque consiste à calculer autant de valeurs hachées que le temps et l'espace le permettent, et à les ranger (dans un ordre donné) avec leur images inverses, afin de chercher une collision.

PROPRIÉTÉ II : Si k chaînes de caractères sont choisies dans Σ^*, avec

$$k \geqslant \frac{1 + \sqrt{1 + (8 \ln 2)|\Sigma|^n}}{2}$$

où $|\Sigma|$ désigne le cardinal de Σ, alors la probabilité que deux valeurs hachées soient égales est supérieure à $\frac{1}{2}$.

PREUVE 1 :

Supposons que les n valeurs hachées possibles appartiennent à l'ensemble $\{1, \ldots, n\}$.

Un événement élémentaire est un k-uple $(b_1, b_2, \ldots, b_k) \in \{1, 2, \ldots, n\}^k$. S'il se produit, alors la valeur hachée du i^e message est b_i, $1 \leqslant i \leqslant k$, et nous avons n^k événements élémentaires. En supposant ces événements élémentaires également probables, la probabilité de chacun est $\frac{1}{n^k}$.

Nous voulons calculer la probabilité p que deux messages aient le même condensé. D'abord, $q = p - 1$ est la probabilité que deux messages aient des valeurs hachées différentes ; nous allons calculer cette probabilité.

L'événement qui nous intéresse est l'ensemble E de tous les vecteurs $(g_1, \ldots, g_k) \in \{1, 2, \ldots, n\}^k$ dont tous les coefficients sont différents. Puisque la probabilité d'un événement élémentaire est $\frac{1}{n^k}$, la probabilité de E est le nombre d'éléments de E divisé par n^k. Or, le nombre d'éléments de E est le nombre de vecteurs dans $\{1, 2, \ldots, n\}^k$ dont les coefficients sont différents.

Le premier coefficient peut être n'importe lequel parmi les n possibilités. Une fois le premier coefficient fixé, il n'y a plus que $n - 1$ possibilités pour le deuxième coefficient, et ainsi de suite. Nous obtenons donc

$$|E| = \prod_{i=0}^{k-1} (n - i)$$

et

$$q = \frac{1}{n^k} \prod_{i=0}^{k-1} (n - i) = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right)$$

Puisque $1 + x \leqslant e^x$ pour tout nombre réel x, la formule précédente donne

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

Nous supposerons pour simplifier que $\Sigma = \{0, 1\}$. Alors

$$k \geqslant f(n) = \frac{1 + \sqrt{1 + (8 \ln 2) 2^n}}{2}$$

est suffisant. Le tableau suivant donne les valeurs de $\log_2 f(n)$ pour des tailles typiques de n.

n	50	100	150	200
$\log_2 f(n)$	25,24	50,24	75,24	100,24

Bref, en calculant plus de $2^{\frac{n}{2}}$ valeurs hachées, l'*attaque des anniversaires* a plus d'une chance sur deux de trouver une collision. Pour se prémunir contre une telle attaque, n doit être choisi de sorte que le calcul de $2^{\frac{n}{2}}$ valeurs hachées soit infaisable.

Il est recommandé de prendre $n \geqslant 128$, voire $n \geqslant 160$. Ces recommandations sont cependant relatives aux performances actuelles des ordinateurs ; elles ne sont que provisoires et ne correspondent en aucun cas à un seuil théorique d'« infaisabilité ».

IV Construction de fonctions de hachage

1 Fonctions de hachage à partir des fonctions de chiffrement

En l'état actuel de nos connaissances, on ne saurait dire s'il existe une fonction de hachage résistante aux collisions[3]. Cependant, on sait faire une fonction de hachage à partir d'un système de chiffrement, qui semblerait résister aux collisions *tant que le système de chiffrement est sûr*. C'est ce qui va maintenant être décrit...

Considérons un cryptosystème où les espaces de messages, de cryptogrammes et de clés sont tous égaux à $\{0;1\}^n$, et notons

$$e_k : \{0;1\}^n \longrightarrow \{0;1\}^n, k \in \{0;1\}^n$$

les fonctions de chiffrement.

Les valeurs hachées sont de longueur $n \geqslant 128$, pour se protéger contre l'attaque des anniversaires.

