FCSC 2022 : **Surface** (Write-up)

Jakobus

Mai 2022

- **Catégorie** : Crypto
- **Points** : 500 (score dynamique)
- **Résolutions** : 10
- **Description** : *Ce script implémente une manière exotique de générer une clé AES qui protège le flag. Pourrez-vous retrouver cette clé ?*
- **Fichiers** : `output.txt`,`T-Rex.py`

1 Découverte de la vulnérabilité

Comme l'explique très bien la description du challenge, le script `surface.py` est une simple implémentation du chiffrement du tant désiré **flag** via AES, dont le chiffré est stocké dans `output.txt`.
La génération de la clé est toutefois particulièrement non standard : elle consiste à prendre des rationnels a et b (les manipulations algébriques sont gérées par le module fractions de Python), puis de vérifier que $c = a^2 + b^2$ est le carré d'un rationnel (en vérifiant que son numérateur et son dénominateur le sont) et que $ab = 20478$. En écrivant $c = (p/q)^2$, la clé de chiffrement est alors p.
On se rend vite compte, en testant quelques valeurs et en bidouillant qu'il ne risque pas d'y avoir beaucoup de candidat suffisamment petits pour a et b, et donc d'autant moins pour p.
Le challenge revient alors à la question suivante : quels sont les $a, b \in \mathbb{Q}$ vérifiant

$$\boxed{\begin{cases} a^2 + b^2 \text{ est le carré d'un rationnel} \\ ab = 20478 \end{cases}}$$

2 Affinage mathématique

On peut essayer de voir ceci sous un autre point de vue : il s'agit de trouver les triangles rectangles de côtés adjacents rationnels et d'aire $20478/2 = 10239$! En faisant quelques recherches, on se rend vite compte que ceci correspond à la notion de **nombres congruents** : un entier $n \in \mathbb{N}$ est dit *congruent* s'il existe $u, v, w \in \mathbb{Q}$ tels que $u^2 + v^2 = w^2$ et $uv/2 = n$.
On peut reformuler le problème en la détermination de points rationnels d'une certaine courbe elliptique. En effet, en supposant n non nul, et donc u, v non nul, w est non nul donc la première condition se réécrit

$$(u/w)^2 + (v/w)^2 = 1$$

Or, on dispose d'un sympathique paramétrage rationnel des points rationnels du cercle unité : leur ensemble s'écrit $\left\{ \left(\frac{1-t^2}{1+t^2}, -\frac{2t}{1+t^2} \right) \;\middle|\; t \in \mathbb{Q} \right\}$. En posant $(u/w, v/w) = \left(\frac{1-t^2}{1+t^2}, -\frac{2t}{1+t^2} \right)$ où $t \in \mathbb{Q}$, on a alors

$$\frac{-(1-t^2)2t}{(1+t^2)^2} = \frac{uv}{w^2} = \frac{2n}{w^2} \Leftrightarrow t^3 - t = n\left(\frac{1-t^2}{w}\right)^2 \Leftrightarrow (nt)^3 - n^2(nt) = \left(n^2 \frac{1-t^2}{w}\right)^2$$

c'est-à-dire, en posant $X = nt, Y = n^2(1-t^2)/w$, $\boxed{Y^2 = X^3 - n^2X}$, donc $(X, Y) \in E(\mathbb{Q})$ où E est la courbe elliptique définie par $(E) : y^2 = x^3 - n^2x$.
Réciproquement, on peut vérifier qu'à partir de tout point (x, y) sur la courbe tel que $y \neq 0$, on peut remonter à un triplet $(u, v, w) \in \mathbb{Q}^3$ solution du problème. Formellement, on peut exhiber la bijection

$$f \; : \; \begin{cases} \big\{(u,v,w) \in \mathbb{Q}^3 \mid u^2+v^2 = w^2 \text{ et } uv/2 = n\big\} \longrightarrow E(\mathbb{Q}) \cap (\mathbb{Q} \times \mathbb{Q}^*) \\ (u,v,w) \longmapsto \left(-\dfrac{nv}{u+w}, \dfrac{2n^2}{u+w}\right) \end{cases}$$

et sa réciproque

$$f^{-1} \; : \; \begin{cases} E(\mathbb{Q}) \cap (\mathbb{Q} \times \mathbb{Q}^*) \longrightarrow \big\{(u,v,w) \in \mathbb{Q}^3 \mid u^2+v^2 = w^2 \text{ et } uv/2 = n\big\} \\ (x,y) \longmapsto \left(\dfrac{n^2-x^2}{y}, -\dfrac{2xn}{y}, -\dfrac{n^2+x^2}{y}\right) \end{cases}$$

