TP2 - ECC et ECDSA

22 Septembre 2021

L'objectif de ces exercices est de vous familiariser avec la cryptographie basée sur les courbes elliptiques (ECC) et d'illustrer le concept avec l'algorithme de signature numérique ECDSA.

> Vous êtes libre de choisir le langage de programmation de votre choix, mais `sage` est fortement recommandé puisqu'il supporte nativement les opérations sur les courbes elliptiques (vous pouvez définir un objet `EllipticCurve`) et de nombreuses méthodes utiles.

Exercice 1 : Manipulation de points sur une courbe elliptique

Rappel Une courbe elliptique est un groupe définit par une **équation** satisfaisant certaines propriétés ($y^2 = x^3 - 3x + 5$ par exemple) **et un corps** (les réels ou les entiers modulo un nombre premier p par exemple). Les éléments de ce groupe sont les points appartenant à cette courbe, c'est à dire les points dont les coordonnées satisfont l'équation sur le corps.

Ici, la loi d'opération entre les éléments du groupe est **additive**, ce qui signifie qu'on peut additionner deux points pour en obtenir un troisième, toujours sur la courbe. La loi d'addition est définie de telle sorte à ce que cette propriété soit satisfaite. [1].

Si la courbe est définie sur un corps fini, le groupe formé par les points est **cyclique**. De ce fait, chaque élément a un **ordre fini** : si un point P est d'ordre k, alors $k \times P = \mathcal{O}$ (ici on a une multiplication et non une exponentiation car la loi est additive et non multiplicative).

En cryptographie, on manipule des points d'ordre très élevés pour assurer un niveau de sécurité suffisant. En effet, la plupart de la cryptographie basée sur les courbes elliptiques repose sur la difficulté du *Elliptic Curve Discrete Logarithm Problem* (ECDLP) : "Étant donné un point G publique, d'ordre ℓ suffisamment grand, et un point aléatoire Q, il est infaisable de retrouver k tel que $Q = k \times G$." Pour résoudre ce problème on connait des algorithme génériques (*i.e.* qui fonctionne sur tous les cas) permettant de calculer le logarithme discret en $\mathcal{O}(\sqrt{n})$, il est donc important de choisir une courbe dont l'ordre est suffisamment grand pour garantir un niveau de sécurité suffisant.

1. Pour commencer, réalisez certaines opérations simples sur `sage` :
   (a) Créez une courbe E_1 correspondant à l'équation $y^2 = x^3 - 3x + 5$ sur $\mathbb{Z}/29\mathbb{Z}$.
   (b) Générez un point aléatoire sur cette courbe à l'aide de la méthode `random_point`.
   (c) Vérifier l'ordre de la courbe à l'aide de la méthode `order`. Faites de même pour l'ordre de différents points. Que constatez-vous ?
2. Implémentez la multiplication scalaire sur cette courbe. [2]
   (a) Implémentez une fonction `add_points` qui prend en paramètre deux points distincts P_1 et P_2 ainsi que la courbe (ou ses paramètres) et retourne `Q=`P_1`+`P_2. Pensez à gérer le point infini.
   (b) Implémentez une fonction `double_points` qui prend en paramètre un point `P` et la courbe (ou ses paramètres) et retourne `Q=P+P`. Pensez à gérer le point infini.

1. `https://www.math.brown.edu/~jhs/Presentations/WyomingEllipticCurve.pdf`
2. `https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication#Point_operations`

(c) Implémentez à l'aide des deux fonctions précédentes la multiplication scalaire avec l'algorithme *double-and-add*[3] qui consiste à décomposer le scalaire en bits, et pour chacun des bits, doubler le point, et additionner le points de base uniquement si le bit vaut 1.

3. Sans utiliser la méthode `order`, écrire une fonction `get_point_order` qui prend en entrée un point, et qui renvoie son ordre (de manière très simple).
4. Listez tous les points sur la courbe E_1 et E_2 (définie par l'équation $y^2 = x^3 - 2x + 20$ sur $\mathbb{Z}/29\mathbb{Z}$).
   (a) Écrire une fonction `is_point_x` qui prend en entrée une coordonnée entière x (on peut supposer $x \in \mathbb{Z}/29\mathbb{Z}$) et la courbe (ou ses paramètres), et qui retourne `True` si cet entier correspond à la coordonnée x d'un point sur la courbe, et `False` sinon.
   La méthode `is_square` permet de vérifier si un entier est un résidu quadratique, c'est-à-dire s'il est le carré d'un autre élément.
   (b) À l'aide de la méthode précédente, trouvez tous les points présents sur cette courbe, et vérifier l'ordre de chacun de ces points. Que constatez-vous ?
5. Considérons maintenant la courbe E_3 définie par $y^2 = x^3 - 3x + 6$ sur les entiers modulo

$$p = 5164669856444950218363050899868468345 3$$

Cette courbe comporte $\ell = 516466985644495021889250598977071334 41$ points (elle est donc d'ordre ℓ).
   (a) Si on prend un point aléatoire sur cette courbe, de quel ordre peut-il être ?
   Indice : regardez l'ordre de la courbe.
   (b) Soit

$$P = (586639159201118869273272914240784164 4, 500829927867318648830632065664935600 50)$$

   Quel est l'ordre de P (n'essayez pas de trouver à la main) ?
   (c) Que pensez-vous de la sécurité apportée par l'usage du point P en tant que générateur si on se base sur la difficulté de ECDLP ? Justifiez.
   (d) Supposons que P_2 soit d'ordre $order(P_2) = \ell$. Est-ce suffisant ?

Exercice 2 : ECDSA

1. Expliquez (sans plagier) ce qu'est ECDSA, sans oublier de dire ce qu'est une signature numérique.
2. Implémentez ECDSA (en `sage` de préférence, pour vous faciliter la tâche).
3. Est-ce que le nonce (communément noté k) utilisé durant la signature est considéré sensible/secret ? Justifier.
   La réponse à cette question est apportée par l'exercice suivant, mais essayez d'identifier les problèmes potentiels en cas de mauvais usage du nonce avant d'avancer dans le TP.

Exercice 3 : Attaques sur ECDSA (Bonus)

Dans un schéma de signature, le message n'est pas forcement considéré secret. Ici, on va considérer que l'attaquant connait le message à chiffrer.
Toutes les signatures sont générées en utilisant la courbe P-256 du NIST (dont les paramètres sont donnés) et la fonction de hachage SHA256.

1. Supposez qu'un utilisateur signe un message et le transmette avec le nonce utilisé durant la signature.
   (a) Quelles informations l'attaquant est-il en mesure de retrouver ? Souvenez-vous que le message m, la signature (r, s) et le nonce k sont connus.

3. `https://wikipedia.org/wiki/Elliptic_curve_point_multiplication#Point_multiplication/`

(b) Implémentez l'attaque avec les valeurs données.

2. Cette fois, l'utilisateur a gardé le nonce secret, mais l'a utilisé pour signer plusieurs messages différents (deux en l'occurence). Ce problème est connu et a causé de nombreux problèmes dans des produits déployés, notamment chez la PS3 [4, 5], compromettant entièrement la console.
   (a) Étant donné deux signatures, comment savoir si le même nonce est utilisé ?
   (b) Comment retrouver la clé privée à partir de ces deux signatures, les messages correspondant et la clé publique ?
   (c) Implémentez l'attaque sur les valeurs fournies.

4. `https://media.ccc.de/v/27c3-4087-en-console_hacking_2010`
5. `https://www.bbc.com/news/technology-12116051`