PROPRIÉTÉ III : Des fonctions de hachage

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

peuvent être définies en posant

$$\begin{aligned} h(k,x) &= e_k(x) \oplus x \\ h(k,x) &= e_k(x) \oplus x \oplus k \\ h(k,x) &= e_k(x \oplus k) \oplus x \\ h(k,x) &= e_k(x \oplus k) \oplus x \oplus k \end{aligned}$$

Ces fonctions de hachage *semblent* être résistantes aux collisions, tant que le cryptosystème est sûr.

PREUVE 2 :
Cette assertion reste à démontrer : c'est une conjecture. ■

2 Fonctions de hachage à partir des fonctions de compression

On peut utiliser des fonctions de compression résistantes aux collisions pour construire des fonctions de hachage, elles aussi résistantes aux collisions...

[3] De même qu'on ne sait pas s'il existe un procédé de chiffrement sûr et efficace.

2.1 Construction de la fonction de hachage

Soit $g : \{0;1\}^m \longrightarrow \{0;1\}^n$ une fonction de compression, et $r = m - n$ (un exemple typique : $n = 128$ et $r = 512$). r est positif, puisque g est une fonction de compression.

Nous expliquons ici la construction de la fonction de hachage $h : \{0;1\}^* \longrightarrow \{0;1\}^n$ (à partir de g) dans le cas $r > 1$.

Soit donc $x \in \{0;1\}^*$.

1. x est complété pour que sa longueur soit divisible par r. Pour cela, on place un nombre adéquat de zéros au début de x.
2. Puis r zéros sont placés à la fin de cette nouvelle chaîne de caractères, ce qui fournit le mot binaire normalisé $\tilde{x}$.
3. Ensuite nous écrivons la représentation binaire de la longueur de x en plaçant des zéros au début de cette représentation pour que le nombre de ses bits soit divisible par $r - 1$.
4. Nous découpons alors la chaîne résultante en mots de longueur $r - 1$, et nous plaçons un 1 au début de chacun d'eux.
5. Enfin, le mot binaire ainsi construit est collé à la fin de $\tilde{x}$.

On peut voir la chaîne de caractères ainsi obtenue comme une suite

$$x = x_1 x_2 \dots x_t \quad \text{où} \quad x_i \in \{0;1\}^r$$

de mots de longueur r.

REMARQUE 2. On peut noter que chacun des mots qui se trouvent à la fin (dans la partie qui représente la longueur de x) commence par le bit 1.

EXEMPLE 6. Avec $r = 4$, prenons $x = 111011$.

Pour commencer, x est transformé en 00111011 pour que sa longueur soit divisible par 4, puis 0000 est inscrit à la fin de ce mot, pour donner $\tilde{x}$ = 001110110000.

La longueur de x est 6, dont le développement binaire est 110 (il n'y a pas de 0 à ajouter : la longueur de ce mot est divisible par 3).

On place alors un 1 devant, ce qui donne 1110 et finalement, on obtient le mot binaire normalisé 0011101100001110.

$h(x)$, la valeur hachée, est calculée par récurrence :

1. H_0 est constituée de n zéros.

2. Puis, on calcule[4] $H_i = g(H_{i-1} \circ x_i)$, pour $1 \leqslant i \leqslant t$.

Et, finalement,

$$h(x) = H_t$$

2.2 Résistance aux collisions

PROPRIÉTÉ IV : Si g est résistante aux collisions, alors h (ainsi construite) l'est aussi.

PREUVE Nous allons démontrer qu'à partir d'une collision de h, on peut déterminer une collision de g.

Soit donc (x, x') une collision de h. On note $x_1, x_2, \ldots x_t, x'_1, \ldots, x'_{t'}$ les suites de blocs pour les x et x' normalisés (comme précédemment), et $H_0, \ldots, H_t$ et $H'_0, \ldots, H'_{t'}$ les suites de valeurs hachées qui leur correspondent.

Supposons $t \leqslant t'$. Comme (x, x') est une collision de h, on a

$$H_t = H'_{t'}$$

Supposons d'abord qu'il existe un indice i avec $0 \leqslant i < t$ tel que

$$H_{t-i} = H'_{t'-i} \quad et \quad H_{t-i-1} \neq H'_{t'-i-1}$$

Alors

$$H_{t-i-1} \circ x_{t-i} \neq H'_{t'-i-1} \circ x'_{t'-i}$$

et

$$g(H_{t-i-1} \circ x_{t-i}) = H_{t-i} = H'_{t'-1} = g(H'_{t'-i-1} \circ x'_{t'-i})$$

est une collision de g.

