Solution à : Planètes anormales.

Le premier point consiste à analyser les courbes de l'énoncé. On voit qu'on peut saisir les données de : 1, 2 3 à notre guise mais que, dans le code, il y a une courbe 4 qui ne semble pas accessible par les menus donc on ne peut pas directement accéder à la courbe 4 par un menu. La première étape est comment accéder à la courbe 4.
Au lancement, on demande un nom et un numéro de courbe par le choix d'une destination et on est obligé de choisir une destination existante sinon le programme se termine. A priori, il n'est pas possible d'accéder à la courbe 4.

Dans le code source, le numéro de la courbe est transformé en : un code de courbe dans les 3 premières ; ensuite, le nom du Jedi est rajouté dans le token JSON ;

{'curve'='secp160r2', 'name'='Joda'}

Ce qui nous donne une faille d'accès. Le nom que l'on saisit va être, par exemple :
me', 'curve': 'other
Ce qui va créer le JSON :
{'curve'='secp160r2', 'name'='**me', 'curve': 'other**'}

Puis le code fait token = ast.literal_eval(token) ce qui nettoie le token en :

=> {'name'='**me', 'curve': 'other**'}

Puis dans :
```
def translate(curveName: str) -> Curve:
    try:
        print("## curve name", curveName)
        return Curves[curveName]
    except KeyError:
```

« other » n'existe pas dans la liste secp112r1, secp160k1 ou secp160r2

=> on a sélectionné la 4ème courbe.

****
Pourquoi la courbe 4 serait-elle meilleure ?
On peut utiliser Sagemath pour tester les caractéristiques des 4 courbes :
(on pourrait utiliser aussi Python mais le calcul serait plus long)
****
Je lance : Sagemath en docker (pour ne pas avoir à l'installer) :
docker run -it --rm - sagemath/sagemath sage

on teste la courbe 4 :

```
p = 0xf7fda1b2f0c9ea506e8a125766fd9e5046fd5716630c84f526fea8ce10497829
a = 0xbb0480e1f010abb2e69e7d72df5d75a23a15bc73710df25b6da04121f904e4f5
b = 0xfa2bddcca24c1d80baf26cb1e1f04cf78e995c675543c9692e959f83b470a03

F = GF(p)
E = EllipticCurve(F, [a, b])
n = E.order()
```

```
print("p =", p)
print("#E(F_p) =", n)
print("Anormale ?" , "O" if n == p else "N")
```

```
sage: p = 0xf7fda1b2f0c9ea506e8a125766fd9e5046fd5716630c84f526fea8ce10497829
....: a = 0xbb0480e1f010abb2e69e7d72df5d75a23a15bc73710df25b6da04121f904e4f5
....: b = 0xfa2bddcca24c1d80baf26cb1e1f04cf78e995c675543c9692e959f83b470a03
....:
....: F = GF(p)
....: E = EllipticCurve(F, [a, b])
....: n = E.order()
....:
....: print("p =", p)
....: print("#E(F_p) =", n)
....: print("Anormale ?" , "O" if n == p else "N")
....:
....:
p = 112169401912846082330019329915166338728317712342445509363066454585387376015401
#E(F_p) = 112169401912846082330019329915166338728317712342445509363066454585387376015401
Anormale ? O
sage:
```

Elle est anormale.
Les 3 autres, même celle qui a des 0, sont « normales »

COURBE 1

p = 0xdb7c2abf62e35e668076bead208b
a = 0xdb7c2abf62e35e668076bead2088
b = 0x659ef8ba043916eede8911702b22

F = GF(p)
E = EllipticCurve(F, [a, b])
n = E.order()

print("p =", p)
print("#E(F_p) =", n)
print("Anormale ?" , "O" if n == p else "N")

COURBE 2

p = 0xffffffffffffffffffffffffffffffffefffac73
a = 0x0000000000000000000000000000000000000000
b = 0x0000000000000000000000000000000000000007

F = GF(p)
E = EllipticCurve(F, [a, b])
n = E.order()

print("p =", p)
print("#E(F_p) =", n)
print("Anormale ?" , "O" if n == p else "N")

