Projet RNRT SAPHIR

Sécurité et Analyse des Primitives de Hachage Innovantes et Récentes

L5.3

Synthèse des Nouvelles Conceptions

Contributeur(s)

Pierre-Alain Fouque

Date de livraison du livrable: M36
Date courante: 20 avril 2009
Partenaire Saphir en charge: École normale supérieure

Niveau de confidentialité: Accès public **Version:** 1.0

Historique

Version	Date	Auteur	Modification
1.0	12/04/2009	Moi-même	Améliorations diverses

Parnenaires Saphir

Date de départ du projet: 10 Mars 2006 Durée: 3 ans

Les informations présentées dans ce document ne peuvent constituer un engagement légal de quiconque et de quelque nature que ce soit. L'utilisateur du document reconnaît utiliser les informations présentes à ses propres risque et responsabilité.

Table des matières

1 Introduction

Le but de ce document est de présenter la synthèse sur la conception de nouvelles fonctions de hachage. Dans le cadre du projet Saphir, les personnes du projet ont développé deux fonctions, présentées dans le rapport précédent L5.2, SIMD et Shabal. D'autres fonctions ont été présentées par d'autres personnes qui font parti du consortium Saphir 2 comme ECHO par Orange Labs, FSB par des membres de l'INRIA et du projet SECRET, ou CRUNCH par des membres de l'Université de Versailles-Saint Quentin.

Il est en général assez difficile de construire une fonction de hachage, même si plusieurs personnes se sont lancées dans le dernier appel du NIST pour la fonction SHA-3. Bien concevoir une primitive cryptographique est un art qui demande de bien comprendre les attaques précédentes, les contre-mesures qui peuvent être utilisées pour contrer les attaques en aboutissant quelquefois à des preuves de sécurité et enfin, le fonctionnement des machines pour avoir des fonctions réellement efficaces. En terme de vitesse, les performances se mesurent en terme de cycles utilisés pour hacher un octet de messages. L'objectif est donné par SHA-1 qui requiert 7,8 cycles/octets et SHA-2 qui demande environ 20 cycles/octets sur un Core 2 Duo qui est la plate-forme de référence demandée par le NIST. Les performances varient en fonction des processeurs et donc, pour uniformiser les performances nous ne donnerons ici, que des chiffres pour cette plate-forme.

Il existe plusieurs familles pour concevoir une fonction de hachage. Dans le cadre du projet Saphir, il est difficile de dire laquelle des familles est la meilleure. Ainsi, faute d'accord commun qui relève plus de la philosophie que de réelle réflexion scientifique, nous présentons les familles avec leurs avantages et leurs inconvénients. Nous mettrons plus l'accent sur la famille des fonctions ad-hoc qui semble la plus utilisée par les différentes personnes qui ont soumis une fonction à la compétition du NIST. Dans cette partie, nous détaillerons un peu la philosophie sous-jacente à la conception de trois grandes sous-familles en prenant comme exemple, ECHO, SIMD et Shabal, issues du projet Saphir et qui sont toutes les trois candidates à l'appel du NIST.

2 Modes opératoires

Ici, nous suivons la direction adoptée par le NIST qui a souhaité que les personnes qui soumettent une fonction de hachage décrivent aussi le mode opératoire utilisé. Il semble que cette approche est plus saine que l'approche, qui voulait dissocier la construction du mode opératoire de la construction de la fonction de compression, c'est-à-dire une fonction de hachage qui prend en entrée un bloc de message de taille fixe. En effet, pour certaines fonctions, comme les sponges fonctions (fonctions éponges), il est difficile de séparer la fonction de compression et le mode opératoire. Ensuite, la littérature sur les modes opératoires a été très active récemment. Il s'agit essentiellement de propositions théoriques qui n'ont que peu de répercussion en pratique.

