WiCyS CTF 2023 Intelligent Curve Writeup

Manav Malik

We begin, as always, by examining the challenge description:

> The other day, I was just looking at some words, and I found a five-letter word I thought was pretty cool! I did what any normal person does and encoded it using this elliptic curve.
>
> Consider the elliptic curve defined by $y^2 = x^3 + 16332764x + 8847792$ over a finite field of size 4952019383419. Points P and Q on the curve are given below such that $kP = Q$ where k is the message (encoded flag).
>
> $P = (637246119517, 588899099134)$
>
> $Q = (1241945849630, 4641791664291)$
>
> Your general solve path should be determining the value of k and decoding the number into the word. The method used to encode the word into a number involves the ASCII values of each letter. The word consists of five letters ([a-zA-Z]{5}). Remember to wrap the word in the WCS{} format.

The first step is, as the challenge description says, determining the value of k that satisfies $kP = Q$. In general, trying to calculate Q/P by hand is computationally very difficult and not a viable solution. Instead, we will need to exploit a major vulnerability of this elliptic curve.

One of the first things to check in any ECC challenge is the order of the elliptic curve, $\#E(F_p)$, where F_p is a finite field of size p. The order of an elliptic curve is essentially equal to the number of reachable points on the curve. If $\#E(F_p) = p$ (the trace of Frobenius is 1), we can use an exploit known as Smart's Attack. We'll use SageMath for most of our calculations in this challenge.

```
p = 4952019383419
E = EllipticCurve(GF(p), [16332764, 8847792])
# GF(n) defines a Galois Field of size n
print(E.order() == p) # outputs True
```

Now that we've determined that Smart's Attack can be applied, we'll use an implementation of it to determine the value of k.

My personal implementation of Smart's Attack that I used to verify this challenge is messy and unreadable, so here I'm going to shamelessly steal Joachim Vandersmissen's implementation from his crypto-attacks repo on GitHub.

```
def lift(E, P, gf):
    x, y = map(ZZ, P.xy())
    for point in E.lift_x(x, all=True):
        _, y_ = map(gf, point.xy())
        if y == y_:
            return point


def attack(G, P):
    E = G.curve()
    gf = E.base_ring()
    p = gf.order()
    assert E.trace_of_frobenius() == 1

    E = EllipticCurve(Qp(p),
        [int(a) + p * ZZ.random_element(1, p)
        for a in E.a_invariants()])
    G = p * lift(E, G, gf)
    P = p * lift(E, P, gf)
    Gx, Gy = G.xy()
    Px, Py = P.xy()
    return int(gf((Px / Py) / (Gx / Gy)))
```

Running attack(P, Q) yields $k \equiv 76694417741 \pmod{p}$. Now, what remains is determining the word.

According to the challenge description, the word was encoded into a number with the ASCII values of each letter. These values will be two to three digits, so we should split our value for k up into triplets (leading zeroes on numbers < 100). (If you think all this is too guessy, this is quickly realized when we see that 76-69-44-177-41 becomes "LE,±".) Thus, we should expect our final number to be 15 digits long (or 14 if the first letter has an ASCII value < 100). Because k is a number modulo p, we can repeatedly add p to our number and test it out.

```
k = 76694417741
p = 4952019383419

alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

while 15 - len(str(k)) > 1:
    k += p

valid = False
word = ""
while not valid:
    word = ""
    ks = str(k)
    chars = []

    if len(ks) == 14:
        chars = [ks[:2],ks[2:5],ks[5:8],ks[8:11],ks[11:]]
    else:
        chars = [ks[:3],ks[3:6],ks[6:9],ks[9:12],ks[12:]]

    valid = True
    for char in chars:
        c = chr(int(char))
        if c not in alpha:
                valid = False
        word += c

    k += p

print(word)
```

In the end, the valid word is "cuRVy," so our flag is thus: WCS{cuRVy}.