TP - ECC et multiplication par un scalaire

8 octobre 2022

Le TP d'aujourd'hui est un examen. Un rendu est attendu : une archive contenant tout votre code. N'oubliez pas de commenter votre code.

Le rendu est à envoyer par mail pour le **dimanche 9 octobre, 23h59**, en incluant "[BCS-tp-note]" dans l'objet, à l'adresse `andrea.lesavourey@irisa.fr`.

> L'objectif est d'implanter **en C** les opérations sur les courbes elliptiques nécessaires à la cryptographie.

Exercice 1 : Manipulation de points sur une courbe elliptique

Implantez les opérations nécessaires à la cryptographie basée sur les courbes elliptiques. Vous pouvez par exemple utiliser la librairie GMP pour manipuler des entiers de taille arbitraire.

Afin de rendre des étapes d'exponentiation plus efficaces, plusieurs méthodes ont été développées [1] [2]. L'objectif des exercices suivants est d'en explorer deux d'entre elles.

Exercice 2 : Fenêtre fixe ou méthode 2^k-aire

La méthode de la fenêtre fixe ou 2^k-aire consiste à calculer à l'avance certaines puissances de l'élément de base g qu'on pourra réutiliser durant le calcul d'éléments de la forme g^e.

1. Adaptez cette méthode décrite dans l'algorithme 1 à la multiplication d'un point P d'une courbe elliptique par un scalaire.
2. Trouvez la taille de la fenêtre k optimale dans le cas de la courbe P256 du NIST.

Notations : Si un entier vérifie $m = 2^s u$ avec $\gcd(s, u) = 1$. on notera dans la suite $\sigma : m \mapsto (s, u)$. De plus, étant donnée un entier de base b, on notera $(n_{l-1} \dots n_1 n_0)_b$ la décomposition de l'entier n en base b.

```
# NIST's P-256 parameters
p = 2**256 - 2**224 + 2**192 + 2**96 - 1 #  prime defining the field
# y^ = x^3 + ax + b
a = -3
b = 41058363725152142129326129780047268409114441015993725554835256314039467401291
# G = (Gx, Gy), nG = 0
Gx = 0x6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296
Gy = 0x4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5
n = 115792089210356248762697446949407573529996955224135760342422259061068512044369
```

1. https://en.wikipedia.org/wiki/Exponentiation_by_squaring
2. http://koclab.cs.ucsb.edu/teaching/ecc/eccPapers/Doche-ch09.pdf

Algorithm 1

Require: An element $g \in G$, a parameter $k \geq 1$, an integer $n = (n_{l-1} \dots n_1 n_0)_{2^k}$ and the precomputed values $g^3, g^5, \dots, g^{2^k-1}$.
Ensure: $h = g^n$
  $h \leftarrow 1; i \leftarrow l - 1$
  while $i \geq 0$ **do**
    $(s, u) \leftarrow \sigma(n_i)$
    for $j = 1$ **to** $k - s$ **do**
      $h \leftarrow h^2$
    end for
    $h \leftarrow hg^u$
    for $j = 1$ **to** s **do**
      $h \leftarrow h^2$
    end for
    $i \leftarrow i - 1$
  end while

Exercice 3 : Forme non adjacente

La forme non adjacente (NAF) d'un entier k est une représentation signée équivalente à la décomposition binaire [3]. Elle permet notamment de remplacer certaines additions par des soustractions lors de la multiplication d'un point P par le scalaire k.

```
####  Exemples de formes non adjacentes ####
# 7
(0 1 1 1)_2 = 4 + 2 + 1 = 7          # décomposition binaire
(1 0 0 -1)_2 = 8 - 1 = 7             # NAF
# 23
(1 0 1 1 1)_2 = 16 + 4 + 2 + 1 = 23  # décomposition binaire
(1 0 -1 0 0 -1)_2 = 32 - 8 - 1 = 23  # NAF
```

1. Codez des fonctions permettant de passer d'une décomposition binaire d'un entier à sa NAF et vice-versa.
2. Implantez un algorithme de multiplication d'un point par un scalaire qui utilise la NAF.

3. https://en.wikipedia.org/wiki/Non-adjacent_form

	Représentation affine	Représentation projective
Equation de la courbe	$E : y^2 = x^3 + ax + b$	$E : zy^2 = x^3 + axz^2 + bz^3$
Coordonnées d'un point	$P(x, y)$	$P(x, y, z)$
Coord. affines $\rightarrow$ proj	$(x, y, 1)$	(x, y)
Coord. affines $\leftarrow$ proj	$(x/z, y/z)$	(x, y, z)
0_E = pt à l'infini	–	$(0, 1, 0)$
$-P$	$(x, -y)$	$(x, -y, 1)$
$P + Q$	formules	
$2P$	formules	
$P + 0_E$	P	
$P + (-P)$	0_E	
$2P(x, 0)$	0_E	
$P(x, y_P) + Q(x, y_Q)$	0_E	

Egalité en projectif : Pour tout $\lambda \in \mathbb{F}_p$, $(x, y, z) = (\lambda x, \lambda y, \lambda z)$ ***modulo*** p

Exemples de fonctions auxiliaires (utiles ?) :

— `is_equal`(P, Q)
— `is_zero (P)`
— `proj2affine`
— `affine2proj`