FCSC 2021 : **Smealog** (Write-up)

Jakobus

Mai 2021

- **Catégorie** : Crypto (la meilleure)
- **Points** : 500 (score dynamique)
- **Résolutions** : 8
- **Description** : *Pourrez-vous résoudre ce logarithme discret ?*
- **Fichiers** : output.txt, Smealog.sage

1 Découverte de la vulnérabilité

Le fichier Smealog.sage correspond au script sage permettant de chiffrer le flag à l'aide d'une courbe elliptique. Plus particulièrement, le script :

- génère une courbe elliptique E, et affiche ses paramètres dans la console ;
- génère un point P de E et un entier s (modulo certaines restrictions dont nous parlerons par la suite), et affiche les coordonnées affines de P dans la console ;
- calcule $Q = sP$ et nous donne ses coordonnées affines dans la console ;
- chiffre le flag en AES avec pour clé le sha256 de s (converti en chaîne de caractères), et affiche le flag chiffré et l'IV utilisé.

La génération de la courbe est toutefois étrange :

```
def gen_curve(bits = 40, k = 4):
    assert bits*k >= 160, "Error: p**k must be at least 160 bits."
    p = random_prime(2 ** (bits + 1) - 1, lbound = 2 ** bits)
    R = Zmod(p**k)
    while True:
        a, b = R.random_element(), R.random_element()
        d = 4 * a ** 3 + 27 * b ** 2
        if d.is_unit():
            E  = EllipticCurve(R, [a, b])
            E_ = EllipticCurve(GF(p), [a, b])
            if E_.order().is_prime():
                return E
```

La procédure génère d'abord un premier p suffisamment grand, puis génère deux courbes de paramètres $a, b \pmod{p^4}$: l'une dans $\mathbb{Z}/p^4\mathbb{Z}$, l'autre dans $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$. Mais si l'ordre de la seconde courbe est premier, alors la procédure va retourner la première. Mais l'anneau $\mathbb{Z}/p^4\mathbb{Z}$ n'est clairement pas un corps, structures dans lesquelles est normalement faite la cryptographie sur ces courbes. C'est le fait que cet anneau ne soit pas un corps que l'on va exploiter !
On peut en effet constater, à l'aide de factordb par exemple, que le modulus n présent dans l'output de Sage (fichier output.txt) se factorise en $n = p^4$ où $p = 1614927334829$.

2 Un peu (beaucoup) de maths

Une courbe elliptique est une courbe algébrique dans un anneau commutatif R, de la forme

$$E_{a,b} : y^2z = x^3 + axz^2 + bz^3$$

dans sa version homogène, où $a, b \in R$ (le discriminant de la courbe doit être inversible dans R). On considère la représentation projective, où chaque point de $E_{a,b}(R)$ (désignant l'ensemble des points de la courbe définie sur R) est identifiée par des coordonnées $(x : y : z)$, et tels que $(x : y : z) \cong (\lambda x : \lambda y : \lambda z)$ pour tout $\lambda \in R$. On peut également, quand ceci est possible (quand z est inversible dans R), donner les coordonnées affines uniques d'un point en multipliant x et y par l'inverse de z. Sinon on peut faire de même avec y sous la même condition pour y. L'ordre d'une courbe est le nombre de points dans celle-ci, à savoir $|E_{a,b}(R)|$.
L'intérêt de telles courbes est que l'on peut définir une addition des points sur celles-ci : à tous points $P, Q \in E_{a,b}(R)$, on peut définir $G = P + Q$ où $G \in E_{a,b}(R)$ d'après des formules d'additions bien définies, que l'on peut par exemple retrouver ici. Cette addition est inversible, commutative, et possède un élément neutre, le point ("à l'infini", même si l'appellation est un peu abusive ici) $\mathcal{O} = (0 : 1 : 0)$. Les points d'une courbe elliptique dans R munie de son addition forme donc un groupe abélien. On peut définir alors l'ordre d'un point $P \in E_{a,b}(R)$, qui est en fait $k = \min\{s \mid sP = \mathcal{O}\}$. Le théorème de Lagrange nous assure que k divise l'ordre de la courbe.
Si R est un corps fini, et que l'on choisit maintenant un certain $s \in R$, un point $P \in E_{a,b}(R)$, et que l'on calcule $Q := sP$ (l'addition de P opérée s fois), alors il est généralement **très** difficile, sachant P, Q, de retrouver s, ce qui s'appelle calculer le logarithme discret de Q en base P. C'est sur ce fait que repose la cryptographie sur courbes elliptiques (ECC), en utilisant par exemple la clé s pour chiffrer des données (ce qui fait curieusement penser au challenge).