De même qu'en ce qui concerne les fonctions de hachage, il me semble pas qu'il y ait unanimité sur le choix d'un mode opératoire par rapport à un autre. Le mode opératoire HAIFA est très utilisé et est un dérivé de Merkle-Damgard. Cependant pour les fonctions éponges, il n'est pas toujours évident de trouver le mode opératoire. Dans tous les cas, il semble que les nouvelles constructions aient toutes retenues l'idée d'avoir un état plus grand que celui des fonctions précédentes. Ceci permet d'éviter les attaques en multicollision (cf. Joux 2004), qui même si elles demandent beaucoup de puissances de calcul, ont l'inconvénient qu'il est facile de les distinguer d'un oracle aléatoire. Augmenter la taille de l'état interne du double de la taille de sortie s'appelle utiliser la technique *wide-pipe* en suivant la terminologie introduite par Lucks en 2005.

Les modes opératoires proposés dans le cadre de la compétition du NIST semblent assez traditionnel. Quelques modes comme MD6 utilisent un hachage en arbre en recommandant d'utiliser plusieurs cœurs pour hacher plus rapidement, mais la plupart utilise Merkle-Damgard avec un état deux fois plus grand et certains avec un indice du numéro de block comme proposé dans HAIFA. Très peu de modes utilisent une taille de sortie égale à la taille de la variable de chaînage, ce qui n'évitent pas les multicollisions.

3 Fonctions prouvée sûres

La conception de fonction de hachage peut se faire de façon théorique en partant d'un problème prouvé difficile, comme trouver un mot de poids faible dans un code correcteur d'erreur ou dans un réseau ou résoudre un problème d'équations polynomiales.

La fonction FSB est basée sur un problème algorithmique prouvée NP-difficile. Mais la résistance de la fonction n'est pas forcément directement relié à ce problème car la théorie de la NP-difficulté nous dit que pour ce problème, il existe des instances difficiles, alors qu'ici une instance bien particulière est utilisée. En particulier, on sait que ce problème est difficile en général, mais pour des raisons d'efficacité, certains choix sont effectués qui peuvent rendre le problème plus facile. En théorie, en ce qui concerne les problèmes de réseaux (lattices en anglais), les concepteurs font face à des problèmes de tailles de paramètres à partir desquelles, le problème devient difficile. Il est difficile de quantifier alors, la réelle résistance de la fonction. Heuristiquement, des simulations sont nécessaires, mais on a aujourd'hui que très peu de recul pour ce type de problèmes qui sont moins étudié que les problèmes plus classique comme la factorisation ou le logarithme discret.

Enfin, la fonction CRUNCH, fait aussi partie des fonctions de hachage prouvées sûres. Cependant, les problèmes utilisées ici ne sont pas des problèmes de théories des nombres ou algorithmiques, mais des problèmes de cryptographie symétrique sur les schémas de Feistel et le fait que le ou exclusif de deux permutations aléatoires est une fonction aléatoire.

Dans tous les cas, on remarque qu'il est difficile d'avoir des fonctions

réellement efficace et comparable en terme de vitesse avec les constructions de fonctions plus ad-hoc. Le gain en sécurité enfin, ne semble pas suffisant, car ces fonctions sont certes résistantes aux collisions par exemple, mais ont une résistance à d'autres propriétés beaucoup moins sûres. Par exemple, sur certaines fonctions prouvées résistantes aux collisions, il est très facile de casser la préimage ou la seconde préimage. Ces fonctions ne sont en général pas utilisées en pratique, mais peuvent servir dans certaines constructions quand on veut éviter d'avoir à faire une hypothèse ad-hoc ou quand on veut se passer du modèle de l'oracle aléatoire. En particulier, pour la fonction de hachage VSH utilisée dans un schéma de signature, il a été possible d'avoir une preuve de sécurité "presque idéale" dans le modèle standard.

4 Fonctions ad-hoc

La deuxième grande famille est celle des fonctions ad-hoc. Elles regroupent toutes les fonctions qui ont été proposées à l'appel du NIST et qui ne font pas parti de la famille précédente. Il y a deux grands principes :

1. avoir beaucoup de tours peu coûteux en terme de calcul et faire rentrer plusieurs fois le message (ce principe est utilisé dans SHA par exemple) avec des variantes sur comment faire rentrer plusieurs fois le message (parti qui s'appelle l'expansion de message), ou
2. faire rentrer qu'une seule fois le message dans une opération plus complexe et coûteuse en terme de calcul, ou
3. utiliser un état très grand et brasser avec une opération moins coûteuse seulement une partie de l'état.