COURBE 3

p = 0xffffffffffffffffffffffffffffffffeffffac73
a = 0xffffffffffffffffffffffffffffffffeffffac70
b = 0xb4e134d3fb59eb8bab57274904664d5af50388ba

```
F = GF(p)
E = EllipticCurve(F, [a, b])
n = E.order()

print("p =", p)
print("#E(F_p) =", n)
print("Anormale ?" , "O" if n == p else "N")
```

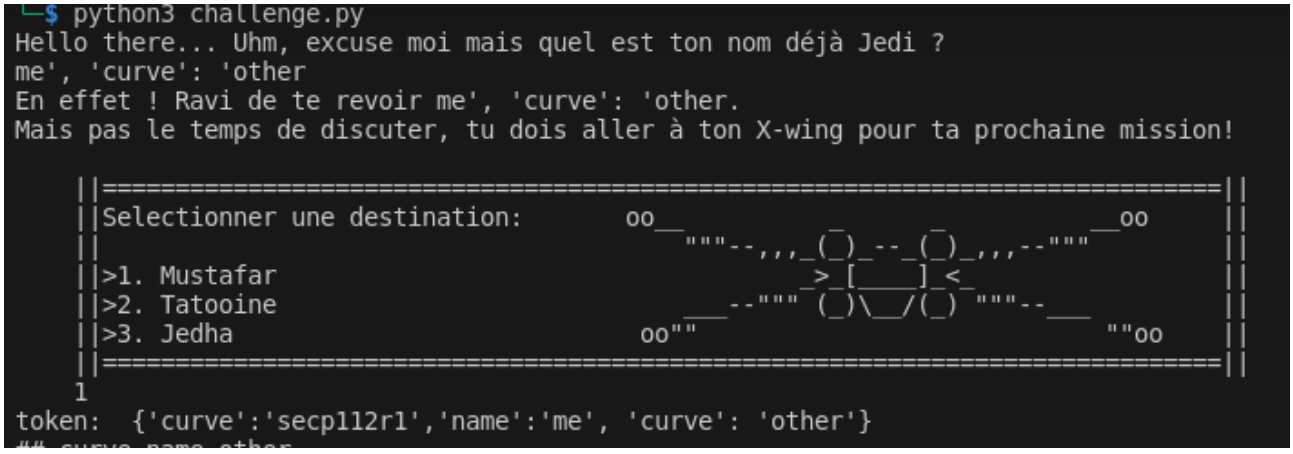
⇒ toutes normales.

Pour avoir un flag :
****
On se connecte sur le netcat distant.

On saisit le nom ci dessus : **me', 'curve': 'other**

et la courbe est sans importance.
Cela donne le point
qx =
393113162295614080169453820122557212768190256524027867019552980031970764197 73
qy =
477120865303209051764776972571045658336215658352042354888537749054562211039 27
et le flag crypté
23c21181c74b0b2084ec1232606ba05b59374f309f849de74530800039abcbedf277d05e6c8ef29c871b78c908297107

Pour décrypter (déchiffrer) une courbe elliptique, il nous faut :
- le point Q (ci dessus)
- le message crypté (ci dessus)
- le paramètre IV : en regardant le source par chance, il s'obtient :
IV = notre nom = « me» car IV = token["name"]



et il faut aussi le point difficile : d = rd.randint(2, curve.p - 1)
Ce : d est quasi impossible à déterminer sur un algo AES CBC normal (des centaines d'heures de calcul) : mais par chance, l'une des courbes est anormale.

Sagemath peut en utilisant un algorithme «SMART» décoder bien plus rapidement (il existe en python mais encore : bien moins rapide).

