Univesity of Bordeaux
College of Science & Technology
351 Cours de la Liberation
33400 Talence

- TER Oral -

Sponge Function's Implementation

Amelie GUEMON & Ida TUCKER

Master 1 CSI — Cryptology & Computer Security

April 24, 2016

En faisant subir une suite de traitements reproductibles à une entrée, une fonction de hachage génère une empreinte servant à identifier la donnée initiale. Cette identification par le biais de fonctions de hachage est un outil indispensable pour de nombreux protocoles cryptographique, du stockage de mots de passes à l'authentification en passant par l'intégrité de messages de grande taille, ils présentent de nombreux intérêts.

Nous nous sommes intéressées à l'évolution des fonctions de hachage, dû à la publication en Août dernier d'une nouvelle famille de fonctions de hachage par le NIST.

Le NIST est l'organisme de normalisation des standards et de la technologie aux USA.

La particularité de cette nouvelle famille est qu'elle repose sur des principes fondamentalement différents de toutes les familles de fonctions ayant été standardisées au part avant. Nous nous sommes donc demandées ce qui a poussé le NIST à chercher une telle différence radicale.

Chapter 1

INTRODUCTION

1.1 Qu'est-ce qu'une fonction de hachage

Une fonction de hachage est un algorithme permettant de calculer une empreinte de taille fixe à partir d'une donnée de taille quelconque en appliquant une série de transformations pour réduire ces données.

On obtient à la sortie une chaîne de caractères hexadécimaux, l'empreinte, qui résume en quelque sorte le fichier.

A l'origine, les fonctions de hachage ont été créés pour faciliter la gestion de base de données. Plutôt que de manipuler des données de taille variable et potentiellement grande, on associe à ces données une empreinte de taille fixe, sur lesquelles des comparaisons sont plus rapides à effectuer. L'utilisation de fonctions de hachage permet par conséquent d'accélerer des opérations comme le tri, l'insertion ou l'accès à un élément.

Cette sortie a une taille fixe qui varie selon les algorithmes (128 bits pour MD5 et 160 bits pour SHA-1).

Remarquons qu'une fonction de hachage ne peut être injective, par conséquent il y aura forcément plusieurs messages menant aux mêmes hachés.

1.2 Fonctions de hachage cryptographiques

Pour qu'une fonction de hachage soit utilisée à des fin cryptographiques, (il s'agit alors d'une fonction de hachage cryptographique) elle doit satisfaire certaines contraintes.

- Il doit être difficile de retrouver ou générer un texte à partir de l'empreinte: **Résistance à la première pré-image**. Il est alors difficile pour qu'un attaquant puisse trouver un message qui mène à un haché donné (signatures falsifiées si Oscar a une liste de signatures que Bob acceptera et qu'il trouve un message qui est valide pour cette signature, la fonction de hachage ne garantie plus l'intégrité).

- La moindre modification dans le fichier original 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: **Résistance à la seconde pré-image**. La résistance

à la seconde pré-image garantie l'intégrité d'un fichier: si elle n'est pas respectée, ou pourrait facilement générer un fichier corrompu mais valide aux yeux de l'utilisateur.

- De même il doit être difficile pour un attaquant de trouver deux messages ayant le mêmee haché, car il pourrait alors substituer un message par un autre, ou nier avoir envoyé un message (intégrité et non répudiation): **Résistance aux collisions**.

1.3 Padding

Étant donné qu'une fonction de hachage agit de façon itérative sur des blocks du message en entrée, le message doit en général être paddé, afin que le message soit de longueur un multiple de la longueur d'un block.

Il faut faire attention à la règle de bourrage utilisée, car si on la choisit trop simple, des collisions peuvent facilement être crées.

Dans le cas de SHA1 et MD5, basés sur Merkle Damgård, les algorithmes agissent sur des blocs de 512 bits. On rajoute un 1, puis un nombre fini de 0, de telle sorte que la longueur du resultat soit congru à 448 mod 512. Ensuite, on y ajoute la longueur du message, sur 64 bits.

Chapter 2

MERKLE DAMGARD

2.1 Idée

L'approche traditionnelle pour construire des fonctions de Hachage a été de se reposer sur une construction appelée la construction de Merkle-Damgard.

Cette construction utilise des fonctions de compression résistantes aux collisions pour construire des fonctions de hachage, elles-aussi résistantes aux collisions.

Une fonction de compression part d'un ensemble fini vers un ensemble fini de taille réduite alors qu'une fonction de hachage part d'un ensemble infini vers un ensemble fini.