4.1 ECHO

ECHO a été proposée par Henri Gilbert d'Orange Labs avec plusieurs personnes de ce laboratoire. Cette fonction suit le second principe et utilise les analyses de sécurité du schéma de chiffrement AES.

L'avantage est qu'une preuve de sécurité de l'AES peut être réutilisée pour prouver ECHO et que les implémentations d'ECHO peuvent réutilisées les fonctions développées pour l'AES comme les nouvelles instructions qui seront bientôt disponibles dans les processeurs Intel (cf. invited talk de Shai Geuron à FSE 2009).

L'inconvénient de ce schéma est que les attaques sur l'AES sont directement transférable sur cette fonction et donc, toute faiblesse nouvelle de l'AES impact la sécurité d'ECHO.

L'idée de cette fonction est similaire à celui de Grindahl, Fugue (équipe d'IBM), Groestl (équipes de l'Université du Danemark et de l'Université de Graz), et Cheetah (équipe de l'Université de Luxembourg). On met le message (1536 bits) pour la version 256 bits dans un état, une matrice 16×16 de mots de 128 bits (contrairement à l'AES où les mots sont de 8 bits) avec une variable de chaînage de 512 bits dans la première colonne. Ensuite,

il y a 8 tours où la S-boîtes de l'AES est remplacée par 2 tours d'AES, ShiftRow et le MiColumn mélange les éléments d'une colonne en prenant un octet dans chacun des mots de 128 bits d'une colonne et en appliquant l'opération MixColumn de l'AES sur ce vecteur de 32 bits. Cette opération est faite en parallèle sur chacun des 16 octets d'une colonne. A la fin, les 4 colonnes de l'état sont xorées pour donner un état de sortie de 512 bits. À la fin, on tronque la sortie en gardant les 256 premiers bits de la variable de chaînage. On voit clairement que l'idée ici est de faire rentrer le plus de message possible et ensuite d'utiliser une opération assez compliquée pour mélanger l'état avant de le tronquer.

Les performances de ECHO sont de l'ordre de celles de 1,5 à 2 fois moins que celles de SHA-2, ce qui reste raisonnable pour le niveau de sécurité qu'ECHO semble apporter par rapport à SHA-2, même si aujourd'hui on ne sait attaquer que très peu de tours sur cette fonction.

4.2 Shabal

Shabal a été proposée par beaucoup de personnes du consortium Saphir dont des membres de Cryptolog, de la DCSSI, d'EADS, de France Télécom, de Gemalto, de l'INRIA équipe Secret, et de Sagem Sécurité. L'idée de cette fonction est d'avoir des tours qui coûtent le moins cher possible et d'en faire assez pour bien mélanger l'état. Ce principe est similaire à celui de Blake, CubeHash, etc.

Cette fonction a un état de 44 mots de 32 bits (1408 bits), et prend 16 mots (512 bits) de message. Elle itère une permutation à clé avec un nouveau mode opératoire prouvé sûr en indifférentiabilité. Enfin, il y a des tours à blanc à la fin. La permutation à clé opère sur une entrée de 28 mots (896 bits) paramétrée par deux valeurs de 16 mots (512 bits). Elle utilise 4 registres à décalages dont deux sont perturbées de façon non linéaire.

L'avantage de cette fonction est que les opérations sont très simples et qu'elle est parmi les plus rapides sur la plupart des plate-formes. Son inconvénient est qu'elle est très ad-hoc aussi dans la façon dont la fonction de compression est construite car elle ne repose pas sur un block cipher mais ressemble à Radiogatun ou MD6. Du coup, la résistance de la fonction en terme d'attaque linéaire et différentielle est à faire entièrement. MD6 propose de telles preuves de sécurité, ce qui n'est pas le cas de Shabal. A contratio, Shabal est de loin plus rapide que MD6 et même que SHA-1, environ 7 cycles/octets.