On donne a Sagemath nos paramètres :

```
....: p  = 0xf7fda1b2f0c9ea506e8a125766fd9e5046fd5716630c84f526fea8ce10497829
....: a  = 0xbb0480e1f010abb2e69e7d72df5d75a23a15bc73710df25b6da04121f904e4f5
....: b  = 0x0fa2bddcca24c1d80baf26cb1e1f04cf78e995c675543c9692e959f83b470a03
....: gx = 0x735d07d96821ec8bff37eb23c31081ea526ddc10abe22375518c44e043a39db0
....: gy = 0x97e570cf7c177584ddd036d9181a3f5f83307f60c92b539a2d4f479d9c9ad4bd
....: qx = 393113162295614080169453820122557212768190256524027867019552980031970764197 73
....: qy = 477120865303209051764776972571045658336215658352042354888537749054562211039 27
....:
....: # Initialisation du corps et de la courbe
....: F = GF(p)
....: E = EllipticCurve(F, [a, b])
....: G = E(gx, gy)
....: Q = E(qx, qy)
....:
....: # Vérification que la courbe est bien anormale
....: assert E.order() == p, "La courbe n'est pas anormale (ordre ≠ p)"
....:
....: print("[*] Attaque par méthode rapide u = x/y mod p (corrigée)...")
....:
....: # Calcul de u(G) et u(Q)
....: uG = (gx * inverse_mod(gy, p)) % p
....: uQ = (qx * inverse_mod(qy, p)) % p
....:
....: # Tentative 1 : d = u(Q)/u(G)
....: try:
....:     d1 = (uQ * inverse_mod(uG, p)) % p
....:     if d1 * G == Q:
....:         d = d1
....:         print("[+] d trouvé avec formule u = x/y mod p (Q direct)")
....:     elif (-d1 % p) * G == Q:
....:         d = (-d1) % p
....:         print("[+] d trouvé avec formule u = x/y mod p (Q opposé)")
....:     else:
....:         print("[-] Formule x/y n'a pas donné de résultat direct.")
....:         d = None
....: except Exception as e:
....:     print("[-] Erreur lors du calcul avec la formule rapide :", e)
....:     d = None
....:
....: # Vérification via padic_elliptic_logarithm
....: print("[*] Attaque de Smart via padic_elliptic_logarithm...")
....: d_smart = G.padic_elliptic_logarithm(Q, p)
....: assert d_smart * G == Q
....:
....: print("[+] d (Smart) =", hex(d_smart))
....:
....: if d is not None and d != d_smart:
....:     print("[!] ⚠d (formule) ≠ d (Smart), différence de signe ou bug")
....:
....: print("[✓] d =", hex(d_smart))
[*] Attaque par méthode rapide u = x/y mod p (corrigée)...
[-] Formule x/y n'a pas donné de résultat direct.
[*] Attaque de Smart via padic_elliptic_logarithm...
[+] d (Smart) = 0x268dc922041bc83705b0f8b79a6c8c2ce252a7608271eb4b43b46f3db8e2027b
[✓] d = 0x268dc922041bc83705b0f8b79a6c8c2ce252a7608271eb4b43b46f3db8e2027b
sage:
```

Ce qui donne : d très vite (sans algorithme smart, cela prend un CPU énorme et finit pas abandonner)
Ayant : d, on a tous les paramètres de décodage.

On écrit juste decrypt.py et on obtient notre flag.

decrypt.py

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad, pad
from Crypto.Util.number import long_to_bytes
import base64

# === Paramètres ===
# secret d
d = 0x268dc922041bc83705b0f8b79a6c8c2ce252a7608271eb4b43b46f3db8e2027b  # <-- données par sagemath
username = "me"         # <-- le nom du début
ciphertext = bytes.fromhex("23c21181c74b0b2084ec1232606ba05b59374f309f849de74530800039abcbedf277d05e6c8ef29c871b78c908297107")  # <-- la chaine crtptée du serveur netcat

# === Clé AES et IV ===
key = pad(long_to_bytes(d), 32)[:32]
iv  = pad(username.encode(), 16)[:16]

# === Déchiffrement ===
cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext_padded = cipher.decrypt(ciphertext)

try:
    plaintext = unpad(plaintext_padded, 16)
    print("[+] Message déchiffré :", plaintext.decode())
except ValueError:
    print("[-] Mauvais padding ou d/IV incorrect")
```

```
└─$ python3 decrypt.py
[+] Message déchiffré : 404CTF{CuRv3S_0N_Th4t_Pl4N3TeS_4rE_aN0mAl0uS}
```