2.2 Construction

La construction de MD reçoit en entrée une valeur initiale IV de même taille que l'empreinte et un message de longueur arbitraire.

Le message est d'abord paddé afin d'obtenir une longueur adéquate, puis scindé en blocks de 512 bits.

On itère ensuite la fonction de compression sur les blocks de taille fixe du message.

Après la première itération, le fonction de compression prend en entrée la sortie de l'itération précédente au lieu de l'IV.

Chapter 3

ATTAQUES

3.1 Attaque par force brute: l'attaque des anniversaires

La meilleure attaque générique en recherche de collisions repose sur le paradoxe des anniversaires. Cette expression désigne une propriété contre-intuitive : parmi un groupe d'au moins 23 personnes prises au hasard, il y a au moins une chance sur deux pour que deux d'entre elles aient leur anniversaire le même jour.

Le calcul de hachés de messages aléatoires correspond à un tirage aléatoire sur l'espace des empreintes , de taille 2^n. D'après le raisonnement précédent, le nombre de hachés qu'il faut calculer pour trouver une collision est de l'ordre de $\sqrt{2^n} = 2^{n/2}$

Si l'on suppose que l'ensemble de messages $\mathcal{E}$ est choisi de façon aléatoire et uniforme parmi tous les messages possibles, on peut montrer que le nombre de messages à tester afin d'avoir une probabilité donnée de trouver des collisions ne dépend que de la taille des condensés.

Ainsi, afin d'avoir une probabilité supérieure à 50% de trouver des collisions, il suffit de tester aléatoirement peu plus de $\sqrt{N}$ messages.

Dans le cas de SHA1 et MD5, puisque les condensés sont de 128 et 160 bits respectivement, l'attaque par force brute se fait de façon naïve en 2^56 et 2^80 essais.

Pour donner une idée en temps, 8 cartes graphiques NVidia Titan X peuvent calculer 115840 millions de condensés MD5 par seconde. Cela prendrait donc ≈ 5 ans afin de trouver une collision avec un tel dispositif. Ainsi avec une armée de machines dédiées à cette tâche, il serait relativement facile de trouver des collisions pour MD5 en un temps raisonnable.

L'attaque par force brute fourni donc une taille minimale nécéssaire pour la taille des condensés afin de garantir une certaine sécurité de base.

Il est recommandé de prendre $n \geq 128$, voir $n \geq 160$. 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 de sécurité.

3.2 Quand une fonction de hachage est-elle considérée cassée?

Une fonction de hashage est dite **cassée** lorsqu'il existe une attaque connue permettant de trouver des collisions ayant une complexité moindre que l'attaque par force brutte.

3.3 MD5 et SHA1: cassés en 2004

Il existe des algorithmes aujourd'hui capables de trouver des collisions pour MD5 en quelques secondes, et l'attaque de SHA1 découverte par l'équipe de Wang en 2004 a une probe supérieure à 50% de trouver des collisions en 2^{69} opérations, c'est a dire plus rapide qu'une attaque par force brute (2^80) d'un facteur 2000. Si 2^{69} reste à l'heure actuelle hors de portée du commun des ordinateurs, cette attaque reste un résultat important dans le domaine de la cryptanalyse, et est suffisant pour qu'il soit déconseillé d'utiliser SHA1 aujourd'hui. Le nombre de fonctions de hachage cryptographique sûres a réduit considérablement dans la dernière décennie, d'autant que l'on suspecte SHA-2 de n'être plus aussi sécurisée qu'on a pu le croire. En effet, bien que SHA-2 ne soit pas cassée, et que son implémentation reste recommandée par le NIST il existe des attaques contre des versions réduites de SHA-2 (la fonction de compression est itérée sur 46 tours au lieu de 64).

Quoi qu'il en soit, ces progrès importants dans la cryptanalyse des fonctions de hachage ont poussé le NIST à développer de nouveaux algorithmes de hachage standards. La recherche de cette nouvelle norme c'et faite sous la forme d'un concours, comme pour AES, la communauté cryptographique était invitée à proposer des algorithmes et parmi eux un ou plusieurs seront sélectionnés à l'issue du processus.

Le NIST souhaitait un algorithme reposant sur une construction complètement différente que Merkle-Damgard, ainsi, même si une attaque est découverte contre ce types de construction, remettant en question la sécurité de SHA2, l'algorithme pour SHA3 ne serait pas affecté.

L'algorithme qui fut sélectionné pour SHA3, en Aout dernier, à l'issu de ce concours repose sur une construction dite en éponge.