On s'est ramené au problème classique consistant à déterminer les points rationnels d'une courbe elliptique. Puisque $E(\mathbb{Q})$ muni son addition usuelle forme un groupe abélien de type fini (théorème de Mordell-Weil), on s'intéresse aux générateurs de ce groupe. Dans notre cas, $n = 10239$: le code Sage suivant est censé trouver les générateurs de $E(\mathbb{Q})$.

```
n = 10239
E = EllipticCurve(QQ, [-n^2, 0])
E.gens()
```

Malheureusement, l'algorithme utilisé par Sage peine clairement, même en augmentant le paramètre `descent_second_limit`... Essayons avec Magma :

```
n := 10239;
E:=EllipticCurve([-n^2, 0]);
Generators(E);
```

qui fonctionne (!) et donne les points

$$(10239 : 0 : 1), (0 : 0 : 1),$$
$$\left(\tfrac{7373437738623010880455094187939218690660 76}{108931592386005773136779172286525118 41} : \tfrac{62586211644444804739345860302955571366245002433098275717297503 0}{35952639365198540562613869494033558726733788804390127889} : 1\right)$$

Les deux premiers points étant les points triviaux, qu'on peut vérifier être d'ordre 2, le dernier d'ordre infini, de sorte que $E(\mathbb{Q}) \cong \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}$.

3 Récupération de la clé

Maintenant que nous avons tout en main, on peut choisir certains points non triviaux de $E(\mathbb{Q})$, calculer à l'aide de f^{-1} le triplet $(a, b, c) \in \mathbb{Q}^3$ et l'éventuelle clé de chiffrement p correspondants. Essayons avec le 3e générateur, avec Sage :

```
n = 10239
E = EllipticCurve(QQ, [-n^2, 0])


def finv(x, y):
  u =  (n^2 - x^2)/y
  v = -2*x*n/y
  w = -(n^2+x^2)/y

(x, y) = (73734377386230108804550941879392186906 6076/108
931592386005773136779172286525118 41, 625862116444448047393458603029555713 66245
0024330982757172975030/35952639365198540562613869494033558726733788804390127889)

P = E(x, y) #pas d'erreur : P est bien dans E(Q) !

(a, b, c) = finv(x, y)
print("p = ", abs(c.numerator()))

#renvoie : p = 5304570600451185316064006943162686343754322223830255783953667934616663888337
```

Puis avec Python, en ayant au préalable isolés l'IV et le cipher de `output.txt` :

```python

def decrypt(n, IV, cipher):
  k = int.to_bytes(n, 32, "big")
  aes = AES.new(k, AES.MODE_CBC, iv = IV)
  return aes.decrypt(cipher)

p = 5304570600451185316064006943162686343754322223830255783953667934616663888337

IV = b'\xa7\x9e\xc4\xa6\r3\xea\xe0\xe0\xd9\xe0o\x8b0\x93H'
cipher = b')\xd4\xc8\xdc\xee\xcbF\x1c\xfc|\x06$-%\x87\x9c\xdc\xf4\x7f\xcaG\xde\xd5\x12\xea\x83\r\ta>\xcdIz\x97 #\x1c\xb4#\xe9^\xd2F?_t\xd8\xf5\xc4\xc9\xb7W\x04\xffs\x8f\xe4\x84u\x19\x1bb\xf1B\x80\xf3,\x05\xda\xf90\n\xb1\xd6\x92\xd8qsq\xdc'

print(decrypt(p, IV, cipher))
```

on trouve le flag : `FCSC{67084c2bc8acfbf5e8a0d5e2809e230d092ab56630713dbe33ca42b8430a992b}`. :)