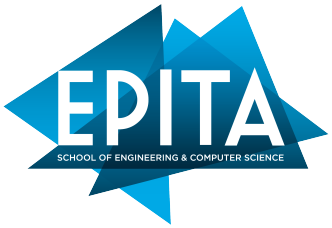
Challenge France Cybersécurité 2023
Comptes Rendus

Quentin Rataud

Avril 2023



1 Les challenges

Elliptic Addventure

```
p = 115792089210356248762697446949407573530086143415290314195533631308867097853951
a = -3
b = 41058363725152142129326129780047268409114441015993725554835256314039467401291

K = GF(p)
E = EllipticCurve([K(a), K(b)])

mid = len(flag) // 2
Ax = K(bytes_to_long(flag[:mid]))
Bx = K(bytes_to_long(flag[mid:]))

A = E.lift_x(Ax)
B = E.lift_x(Bx)

print(f"{A + B = }")
print(f"{A - B = }")
```

Elliptic Addrenaline

```
p = 2**255 - 19
a = 19298681539552699237261830834781317975544997444273427339909597334573241639236
b = 55751746669818908907645289078257140818241103727901012315294400837956729358436

K = GF(p)
E = EllipticCurve([K(a), K(b)])

mid = len(flag) // 2
Ax = K(bytes_to_long(flag[:mid]))
Bx = K(bytes_to_long(flag[mid:]))

A = E.lift_x(Ax)
B = E.lift_x(Bx)

print(f"{A + B = }")
print(f"{A - B = }")
```

Deux points A et B sont présents sur une courbe elliptique d'équation

$$(E): \quad y^2 = x^3 + ax + b \mod p$$

Dans le fichier de sortie donné au candidat, les coordonnées des points $A + B$ et $A - B$ sont indiqués. L'objectif est de retrouver les coordonnées des points A et B, qui encodent la solution des challenges.

Seules les équations des courbes et les coordonnées des points diffèrent entre les deux challenges.

2 Stratégie de résolution

Trouver $2 \cdot A$ et $2 \cdot B$ En connaissant $\nabla = A + B$ et $\Delta = A - B$, il est facile d'en déduire les coordonnées de $2 \cdot A$ et de $2 \cdot B$:

$$\begin{aligned} \nabla + \Delta &= (A + B) + (A - B) \\ &= 2 \cdot A \end{aligned}$$

$$\begin{aligned} \nabla - \Delta &= (A + B) - (A - B) \\ &= 2 \cdot B \end{aligned}$$

Cependant, retrouver A et B depuis $2 \cdot A$ et $2 \cdot B$ relève encore un peu de défi.

2.1 Elliptic Addventure

Propriétés des données Deux propriétés permettent de résoudre ce challenge sans difficulté.

- (E) est un groupe cyclique;
- L'ordre de (E) est un nombre premier.

Ces deux informations peuvent être vérifiées grâce à un court code `sage` :

```
p = 115792089210356248762697446949407573530086143415290314195533631308867097853951
o = 115792089210356248762697446949407573529996955224135760342422259061068512044369
a = -3
b = 41058363725152142129326129780047268409114441015993725554835256314039467401291

K = GF(p)
E = EllipticCurve([K(a), K(b)])

print(E.gens())
# ((38764697308493389993546589472262590866107682806682771450105924429005322578970 :
#   112597290425349970187225006888153254041358622497584092630146848080355182942680 :
#   1),)

print(E.order())
# 115792089210356248762697446949407573529996955224135760342422259061068512044369
# C'est un nombre premier !
```

Vu que `E.gens()` retourne un tuple contenant un unique élément, (E) forme un groupe cyclique. L'ordre de E est un nombre premier que nous appelerons k par la suite.

k étant premier, il possède alors un inverse modulaire modulo 2, et il est possible de calculer cet inverse :

```
k = E.order()
h = (k + 1) // 2
# 57896044605178124381348723474703786764998477612067880171211129530534256022185
```

h est l'unique nombre tel que $2 \cdot h = 1 \mod k$. En effet, $2 \cdot h = 1 + k$.

Trouver A et B Étant donné que l'ordre de (E) est un nombre premier, tous les points de la courbe en sont des générateurs. Ceci implique que pour tout point P dans E,

$$P \cdot k = O$$

Cette propriété peut être utilisée pour retrouver les valeurs de A et B très rapidement depuis les valeurs de $2 \cdot A$ et $2 \cdot B$:

$$\begin{aligned}(2 \cdot A) \cdot h &= A \cdot (2 \cdot h) \\ &= A \cdot (1 + k) \\ &= A + A \cdot k \\ &= A + O \\ &= A\end{aligned}$$

B peut être retrouvé de manière identique.