3 Exploitation de la vulnérabilité

En faisant quelques (longues) recherches, je suis tombé sur l'article suivant. Le résultat qui nous intéresse est le corollaire 18 et la proposition 19 (il est néanmoins nécessaire de lire tout ce qui précède pour les comprendre...) : on considère une courbe elliptique $E_{a,b}$ dans l'anneau $\mathbb{Z}/p^e\mathbb{Z}$ où p est premier, et un entier naturel $2 \leqslant e \leqslant 5$. On définit le morphisme de réduction $\pi : \mathbb{Z}/p^e\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$ qui consiste simplement à réduire la courbe $\mod p$ (possible car $p \mid p^e$). Alors le morphisme

$$\begin{aligned} \Phi : E_{a,b}(\mathbb{Z}/p^e\mathbb{Z}) &\to E_{a,b}(\mathbb{Z}/p\mathbb{Z}) \times \mathbb{Z}/p^{e-1}\mathbb{Z} \\ P &\mapsto \left(\pi(P), \frac{1}{p}\frac{(qP)_x}{(qP)_y}\right) \end{aligned}$$

est un isomorphisme si l'ordre de $E_{a,b}$ $q := |E_{a,b}(\mathbb{Z}/p\mathbb{Z})| \neq p$ (en fait on doit avoir $q \wedge p = 1$, mais comme d'après le "Hasse bound" - est-ce que ceci a un équivalent en français ? - $|p+1-q| \leqslant 2\sqrt{p}$...). L'article démontre que toutes les opérations sont bien définies, s'agit bien d'un morphisme, et même d'un isomorphisme sous ces conditions. En appliquant Φ à l'égalité $Q = sP$, on obtient alors

$$\left(\pi(Q), \frac{1}{p}\frac{(qQ)_x}{(qQ)_y}\right) = \Phi(Q) = \Phi(sP) = \left(\pi(sP), \frac{1}{p}\frac{(qsP)_x}{(qsP)_y}\right) = \left(s\pi(P), s\frac{1}{p}\frac{(qP)_x}{(qP)_y}\right)$$

c'est-à-dire

$$\begin{cases} \pi(Q) = s\pi(P) \\ \dfrac{1}{p}\dfrac{(qQ)_x}{(qQ)_y} \equiv s\dfrac{1}{p}\dfrac{(qP)_x}{(qP)_y} \pmod{p^{e-1}} \end{cases}$$

La deuxième équation est soluble très facilement en inversant directement $\dfrac{1}{p}\dfrac{(qP)_x}{(qP)_y}$ (bien inversible dans $\mathbb{Z}/p^{e-1}\mathbb{Z}$ d'après ce qui précède dans l'article), on note sa solution s_1. La première est relativement facile à résoudre à l'aide de Sage, il s'agit de calculer le logarithme discret de $\pi(Q)$ en base $\pi(P)$. On note sa solution $s_0 \in \mathbb{Z}/q\mathbb{Z}$. On peut alors, par le théorème des restes chinois, puisque $q \wedge p^{e-1} = 1$, reconstruire s

(mod qp^{e-1}) d'après le système (l'ordre de la courbe originale étant qp^{e-1}, ce qui s'observe directement à travers le noyau de π, le monde est bien fait)

$$\begin{cases} s \equiv s_0 \pmod{q} \\ s \equiv s_1 \pmod{p^{e-1}} \end{cases}$$

On a donc transformé notre problème **très** difficile en problème facile !
On liste alors nos différents paramètres qu'on obtient du fichier output.txt :

- $p = 1614927334829$;
- $e = 4$;
- $a = 469245061185347057653091531882370383913875080361 5$;
- $b = 511435168365576432925310624542858208458753664050 3$
- $P = (48182986080 \ldots 19975611403 : 3354976854279375 \ldots 43236550702 : 1)$;
- $Q = (627667271283 \ldots 47993984369352 : 5153262096245 \ldots 0306746041453 : 1)$