Donc, s'il n'a pas été trouvé de collision pour g, c'est que

$$H_{t-i} = H'_{t'-i}$$

pour tout $0 \leqslant i \leqslant t$. S'il existe un indice i avec $0 \leqslant i \leqslant t-1$ et $x_{t-i} \neq x'_{t'-i}$, alors

$$H_{t-i-1} \circ x_{t-i} \neq H'_{t'-i-1} \circ x'_{t'-i}$$

mais, comme

$$g(H_{t-i-1} \circ x_{t-i}) = H_{t-i} = H'_{t'-i} = g(H'_{t'-i-1} \circ x'_{t'-i})$$

[4] $\circ$ désigne ici l'opérateur de concaténation.

nous avons encore trouvé une collision de g. Finalement, dans tous les cas, on trouve une collision de g.

Reste donc à montrer qu'il existe un indice i, avec $0 \leqslant i < t$, tel que

$$x_{t-i} \neq x'_{t'-i}$$

Si le nombre de mots requis pour représenter la longueur de x est plus petite que le nombre de mots requis pour représenter de façon normalisée la longueur de x', la chaîne de caractères située entre le x initial et la représentation normalisée de sa longueur est la chaîne nulle, alors que la chaîne de caractères située au même endroit dans la représentation normalisée de x' est non nulle.

Si le nombre de mots requis pour représenter de façon normalisée la longueur de x est la même que le nombre de mots requis pour représenter de façon normalisée la longueur de x', mais si la longueur de x est différente de la longueur de x' alors, les représentations des longueurs contiennent un mot différent situé au même endroit.

Finalement, si les longueurs de x et x' sont les mêmes, les chaînes de caractères représentant les x et x' d'origine contiennent des mots différents qui occupent la même place.

Nous avons montré comment trouver une collision de g à partir d'une collision de h. ■

REMARQUE 3. Pour obtenir un théorème « digne de ce nom », il faudrait avant tout définir proprement la notion de *résistance aux collisions*.

V Exemples d'utilisation des fonctions de hachage

L'utilisation des fonctions de hachages est fréquente en informatique, dans différents cas de figure...

1 Le contrôle d'accès

Il est évident qu'un mot de passe ne doit pas être stocké en clair sur une machine : on conserve uniquement sa valeur hachée (obtenue par SHA-1 ou MD5, habituellement). L'ordinateur comparera alors, lors de l'identification d'un utilisateur, l'empreinte du mot de passe stocké avec l'empreinte de celui saisi au moment de l'authentification.

Cependant, si deux utilisateurs choisissent le même mot de passe, alors les condensés ainsi obtenu seront identiques, ce qui constitue une faille de sécurité. On pourrait, par exemple, « espérer » que ce mot de passe n'est pas une quelconque suite de symboles, mais un mot courrant, et donc faire une attaque par dictionnaire : calculer toutes les valeurs hachées des entrées d'un dictionnaire, et les comparer au dit mot de passe.

Il est possible de contrer ce type d'attaque : dans les faits, on ajoute une composante aléatoire lors de la génération initiale de l'empreinte. Cette composante, que l'on

appele « sel », qui est par exemple l'heure d'attribution du mot de passe, peut être stockée en clair. Il reste alors à concaténer le mot de passe et le sel, pour obtenir deux signatures différentes avec le même mot de passe.

2 Codes d'authentification de message

On a vu que l'on pouvait utiliser des fonctions cryptographiques de hachage pour tester si un fichier a été modifié : la valeur hachée du fichier est stockée séparément, et l'intégrité du fichier est testée en comparant la valeur hachée du fichier actuel à la valeur hachée stockée.

Pour prouver l'intégrité d'un document *et son authenticité*, on peut utiliser des fonctions de hachage paramétrées.

DÉFINITION 8. *Une fonction de hachage paramétrée est une famille* $\{h_k, k \in \mathcal{K}\}$ *de fonctions de hachage, où* $\mathcal{K}$ *est un ensemble appelé l'*espace des clés *de* h. ♢

Une fonction de hachage paramétrée s'appelle un *code d'authentification de message* (on parle encore de MAC[5]).

EXEMPLE 7. Soit une fonction de hachage $g : \{0;1\}^* \longrightarrow \{0;1\}^4$. On peut construire un MAC ainsi :

$$\begin{array}{rcl} h_k : \{0;1\}^* & \longrightarrow & \{0;1\}^4 \\ x & \longmapsto & g(x) \oplus k \end{array}$$

qui admet $\{0;1\}^4$ pour espace des clés.

Nous illustrons comment utiliser un MAC.

