Université Paris Diderot - Paris 7

Full Domain Hash

Présenté le 24 octobre 2017 par

Edward Guyot et Gaïd Revaud

Introduction

On appelle Full Domain Hash un schéma de signature utilisant une fonction de permutation à sens unique. Ce schéma est prouvé sécurisé dans un modèle d'oracle aléatoire. Cela signifie qu'il n'est pas possible de forger une signature *i.e.* de retrouver h tel que $h = \mathbf{H}(m)$ où $\mathbf{H}$ est une fonction de hachage inconnue et m un message.

Pour mieux comprendre ce schéma de signature, nous allons commencer par expliquer le fonctionnement d'un oracle aléatoire pour une fonction de hachage. Nous allons ensuite présenter le schéma en deux temps : d'abord avec une fonction de permutation à sens unique générale, puis avec celle que nous connaissons, RSA. Nous finirons en présentant une preuve de la sécurité de ce schéma.

0.1 Oracle aléatoire

Soit $\mathcal{O}$ une fonction qui modélise une fonction de hachage et qui a comme propriété d'être une véritable fonction aléatoire. Par exemple, si nous souhaitons modéliser la fonction de hachage $\mathbf{H}$ telle que $\mathbf{H} : \mathcal{M} \to \mathcal{H}$, on pose $\mathcal{O}$ comme une des fonctions choisie uniformément dans l'ensemble des fonctions $f : \mathcal{M} \to \mathcal{H}$. Cette fonction peut donc être vue comme une boîte noire qui, à chaque message reçu, renvoie une valeur aléatoire de $\mathcal{H}$. Si un même message est reçu plusieurs fois, la même réponse est renvoyée : le couple (message, haché) est sauvegardé en mémoire. Cette fonction $\mathcal{O}$ est appelée un oracle aléatoire.

Étant donné qu'il existe $\mid \mathcal{H} \mid^{\mid \mathcal{M} \mid}$ telles fonctions, il n'est pas possible de toutes les décrire et donc d'utiliser un oracle aléatoire en pratique. Cependant, si un système cryptographique dont toutes les fonctions de hachage ont été modélisées par un oracle aléatoire est prouvé sécurisé, alors on dit que ce système est prouvé sécurisé dans le modèle d'oracle aléatoire. Si cela ne permet pas d'assurer la sécurité dans un modèle standard, cela permet d'écarter un certain nombre d'attaques classiques.

0.2 Full Domain Hash

0.2.1 Rappels

Définition 1. (Fonction de permutation à sens unique)

Soit f une fonction de permutation à sens unique sur $\mathcal{X}$. Alors f est définie par un triplet d'algorithmes $(\mathbf{G}, \mathbf{Perm}, \mathbf{PermInv})$ tels que :

1. $\mathbf{G}()$ est un algorithme aléatoire n'ayant aucune entrée et génère un couple (pk, sk), où pk est une clé publique et sk est une clé secrète.
2. **Perm** est un algorithme déterministe tel que, pour toute clé publique pk fixée, l'application $x \mapsto \mathbf{Perm}(pk, x)$ est une bijection.
3. **PermInv** est un algorithme déterministe tel que, pour tout couple possible (pk, sk) généré par $\mathbf{G}()$ et pour tout $x \in \mathcal{X}$, on a : $\mathbf{PermInv}(sk, \mathbf{Perm}(pk, x)) = x$.

Définition 2. (Schéma de signature)

Un schéma de signature est définie par un triplet d'algorithmes $(\mathbf{Gen}, \mathbf{Sign}, \mathbf{Verif})$ tels que :

1. **Gen**(1^k) est un algorithme probabiliste qui, étant donné 1^k, génère un couple (pk, sk), où pk est une clé publique et sk est une clé secrète.
2. **Sign** est un algorithme qui, étant donnés un message m à signer et la clé secrète sk, renvoie la signature $y = \mathbf{Sign}(sk, m)$. Cet algorithme peut être probabiliste.
3. **Verif** est un algorithme qui, étant donnés un message m, une signature y et la clé publique pk, renvoie un bit. Ce bit est égal à 1 si la signature est acceptée, 0 sinon. On a nécessairement que si $y = \mathbf{Sign}(sk, m)$, alors $\mathbf{Verif}(pk, m, y) = 1$.