2.2 Elliptic Addrenaline

Propriétés des données Premièrement, il est intéressant de constater que (E) forme également un groupe cyclique. Cependant, cette fois-ci, son ordre n'est pas un nombre premier. Il est possible de prouver ceci et de trouver un générateur de la courbe elliptique simplement grâce à un court code `sage` :

```
p = 2**255 - 19
a = 19298681539552699237261830834781317975544997444273427339909597334573241639236
b = 55751746669818908907645289078257140818241103727901012315294400837956729358436
K = GF(p)
E = EllipticCurve([K(a), K(b)])

print(E.order())
# 57896044618658097711785492504343953926856930875039260848015607506283634007912

print(E.gens())
# ((177945032291343536359704398129492971002254894875881725683893278977547465462 :
#   176710334591119687109882960616765240366527493654242109516653296835943560300 :
#   1),)
```

Vu que `E.gens()` retourne un tuple avec un unique élément G, la courbe elliptique est cyclique et G en est un générateur.

Notons ensuite quelques propriété concernant A et B :

```
ordre = E.order()
# 57896044618658097711785492504343953926856930875039260848015607506283634007912

## Coordonnées des points données dans output.txt
nabla = E(
    36383477447355227427363222958872178861271407378911499344076860614964920782192,
    26621351750863883655273158873320913584591963316330338897549941610801666281894,
    1
) # = A + B

delta = E(
    35017143636654127615837925410012912090234292410137109973033835965781971515338,
    55888666729705323990488128732989325970476008697224551268788692630541877244410,
    1
) # = A - B

aa = nabla + delta # = 2A
bb = nabla - delta # = 2B

print(aa.order())
# 28948022309329048855892746252171976963428465437519630424007803753141817003956
```

```
print(bb.order())
# 2894802230932904885589274625217197696342846543751963042400780375314181700395 6
```

Il est intéressant de constater que l'ordre de $2 \cdot A$ et l'ordre de $2 \cdot B$ est précisément 2 fois inférieur à l'ordre de (E). Ceci signifie que ces deux points sont deux générateurs de l'unique sous-groupe $\langle 2 \cdot G \rangle$, et cela implique également que A et B étaient des générateurs de (E).

Finalement, il est important de noter que l'ordre de (E) est un multiple de 8, mais pas un multiple de 16. Notons alors k le huitième de l'ordre de E. k possède alors un inverse modulaire modulo 2, et il est possible de calculer cet inverse :

```
k = ordre // 8
h = (k + 1) // 2
# 3618502788666131106986593281521497120428558179689953803000975469142727125495
```

h est l'unique nombre tel que $2 \cdot h = 1 \mod k$. En effet, $2 \cdot h = 1 + k$.

Trouver A et B Soit G n'importe quel générateur de (E). Vu que (E) est cyclique d'ordre $8k$, on sait que $A = G \cdot x$ pour un unique x entre 0 et $8k$.

On possède $2 \cdot A = G \cdot (2x)$. Si nous multiplions ce point par h, nous obtenons

$$\begin{aligned}(2 \cdot A) \cdot h &= G \cdot (2hx) \\ &= G \cdot (x + kx)\end{aligned}$$

Si par chance x est un multiple de 8, alors $x = 8q$ pour un certain q. On obtient alors :

$$\begin{aligned}(2 \cdot A) \cdot h &= G \cdot (x + 8kq) \\ &= (G \cdot x) + (G \cdot 8kq) \\ &= A + O \\ &= A\end{aligned}$$

On pourrait donc être tenté de générer des générateurs aléatoires de (E), d'espérer que x soit un multiple de 8 et retrouver la valeur de A de cette manière. Cependant, voici une autre manière de procéder sans calculer plus de générateurs quand x n'est pas un multiple de 8.

Supposons que $x = 8q + r$ pour q, r entiers et $0 \leq r < 8$. Alors,

$$\begin{aligned}(2 \cdot A) \cdot h &= G \cdot (x + k(8q + r)) \\ &= (G \cdot x) + (G \cdot 8kq) + (G \cdot kr) \\ &= A + O + G \cdot kr \\ (2 \cdot A) \cdot h - G \cdot kr &= A\end{aligned}$$

Tester les 8 différentes valeurs de r permet de retrouver la valeur de A à coup sûr. On peut procéder de la même manière pour retrouver B.

3 Implémentation

3.1 Elliptic Addventure

```
A = aa * h
partie = long_to_bytes(int(A[0]))
print(partie.decode(), end = '')

B = bb * h
partie = long_to_bytes(int(B[0]))
print(partie.decode())

# FCSC{a0c43dbbfaac7a84b5ce7feb81d492431a69a214d768aa4383aabfd241}
```

3.2 Elliptic Addrenaline

```
for r in range(8):
    A = aa * h - G * k * r
    if A + A == aa:
        partie = long_to_bytes(int(A[0]))
        if partie.startswith(b"FCSC{"):
            print(partie.decode(), end = '')
            break

for r in range(8):
    B = bb * h - G * k * r
    if B + B == bb:
        partie = long_to_bytes(int(B[0]))
        if partie.endswith(b"}"):
            print(partie.decode())
            break

# FCSC{1f0b8b8d4ff304004126f245ee5f46d8961b60dff4b187ef6fe4f09e34}
```