EXEMPLE 8. Le professeur Alice envoie par mail, au service de la scolarité, la liste des étudiants reçus à l'examen de cryptographie : ce service doit pouvoir être assuré de l'authenticité de ces notes. Pour ce faire, un MAC $\{h_k, k \in \mathcal{K}\}$ est utilisé.

Alice et le service des examens échangent une clé secrète $k \in \mathcal{K}$ et, avec sa liste x, Alice envoie aussi la valeur hachée $y = h_k(x)$.

Bernard, le responsable du service, calcule la valeur hachée $y' = h_k(x')$ du message reçu : il accepte x' si $y = y'$.

Le protocole de l'exemple précédent prouve l'authenticité, à condition qu'il soit impossible de calculer le couple $(x, h_k(x))$ sans connaître k.

[5] Message Authentification Code.

VI Conclusion

MD5 et SHA-0 ont été cassé en 2004, et l'équipe de Wang a découvert (conférence CRYPTO-2005) une attaque de SHA-1 en 2^{69} opérations, c'est à dire plus rapide d'un facteur 2000 par rapport à l'attaque par force brute en 2^{80}.

Si 2^{69} opérations reste à l'heure actuelle hors de portée du commun des ordinateurs, cette attaque, basée sur une précédente attaque de SHA-0, reste un résultat très important dans le domaine de la cryptanalyse.

Le nombre de fonctions de hachage cryptographique sûres commence à se réduire, d'autant que l'on suspecte SHA-2 de n'être plus aussi sécurisé qu'on a pu le croire. Le monde de la cryptographie en est à se demander s'il est réellement possible de construire un algorithme de hachage suffisamment solide.

Quoi qu'il en soit, il est grand temps de réfléchir à un nouveau standard. Aussi, le NIST (l'organisme de normalisation des standards et de la technologie aux USA) a annoncé le 23 janvier 2007 l'ouverture d'un processus de développement de nouveaux algorithmes de hashage standard.

Comme pour AES, la communauté cryptographique est donc invitée à proposer des algorithmes et parmi eux un ou plusieurs seront sélectionnés à l'issue du processus. Ces fonctions doivent pouvoir retourner des condensés de tailles 256, 384 et 512 bits (comme SHA-2 donc).

Si tout se passe bien, le nouveau standard devrait être connu à la fin 2011.

ANNEXE : SHA-1 et autres fonctions de hachage

Une fonction cryptographique de hachage fréquemment utilisée est SHA-1. Elle intervient, par exemple, dans le DSS (Digital Signature Standard). Nous allons décrire son algorithme.

1 Normalisation du message à hacher

Soit $x \in \{0;1\}^*$, dont la longueur $||x||$ est supposée plus petite que 2^{64}. Pour calculer la valeur hachée de x, on procède ainsi...

On commence par ajouter des éléments à x pour que sa longueur soit un multiple de 512, en procédant ainsi :

1. 1 est ajouté à la fin de x (*i.e.* $x \leftarrow x \circ 1$).
2. On ajoute des 0 à la fin de x pour que $||x|| = 512n - 64$.
3. $||x||$ est écrite en base 2 sur 64 bits, qui sont rajoutés à la fin du mot ci-dessus.

EXEMPLE 9. Soit x le mot (binaire)

$$01100001\ \ 01100010\ \ 01100011\ \ 01100100\ \ 01100101$$

Après la première étape, x devient

$$01100001\ \ 01100010\ \ 01100011\ \ 01100100\ \ 01100101\ \ 1$$

$||x|| = 41$, nous devons donc ajouter 407 zéros à x pour que cette longueur devienne 448=512-64. En représentation hexadécimale, x est maintenant

61626364	65800000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000		

À l'origine, $||x||$ était égale à 40 ; alors on écrit 40 en base 2 comme un nombre de 64 bits ; en représentation hexadécimale :

$$00000000\ 00000028$$

et on colle ce nombre à la fin de x, ce qui donne finalement

61626364	65800000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000028

---

2 Calcul de la valeur hachée

Pour calculer la valeur hachée, on a besoin de 80 fonctions

$$f_t : \{0;1\}^{32} \times \{0;1\}^{32} \times \{0;1\}^{32} \longrightarrow \{0;1\}^{32}$$

définies de la façon suivante[6]

