Implémentation de DSA

Thomas Prest

7 mai 2010

Table des matières

1 Introduction

Pour cette implémentation de DSA, j'ai décidé de programmer en C en m'aidant de la bibliothèque GMP. Le projet comporte deux parties :

1. La fonction de hachage est SHA-256. Elle est programmée dans le fichier `SHA.c`.
2. Le standard DSA utilisé est DSA-(2048,256). Il est programmé dans le fichier `DSA.c`, qui contient également l'interface graphique.

Voici comment utiliser le programme en ligne de commande :

1. Placez-vous dans le répertoire courant et compiler en tapant `make`
2. Lancez le programme en tapant `./hello`
3. Suivez les instructions du programme

2 SHA

Parmi les différentes fonctions de hachage existantes, j'ai choisi SHA-256, et ce pour trois raisons :

1. Il s'agit de l'un des standards actuels spécifiés par le NIST (aux côtés de SHA-224, SHA-384 et SHA-512).
2. Ses mots peuvent être représentés en C par des `long unsigned int` (alors que SHA-512 nécessite l'utilisation de la bibliothèque GMP ou de types non standards tels que les `long long int`).
3. DSA ne gère pas les hachés de taille supérieure à 256 bits, ou tout du moins il n'en est pas fait mention dans les publications du NIST.

La fonction `SHA256` procède ainsi :

1. elle prend en entrée une chaîne de caractères `message`.
2. elle crée une copie de `message` dans le fichier `temp.temp`, fichier qui est ensuite "complété" pour avoir une taille multiple de 64 octets.
3. elle traite `temp.temp` et renvoie le haché sous la forme d'un tableau de `long unsigned int` à 8 entrées.

3 DSA

J'ai fait appel à la bibliothèque GMP pour gérer les grands nombres et effectuer les opérations telles que l'exponentiation modulaire ou les tests de primalité.

Génération de clefs

Concernant les tailles de clefs, j'ai choisi $(L, N) = (2048, 256)$. Selon le NIST, ces paramètres fournissent 112 bits de sécurité et ne seront pas obsolètes avant 2030(voir NIST Special Publication 800-57).
Pour la génération de p et q, je n'ai pas suivie la procédure suggérée par le NIST, j'ai développé ma propre méthode. Les nombres p et q obtenus ont une forme très particulière, mais elle est rapide, facile à programmer et p et q vérifient toutes les conditions énoncées.

`generation_p_q :`

1. `Posons` $K = \prod_{p_i \leq 1283} p_i \approx 3.05 \times 10^{537} \approx 2^{1786}$.
2. `Sélectionnons` q `un entier premier au hasard compris entre` 2^{255} `et` 2^{256}.
3. `Posons` $p = K \times q$ `et effectuons` $p := 2p$ `tant que` p `n'a pas 2048 bits. Effectuons ensuite` $p := p + 1$.
4. `Si` p `est premier, terminons l'algorithme. Sinon, reprenons à l'étape 2.`

`generation_p_q` boucle donc tant que p n'est pas premier. Le choix de K est ici ce qui fait toute la différence. Son avantage par rapport à un choix aléatoire de K est qu'il nous assure que p ne sera divisible par aucun nombre premier compris entre 2 et 1283, ce qui multiplie (en théorie) le temps de calcul par un facteur 0.078.

Remarquons que cet algorithme renvoie systématiquement des couples de la forme $(q, p) = (q, 2^d Kq + 1)$, ce qui pourrait constituer une faiblesse potentielle.

Opérations coûteuses en temps.
Il s'agit des tests de primalité, et plus précisément du test de Miller-Rabin, que j'ai pour chaque p appliqué 10 fois, ce qui nous assure qu'il soit premier avec une probabilité de 99.9999%.

Signature et vérification

Il n'y a pas de remarque particulière à faire sur l'élaboration de ces fonctions, je n'ai fait qu'appliquer à la lettre les procédures décrites par le NIST et Wikipedia.

Opérations coûteuses en temps
Il s'agit du hachage d'un part, ce qui est normal puisqu'il s'agit d'une opération complexe. Remarquons que ce temps est proportionnel à la taille du fichier à signer/vérifier. Plus précisément, les mesures nous permettent d'estimer un temps asymptotique de hachage d'environ 6,25 mo/s.
D'autre part il s'agit de l'exponentiation modulaire, ce qui est également normal étant donné la taille énorme des bases (2048 bits) et des exposants (256 bits).

4 Mesures expérimentales et conclusion

Les mesures ont été effectuées sous Linux Ubuntu 9.10, sur un ordinateur portable tournant avec un processeur Intel Centrino Core 2 Duo cadencé à 2Ghz, et doté de 2 Go de mémoire vive.

Voici les valeurs expérimentales concernant les données suivantes :

1. **Temps moyen de génération de clefs DSA :** 4.29s
2. **Nombre moyen de bouclages avant de générer une clef DSA :** 110.3
3. **Vitesse de hachage d'un fichier :** 6.25 Mo/s
4. **Temps de signature d'un fichier :** 0.0076 + 0,16×T, où T est la taille du fichier à signer en Mo
5. **Temps de vérification d'un fichier :** 0.009 + 0,16×T, où T est la taille du fichier à vérifier en Mo

Cela nous permet d'estimer le temps nécessaire :

1. **Pour signer (ou vérifier) un million de fichiers de 10ko : 2h34**
2. **Pour signer (ou vérifier) un million de fichiers de 100ko : 6h43**

Il doit être possible d'améliorer encore les performances en changeant les options d'optimisation dans le Makefile. Cependant, je ne l'ai pas fait, par souci de portabilité.