0.2.2 Protocole général

Dans un protocole général, le schéma de signature utilise une fonction de permutation à sens unique définie comme dans la définition 1. De plus, il utilise une fonction de hachage **H**. Cependant, nous nous plaçons dans un modèle d'oracle aléatoire, nous devons donc voir cette fonction comme un oracle aléatoire $\mathcal{O}$. Nous avons alors les étapes suivantes :

— Génération des clés : on génère les clés publiques et privées pk et sk avec l'algorithme **G**.

— Génération de la signature : on hache le message à signer, puis on utilise l'algorithme de permutation inverse.

1. $h \leftarrow \mathcal{O}(m)$
2. $y \leftarrow \mathbf{PermInv}(sk, h)$

On a donc $\mathbf{Sign}(sk, m) = y$.

— Vérification de la signature : on utilise l'algorithme de permutation et on vérifie que le résultat est bien égal au haché de m.

1. $h \leftarrow \mathcal{O}(m)$
2. $h' \leftarrow \mathbf{Perm}(pk, y)$
3. si $h' == h$ ACCEPTE ($\mathbf{Verif}(pk, m, y) = 1$), sinon REFUSE ($\mathbf{Verif}(pk, m, y) = 0$)

Ici, puisque nous travaillons dans $\mathbb{Z}/n\mathbb{Z}$, il est nécessaire que pour tout message m à hacher, $\mathcal{O}(m) \in \mathbb{Z}/n\mathbb{Z}$. Cependant, cela signifie que l'espace d'arrivée de notre fonction de hachage dépend de n qui est différent pour chaque clé publique. Pour modifier cela, il suffit de borner l'espace d'arrivée par l'ensemble $\{1, \ldots, 2^{2l-2}\}$ qui est inclus dans $\mathbb{Z}/n\mathbb{Z}$.

0.2.3 Protocole utilisant RSA

Notre fonction de permutation à sens unique va être RSA. On a alors $\mathbf{G}_{RSA}(l, e) = (pk, sk)$ avec $pk = (n, e)$ où $n = pq$, p et q étant deux nombres premiers codés sur l bits, et $sk = (p, q, d)$ où $ed \equiv 1[\varphi(n)]$. La permutation facile devient alors $\mathbf{Perm}_{RSA}(pk, x) \equiv x^e[n]$ et la permutation inverse est obtenue par $\mathbf{PermInv}_{RSA}(sk, y) \equiv y^d[n]$.

Le schéma de signature suit les étapes suivantes :

— Génération des clés : on génère les clés publiques et privées $pk = (n, e)$ et $sk = (p, q, d)$ avec l'algorithme $\mathbf{G}_{RSA}$.

— Génération de la signature : on hache le message à signer, puis on utilise l'algorithme de permutation inverse.

1. $h \leftarrow \mathcal{O}(m)$
2. $y \leftarrow \mathbf{PermInv}_{RSA}(sk, h) \equiv h^d[n]$

On a donc $\mathbf{Sign}(sk, m) = y \equiv h^d \equiv \mathcal{O}(m)^d[n]$.

— Vérification de la signature : on utilise l'algorithme de permutation et on vérifie que le résultat est bien égal au haché de m.

1. $h \leftarrow \mathcal{O}(m)$
2. $h' \leftarrow \mathbf{Perm}_{RSA}(pk, y) \equiv y^e[n]$
3. si $h' == h$ ACCEPTE ($\mathbf{Verif}(pk, m, y) = 1$), sinon REFUSE ($\mathbf{Verif}(pk, m, y) = 0$)

0.2.4 Propriétés

0.2.4.1 Schéma de signature unique

Ce schéma de signature est un schéma de signature unique *i.e.*, étant donné une clé publique pk, chaque message m a une unique signature y qui sera validée par la vérification de signature pour m. En effet, puisque la fonction de hachage est modélisée par un oracle aléatoire, nous savons que tout message m est relié à un unique haché h. Ce message ne peut donc avoir qu'une unique signature. Ainsi, ce protocole n'est pas sémantiquement sûr.

0.2.4.2 Dépendance à la fonction de hachage

La sécurité de la signature dépend nécessairement de la fonction de hachage. Sans cette fonction de hachage, il est facile de prouver que la signature n'est pas sécurisée. En effet, supposons que nous ne hachons pas le message m, nous savons alors que ce message est signé directement comme $y = \mathbf{PermInv}(sk, m)$.

