Université de Limoges

Développement Logiciel Cryptographique

TP n° 4

Dans ce TP vous allez développer les différentes couches algorithmiques permettant de coder un schéma de signature/vérification de type ECDSA sur une courbe elliptique définie sur un corps de grande caractéristique.

On rappelle la hiérarchie de couches algorithmiques suivante :

— Au niveau le plus bas, à partir des opérations sur le corps de base [1], on définit les opérations d'addition de deux points et de doublement d'un point sur la courbe elliptique.

— À partir des fonctions d'addition et de doublement, on définit la multiplication scalaire d'un point P par un entier k comme valant $k \cdot P = P + P + \ldots + P$ (le terme P apparaissant k fois).

— Au niveau le plus haut, la multiplication scalaire d'un point par un entier est utilisée dans le schéma ECDSA aussi bien pour le calcul d'une signature que pour sa vérification.

Dans le suite du TP nous utiliserons une courbe elliptique proposée par le NIST définie sur $\mathbb{GF}(p)$ et ayant pour équation

$$y^2 \equiv x^3 - 3x + b \pmod{p}$$

avec les valeurs suivantes :

$$p = \texttt{6277101735386680763835789423207666416083908700390324961279}_{10}$$
$$b = \texttt{64210519e59c80e70fa7e9ab72243049feb8deecc146b9b1}_{16}$$

Le point de base $G = (G_x, G_y)$ est défini par :

$$G_x = \texttt{188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012}_{16}$$
$$G_y = \texttt{07192b95ffc8da78631011ed6b24cdd573f977a11e794811}_{16}$$

Tous les calculs se font donc dans le sous-groupe $\langle G \rangle$ dont l'ordre n a pour taille 192 bits et pour valeur :

$n = \texttt{6277101735386680763835789423176059013767194773182842284081}_{10}$.

1. Ici le corps de base est $\mathbb{GF}(p)$ donc les opérations sur ce corps sont simplement les opérations modulo p.

1 Addition et Doublement

Nous adoptons le système de coordonnées affines [2]. Le point à l'infini $\mathcal{O}$ sera représenté par un couple de coordonnées nulles : $\mathcal{O} = (0, 0)$. Ceci est un encodage purement arbitraire d'un point qui ne peut formellement pas être représenté comme les autres points puisqu'il faudrait attribuer une valeur infinie à son ordonnée. Dans nos implémentations, toute interaction avec le point $\mathcal{O}$ devra être traitée comme un cas particulier.

On définira un type `point` comme étant une structure contenant deux valeurs x et y de type `mpz_t`.

1. Écrivez une fonction ayant pour prototype

   ```
   int sur_courbe(point P, mpz_t p, mpz_t a, mpz_t b);
   ```

   qui renvoie 1 si le point P appartient à la courbe elliptique définie par les paramètres p, a et b, et 0 sinon.

2. Écrivez une fonction ayant pour prototype

   ```
   int addition(point *R, point P, point Q,
                mpz_t p, mpz_t a, mpz_t b);
   ```

   qui calcule le point $R = P + Q$. Votre fonction devra vérifier que les points P et Q appartiennent à la courbe et vérifient les conditions pour pouvoir être additionnés. Elle renverra 0 si l'addition est possible et -1 sinon.

3. Écrivez une fonction ayant pour prototype

   ```
   int doublement(point *R, point P,
                  mpz_t p, mpz_t a, mpz_t b);
   ```

   qui calcule le point $R = 2 \cdot P$. Votre fonction devra vérifier que le point P appartient à la courbe. Elle renverra 0 si le doublement est possible et -1 sinon.

2 Multiplication Scalaire

1. En adaptant au contexte des courbes elliptiques la méthode d'exponentiation modulaire de votre choix, écrivez une fonction ayant pour prototype

   ```
   int multiple(point *R, point P, mpz_t k,
                mpz_t p, mpz_t a, mpz_t b);
   ```

2. Dans ce système un point est simplement représenté par ses deux coordonnées x et y.

qui calcule le point $R = k \cdot P$. Ici aussi vous vérifierez que le point P appartient à la courbe et renverrez -1 si tel n'est pas le cas.

2. Écrivez un programme qui lit les différents paramètres systèmes [3] depuis un fichier texte, qui prend en entrée un entier k sur la ligne de commande, et qui calcule le point $k \cdot G$ de deux manières différentes de sorte à pouvoir repérer d'éventuelles erreurs dans les fonctions d'addition et de doublement.
   Par exemple si k est pair vous pourrez d'une part calculer $k \cdot G$ comme le double de $(k/2) \cdot G$, d'autre part comme $((k-1) \cdot G) + G$. Bien entendu, les deux calculs doivent donner le même résultat.
   Vous trouverez par vous-même quoi faire dans le cas où k est impair.

3 Signature et Vérification ECDSA

3.1 Génération d'une paire de clés

Écrivez un programme qui lit les différents paramètres systèmes depuis un fichier texte, et génère aléatoirement une clé publique et sa clé privée associée, puis les affiche en hexadécimal à l'écran sous la forme :

```
d   = 0x...
Q_x = 0x...
Q_y = 0x...
```

où $Q = (Q_x, Q_y) = d \cdot G$.

La paire de clé sera stockée dans un fichier par redirection de *stdout*.

3.2 Signature ECDSA

Écrivez un programme qui génère la signature ECDSA d'un message. Le nom du fichier à signer sera lu sur la ligne de commande, alors que les différents paramètres systèmes et la partie privée de la clé seront lus depuis deux fichiers différents. Vous utiliserez la fonction de hachage de votre choix.

3.3 Vérification ECDSA

Écrivez un programme qui vérifie la signature ECDSA d'un message. Le nom du fichier dont il faut vérifier la signature sera lu sur la ligne de commande, alors que les différents paramètres systèmes, la partie publique de la clé et la valeur de la signature seront lus depuis trois fichiers différents.

3. La courbe, le point de base et son ordre.