Pour calculer q, il suffit de rentrer dans Sage l'instruction suivante :

```
EllipticCurve(GF(p), [a, b]).order()
```

qui nous donne enfin $q = 1614926806643$ en une seconde. Il faut également calculer le logarithme discret de $\pi(Q)$ en base $\pi(P)$, ce que Sage fait en quelques minutes à l'aide du code suivant :

```
E = EllipticCurve(GF(p), [a, b])
piP = E(P)
piQ = E(Q)
discrete_log(piQ,piP,piP.order(),operation='+')
```

qui nous donne $s_1 = 1330461465055$. On peut enfin calculer le tant recherché

$$s = 5901549400384790567861449014099270154408538055049$$

et savourer notre victoire bien méritée en déchiffrant le flag ! :)

Voici le code python utilisé pour l'exploitation, Sage n'appréciant par les coordonnées projectives (snif) :

```

def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m


def solve_CRT(a, p, b, q):
    p1 = modinv(p, q)
    q1 = modinv(q, p)
    return (a * q * q1 + b * p * p1) % (p * q)


def affForm(P, n, a, b):
    x, y, z = P
    #print("coords: ", x, y, z)
    if egcd(z, n)[0] == 1:
```

```
        x2 = (x * modinv(z, n)) % n
        y2 = (y * modinv(z, n)) % n
        return (x2, y2, 1)
    elif egcd(y, n)[0] == 1:
        x2 = (x * modinv(y, n)) % n
        z2 = (z * modinv(y, n)) % n
        return (x2, 1, z2)
    else:
        print('pas de forme affine')

def iso(P, p, e, a, b, q):

    piP = (P[0] % p, P[1] % p, P[2] % p)

    qP = mult(q, P, p ** e, a, b)
    X = qP[0]
    print(X % p == 0)
    r = X // p
    print(piP, r)
    return(piP, r)

def isoInv(P, r, p, e, a, b, q):
    pass

def add_points(P1, P2, n, a, b):
    if P2 == (0, 1, 0): return P1
    if P1 == (0, 1, 0): return P2
    if P1 == P2:
        X1, Y1, Z1 = P1
        X3 = ((3 * X1**2 + a) ** 2 * 2 * Y1 - 2 * X1 * (8 * Y1 ** 3))
        Y3 = (3 * X1**2 + a) * (12*X1*Y1**2 - (3*X1**2 + a)**2) - 8 * Y1 ** 4
        Z3 = (8 * Y1 ** 3)
        Q = (X3 % n, Y3 % n, Z3 % n)
        return affForm(Q, n, a, b)
    X1, Y1, Z1 = P1
    X2, Y2, Z2 = P2

    if X1 == X2 and Y1 == n - Y2:
        return (0, 1, 0)
    elif X1 != X2:
        X3 = (Y2 - Y1)**2 * (X2 - X1)- (X1 + X2) * (X2 - X1)**3
        Y3 = (Y2 - Y1) * ((2 * X1 + X2) * (X2 - X1)**2 - (Y2 - Y1)**2) - Y1 * (X2 - X1)**3
        Z3 = (X2 - X1)**3
        Q = (X3 % n, Y3 % n, Z3 % n)
        return affForm(Q, n, a, b)

def mult(k, P, n, a, b):
    if k == 0:
        return (0, 1, 0)
    elif k == 1:
        return P
    elif k % 2 == 1:
        return add_points(P, mult(k - 1, P, n, a, b), n, a, b)
    else:
        return mult(k // 2, add_points(P, P, n, a, b), n, a, b)



p = 1614927334829
e = 4

n = p ** e

P = (481829860802966505139388071282510920981997561140 3,
    335497685427937548731234120185005102314323655070 2, 1)
Q = (627667271283710020684607734085408774799398436935 2,
    515326209624560602185775302799447950030674604145 3, 1)

```

```
q = 1614926806643 #pre calcule (merci quand meme Sage)

a = 469245061185347057653091531882370383913875080361 5
b = 511435168365576432925310624542858208458753664050 3

m = iso(P, p, e, a, b, q)[1]
n = iso(Q, p, e, a, b, q)[1]

s1 = 1330461465055 #pre calcule

s0 = n * modinv(m, p ** (e-1)) % p**(e-1)
print("base s0 =", s0)
s = solve_CRT(s0, p ** (e-1), s1, q)
print(s)
```

Merci aux orgas pour ce chall qui a suscité la création des meilleurs memes à mon goût !