Ainsi, bien qu'un attaquant ne connaisse pas la clé secrète sk, il lui suffit de choisir aléatoirement une signature $y \in \mathcal{X}$. Il va ensuite calculer $m = \mathbf{Perm}(pk, y)$, les algorithmes étant connus de tous. Le couple (m, y) vérifie bien $\mathbf{Verif}(pk, m, y) = 1$. En effet, ici la vérification consiste simplement à s'assurer que $m == \mathbf{Perm}(pk, y)$, ce qui est le cas puisque l'attaquant a choisi m en utilisant cette même fonction. Or l'attaquant n'a pas eu besoin de demander la signature de m. Il a donc réussi à forger une signature sans avoir besoin d'algorithmes complexes ou d'une grande puissance de calcul.

Il est donc essentiel de hacher le message m avant de calculer sa signature pour éviter les signatures forgées.

0.3 Sécurité dans le modèle d'oracle aléatoire

Définition 3. (Sécurité exacte)

Soit y tiré aléatoirement dans $\mathbb{Z}_N^*$.

On dit qu'une permutation à sens unique f est (t', e')-sûre si un attaquant peut réussir à retrouver $f^{-1}(y)$ en un temps $t'(k)$, avec une probabilité $e'(k)$.

Ces valeurs peuvent être obtenues selon la résistance perçue de la permutation P aux attaques cryptanalytiques.

Définition 4. (Sécurité d'une fonction de signature)

On dit qu'une fonction de signature est (t, q_h, q_s, e)-sûre si un attaquant peut réussir à forger la signature d'un nouveau message en un temps $t(k)$ avec probabilité $e(k)$, s'il a accès à $q_s(k)$ paires message/signature et peut faire appel $q_h(k)$ fois à la fonction (idéale) de hachage.

Théorème 1. Soit f une permutation à sens unique (t', e')-sûre de complexité C.
Alors la fonction de signature FDH associée (noté f-FDH) est (t, e)-sûre, avec

$$t = t' - (q_h + q_s + 1).C$$
$$e = e'.(q_s + q_h)$$

Nous avons donc $t \approx t'$ et $e \approx e'.q_h$, c'est-à-dire que f-FDH perd un facteur q_h en terme de sécurité exacte par rapport à la sécurité de f.

proof. Soit F un algorithme qui (t, q_h, q_s, e)-casse f-FDH.
Alors l'algorithme I suivant (t', e')-casse f :

On a f et y.
On cherche x tel que $x = f^{-1}(y)$.

I commence par lancer F sur la fonction f et va prendre le role d'oracle aléatoire pour F.
F va alors faire des requêtes de signature et de hachage par oracle à I qui devra alors lui donner les réponses à ces requêtes.
Si F fait une requête de signature d'un message m avant d'avoir fait une requête du haché de m, alors I fera d'abord comme si F lui avait fait une requête du haché de m, puis il traitera la requête de signature.
On peut alors dire, sans perte de généralité, que F fait toutes ses requêtes d'un haché avant de demander la signature, on a alors au plus $q_h + q_s + 1$ requêtes.

On pose $q = q_h + q_s$, et on tire aléatoirement j dans $\{0, ..., q\}$.
Soit M_i le ieme message pour lequel F a fait une requête de hachage.
Alors si $i = j$ alors I renvoie à F y en tant que haché de M_i et associe y à M_i.
Sinon, il choisi r_i aléatoirement dans l'ensemble des pré-images possibles pour f, renvoie $f(r_i)$ à I en tant que haché de M_i, et associe $f(r_i)$ à M_i.
Alors, lorsque F fera une requête de signature d'un message m, I aura déjà associé ce message à un $f(r_i)$ et pourra donc lui renvoyer le r_i correspondant.
Lorsque F s'arrête, il a alors déduit une signature x d'un message m avec une probabilité de e.
Alors il existe un i tel que $M_i = m$ et, si la signature déduite est valide, avec une probabilité de $\frac{1}{q}$, on a que $i = j$ et $x = f^{-1}(y_i) = f^{-1}(y)$ était le bon inverse pour f.

On peut obtenir une meilleure sécurité si on utilise RSA comme permutation à sens unique.