Chapter 4

FONCTIONS EN ÉPONGE

4.1 Construction de fonctions en éponge

La construction de l'éponge créé une fonction dite en éponge. Cette construction repose sur:

- Une transformation ou permutation de taille fixe:

$$f\{0,1\}^b \rightarrow \{0,1\}^b$$

- Une règle de bourrage adaptée à la construction de l'éponge
- La quantité de bits du message traitée à chaque permutation le **bitrate** r

Une sortie de taille finie fixée peut être obtenue en tronquant la sortie, mais à la différence de la construction de Merkle-Damgard, la construction de l'éponge prend en entrée un message de taille quelconque, applique une série de transformations et on obtient à la sortie une chaîne de bits de taille quelconque.

La permutation f opère sur un nombre fini de bits appelé la **largeur** b. La construction de l'éponge a un état de b bits.

Notez qu'une fonction en éponge opère différement sur les r premiers bits de l'état, qui constituent la *partie interne de l'état* des $c = b - r$ derniers bits de l'état, qui eux constituent la partie externe de l'état.

Tout d'abord, la règle de padding est appliquée au message M qui est ensuite scindé en blocks de r bits. Et les bits de l'état sont tous initialisés à 0.

La construction de l'éponge opère selon deux modes de fonctionnement:

- La phase d'absorption, au cours de laquelle la construction *absorbe* le message en entrée. Un XOR est effectuée entre les blocs du message paddé de longueur r et la partie interne de l'état. Suivie d'une application de la permutation f sur la totalité des bits de l'état. Ces étapes sont répétées jusqu'à ce que tous les blocs du message en entrée soient absorbés.
- La deuxième phase est l'*essorage*. Si la taille de la sortie l désirée est inférieure à c, le résultat de l fonction en éponge est constitué des l premiers

bits de l'état. Sinon, on sort les r premiers bits de l'état, puis on applique f à l'état dans sa totalité, et on itère le procédé jusqu'à obtenir une sortie de longueur l.

Et la sortie est tronquée à ses l premiers bits.

Remarque: La partie interne de l'état n'est jamais directement affectée par les blocs du message en entrée, et ne sont jamais directement sortis.

La capacité c détermine en fait la sécurité que peut atteindre la construction.

4.2 Keccack

La fonction de compression Keccak travaille sur les mots de $w = 2^k$ bits.

Pour un processeur de 64 bits il est naturel de choisir $w = 64$.

- Θ: La première routine Θ calcule la parité de chaque colonne, puis effectue des ou exclusifs entre les parités de certaines colonnes deux à deux (cela correspond donc à la parité de deux colonnes). Enfin un XOR est effectué entre chaque bit de l'état et une des parités de deux colonnes calculée précédemment.
- ρ: Décalage circulaire des rangées (ou mots) de l'état: rotation des rangées par un 'offset' qui dépend de la rangée.
- π: Permutation des 25 mots avec un motif fixé: réarrange la position des rangées.
- χ: XOR's entre chaque bit de l'état et une fonction non linéaire dépendant de deux autres bit dans la même ligne (y et z fixés). Décalage circulaire des rangées (ou mots) de l'état.
- ι: Calculer le ou exclusif d'une constante avec un mot de l'état, qui dépend du numéro de l'itération i_r:

4.3 SHA-3

Processeur de 64 bits $\Rightarrow$ choix naturel $w = 64$.

Chapter 5

Conclusion

Ainsi, dû à des avancées majeures dans le domaine de la cryptanalyse des algorithmes de hachage basés sur la construction de Merkle-Damård dans la dernière décennie, il a été nécessaire de s'éloigner de cette brique de base sur laquelle reposaient tous les algorithme de hachages cryptographiques standardisés.

Cette perspective a ainsi mené à la mise au point d'une nouvelle norme, la famille SHA-3, qui ne repose non plus sur une fonction de compression, mais sur une construction en éponge, qui itère une permutation de taille fixe. Du fait de son modèle distinct, les attaques affectant le modèle de Merkle-Damgård ne sont pas efficace contre SHA-3. Elle présente de plus l'avantage de pouvoir choisir la taille des empreintes produites.

Cependant, il existe des modification mineurs, qui sont facilement implémentables en pratique, qui permettent également de palier aux défauts de la structure de Merkle-Damgård.

Ainsi il pourrai paraître risqué de se lancer directement dans une implémentation majeure de SHA3 pour des protocoles de sécurité déployés à grande échelle. En effet la construction en éponge est encore très jeune, il est fort possible qu'il existe des faiblesses importantes pour SH3 auxquelles nous n'avons tout simplement pas encore pensé.