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & \text{pour } 0 \leqslant t \leqslant 19 \\ B \oplus C \oplus D & \text{pour } 20 \leqslant t \leqslant 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{pour } 40 \leqslant t \leqslant 59 \\ B \oplus C \oplus D & \text{pour } 60 \leqslant t \leqslant 79 \end{cases}$$

On utilise aussi des constantes

$$K_t = \begin{cases} 5A827999 & \text{pour } 0 \leqslant t \leqslant 19 \\ 6ED9EBA1 & \text{pour } 20 \leqslant t \leqslant 39 \\ 8F1BBCDC & \text{pour } 40 \leqslant t \leqslant 59 \\ AC62C1D6 & \text{pour } 60 \leqslant t \leqslant 79 \end{cases}$$

Soit x un mot binaire qui a été formaté en suivant la règle précédente. Sa longueur est donc divisible par 512, et on peut l'écrire comme une suite de mots de 512 bits

$$x = M_1 M_2 \dots M_n$$

Pour l'initialisation, on utilise les mots de 32 bits :

- $H_0 = 67452301$,
- $H_1 = EFCDAB89$,
- $H_2 = 98BADCFE$,
- $H_3 = 10325476$,
- $H_4 = C3D2E1F0$.

Soit $S^k(w)$ le décalage à gauche de k bits (en permutation circulaire), d'un mot de 32 bits w, et + l'addition mod 2^{16} des entiers correspondant aux mots de 16 bits.

On exécute maintenant la procédure suivante, pour $i = 1, 2, \dots, n$:

1. Écrire M_i comme une suite $M_i = W_0 W_1 \dots W_{15}$ de 16 mots de 32 bits.
2. Pour $t = 16, 17, \dots, 79$, calculer $W_t = S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$.
3. Poser $A = H_0, B = H_1, C = H_2, D = H_3$, et $E = H_4$.
4. Pour $t = 0, 1, \dots, 79$, calculer
   - $T = S^5(A) + f_t(B,C,D) + E + W_t + K_t$,
   - $E = D$,
   - $D = C$,

[6] $\wedge$ représente le *et* logique exécuté bit à bit, $\vee$ est le *ou* logique, quand $\neg$ est la négation.

– $C = S^6(B)$,
– $B = A$,
– $A = T$.

5. Ajouter A à H_0, B à H_1, C à H_2, D à H_3 et E à H_4.

La valeur hachée est alors

$$SHA-1(x) = H_0H_1H_2H_3H_4$$

3 Autres fonctions de hachage

Les autres fonctions de hachage utilisées en pratique sont construites comme dans la section IV. Le tableau ci-dessous donne certaines caractéristiques de fonctions de hachage célèbres.

fonction de hachage	longueur des blocs	vitesse relative
MD4	128	1,00
MD5	128	0,68
RIPEMD-128	128	0,39
SHA-1	160	0,28
RIPEMD-160	160	0,24

Toutes ces fonctions sont très efficaces.

La fonction MD4 ne peut plus être considérée comme résistante aux collisions[7]. De même, MD5 n'est plus totalement sûre.

Enfin, compte tenu de l'attaque des anniversaires, les 160 bits de SHA-1 impliquent une résistance à 2^{80} essais pour l'attaque brutale. On exige dorénavant une résistance à 2^{128} essais pour les circuits de chiffrement.

[7] Dobbertin a trouvé une collision en calculant 2^{20} valeurs hachées.

GLOSSAIRE

Algorithme de hachage cassé : Un algorithme de hachage est « cassé » quand il existe une méthode de coût bien moindre que la recherche brute pour générer de façon déterministe des collisions, c'est-à-dire deux fichiers ayant la même empreinte.

Attaque : Une attaque d'un algorithme de hachage consiste à essayer de le casser.

Chiffrement : Le chiffrement (parfois appelé à tort cryptage) est le procédé grâce auquel on peut rendre la compréhension d'un document impossible à toute personne qui n'a pas la clé de (dé)chiffrement.

Cryptogramme : Synonyme de texte chiffré ou de « message chiffré ».

Cryptosystème : Terme utilisé en cryptographie pour désigner un ensemble composé d'algorithmes cryptographiques et de tous les textes en clairs, textes chiffrés et clés possibles.

Espace de message : Ensemble d'« objets » dont on souhaite réaliser le chiffrement.

Fonction de chiffrement : fonction qui permet le chiffrement d'un message en son cryptogramme.

Racine primitive : Une racine primitive modulo n est un entier g tel que, modulo n, chaque autre entier est juste une puissance de g.

Système de chiffrement : Synonyme de cryptosystème.