Théorème 2. On considère que RSA est (t', e')-sûr.
Alors la fonction de signature FDH associée (noté RSA-FDH) est (t, e)-sûre, avec

$$t = t' - (q_h + q_s + 1).\mathcal{O}(k^3)$$
$$e = e'.\frac{1}{(1 - \frac{1}{q_s+1})^{q_s+1}}.q_s$$

proof. Soit F un algorithme qui (t, q_h, q_s, e)-casse RSA-FDH.
Alors l'algorithme I suivant (t', e')-casse RSA :

On a N, e la clef publique et y le message à casser.
On cherche x tel que $x = f^{-1}(y)$ avec f la fonction RSA définie par (N, e).

I commence par lancer F sur f et va prendre le role d'oracle aléatoire pour F.
F va alors faire des requêtes de signature et de hachés par oracle à I qui devra alors lui donner les réponses à ces requêtes.
Si F fait une requête de signature d'un message m avant d'avoir fait une requête du haché de m, alors I fera d'abord comme si F lui avait fait une requête du haché de m, puis il traitera la requête de signature.
On peut alors dire, sans perte de généralité, que F fait toutes ses requêtes d'un haché avant de demander la signature, on a alors au plus $q_h + q_s + 1$ requêtes.

On pose p une probabilité fixée que l'on déterminera plus tard.
Soit M_i le ieme message pour lequel F a fait une requête de haché.
Alors I choisit $r_i \in \mathbb{Z}_N^*$ et renvoie à F $h_i = r_i^e mod N$ en tant que haché de M_i avec une probabilité p ou bien $h_i = y.r_i^e mod N$ avec une probabilité $1 - p$, et associe h_i à M_i.
Alors, lorsque F fera une requête de signature d'un message m, I aura déjà associé ce message à un h_i.
Si $h_i = r_i^e mod N$ il renvoie r_i comme signature de M_i , sinon il s'arrête et l'inversion n'a pas réussi.
Lorsque F s'arrête, il a alors déduit une signature x d'un message m avec une probabilité de e.
Alors il existe un i tel que $M_i = m$ et, si le h_i associé est de la forme $h_i = y.r_i^e mod N$, on a que $x = h_i^d = y^d.r_i mod N$ donc que $y^d = \frac{r_i}{x} mod N$.
Sinon le processus s'arrête et l'inversion n'a pas réussi à inverser y.

La probabilité que I réussisse à répondre à toutes les requêtes de signature est au moins égale à p^{q_s}, puis I renvoie l'inverse de y avec une probabilité de $1 - p$.

Donc I renvoie l'inverse de y avec une probabilité d'au moins $\alpha(p) = p^{q_s}.(1 - p)$.
Or cette fonction $\alpha(p)$ est maximale pour $p_{max} = 1 - \frac{1}{q_s+1}$ et dans ce cas,

$$\alpha(p_{max}) = \frac{1}{q_s}(1 - \frac{1}{q_s + 1})^{q_s+1}$$

.
D'où

$$e = e'.\frac{1}{(1 - \frac{1}{q_s+1})^{q_s+1}}.q_s$$

Nous avons alors une sécurité exacte qui ne dépend plus du nombre d'appel à un oracle aléatoire, qui n ?était limitée que par la puissance de calcul de la part de l'attaquant, mais qui dépend seulement du nombre d'appel à une signature qui lui, peut être limité par le signataire.
Cette propriété peut être étendue à un certain nombre d'autres fonctions de permutation à sens unique (voir [DR02]).
Il existe cependant des algorithmes ayant une meilleure sécurité exacte (voir PSS et PFDH).

Conclusion

Ce schéma de signature ne peut pas être utilisé en pratique à cause de l'utilisation d'un oracle aléatoire. Cependant, en modélisant la fonction de hachage par un oracle aléatoire, ce protocole a permis de prouver la sécurité du schéma de signature RSA dans le modèle d'oracle aléatoire. Il suffit alors de choisir une fonction de hachage selon les critères de sécurité exigés par les utilisateurs.

Bibliographie

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical : A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures - how to sign with RSA and rabin. In *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 399–416, 1996.

[BS17] Dan Boneh and Victor Shoup. In *A Graduate Course in Applied Cryptography*, 2017.

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 229–235, 2000.

[DR02] Yevgeniy Dodis and Leonid Reyzin. On the power of claw-free permutations. In *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, pages 55–73, 2002.