4.3 SIMD

SIMD a été proposé par des personnes de l'ENS. L'idée de départ est d'avoir une fonction traditionnelle comme celles de la famille MD4 en ayant une phase d'expansion de message renforcée par rapport aux fonctions de cette famille.

Afin de limiter le nombre de tours pour rendre le hachage plus rapide, le message est étiré en utilisant un code correcteur garantissant une distance

minimale assez grande. Cela permet d'éviter les attaques différentielles en collision car l'introduction de n'importe quelle différence dans le message donne au moins 520 différences dans le message étendu pour la version 256 bits. De plus, les différences donnant ce nombre minimal de différences ne peuvent pas forcément se calculer facilement. Enfin, le message étendu rentre en parallèle dans une sorte de quatre MD4 en parallèle qui s'échangent de l'information à chaque tour pour une diffusion plus rapide de l'état.

Le code correcteur d'erreur utilisé par SIMD est un code de Reed-Solomon qui est concaténé avec un code non-linéaire implémenté en utilisant une multiplication dans un corps fini. C'est la principale innovation de SIMD qui utilise pour le hachage un principe bien étudié en suivant la philosophie des fonctions de la famille MD4. Le principal avantage de SIMD par rapport à la famille MD4 est qu'avec une distance minimale suffisante, il est possible de faire des preuves (arguments) de sécurité pour la fonction de hachage, ce qui n'est pas possible pour SHA-1, même si Jutla a proposé une variante de SHA-1 utilisant un code correcteur d'erreur *à la SHA-1* ayant une bonne distance minimale. Ici, avec SIMD, il est possible de fournir une preuve de sécurité contre les attaques différentielles même en donnant beaucoup de pouvoir à l'adversaire compte-tenu des attaques couramment utilisées sur la famille MD4. C'est une des premières fois qu'il est possible de faire des preuves de sécurité contre ce type d'attaque pour une fonction de hachage de la famille SHA-1.

SIMD a été prévu pour être rapide sur des processeurs ayant des unités de calcul vectoriel : c'est-à-dire des instructions manipulant plusieurs mots en parallèle. Dans le cas où ce genre d'instructions existent, les performances de SIMD sont 80% celles de SHA-1, c'est-à-dire 10 cycles/octets.

L'inconvénient de SIMD est d'utiliser des instructions vectorielles qui n'existent pas sur tous les processeurs, notamment les cartes à puces. Cependant, SIMD devrait avoir de très bonnes performances en hardware ou sur tous les processeurs fournissant ce type d'instructions, c'est-à-dire sur beaucoup plus de processeurs que seulement les Intel qui fourniront des instructions AES.

5 Conclusion

Dans ce rapport, nous avons décrit les différentes constructions de fonctions de hachage qui ont été principalement utilisées par les différentes personnes qui ont soumis une fonction de hachage à la compétition du NIST. Ce rapport fait suite à la présentation des différentes fonctions faites au workshop organisé par le NIST fin février 2009, juste après la conférence FSE 2009 à Leuven (Belgique).

Le NIST retiendra sûrement une fonction de la famille ad-hoc qui présente le plus de garantie de sécurité et qui soit efficace sur la plupart des plate-formes possibles. Dans les mois à venir, les cryptographes vont étudier la sécurité des différentes fonctions en suivant les indications que le NIST

donnera. Par exemple, le NIST va sélectionner une liste de 15 candidats à la fin du mois d'août 2009, ce qui restreindra la recherche sur ces fonctions.

On peut remarquer que certaines fonctions ont été cassées, mais très peu ont été complètement cassées. Il demeure que même si certaines fonctions ne sont cassées que sur un nombre de tours restreint, ces fonctions ne sont pas réellement bien conçues car augmenter le nombre de tours pour éviter les attaques n'est pas une garantie de sécurité. On peut en réfléchissant bien réduire le nombre de tours et gagner du coup en vitesse en diffusant plus rapidement.

Enfin, on peut noter qu'il y a un grand nombre de fonctions de hachage qui réutilisent des composants de l'AES pour avoir des preuves de sécurité. Ces arguments, plutôt que réelle preuve de sécurité, sont importants pour bien connaître le nombre de tours à partir duquel la fonction est sûre.