Project 4: Implement and Attack ECDSA with Repeated Nonce

# Good to know

I renamed N to ORDER\_N for my personal understanding.

I also renamed the variables of the G point to G\_POINT, G\_POINTx, G\_POINTy also for my personal understanding.

# Private key, nonce and attack conclusion

To create the signature :

To compute the private key, I used the following computation :

*PRIVATE\_KEY = randint(0, ORDER\_N - 1)*

It is the same for the nonce ‘k’:

*k = randint(1, ORDER\_N - 1)*

They both need to be ORDER\_N - 1 so it can be in the elliptic curve’s order.

To attack when there was the same nonce used :

I determine the nonce k = 10.

The goal of the attack is to retrieve the private key when the same nonce was used. Therefore, by writing the following equation into Python, I could retrieve the nonce k and use it to retrieve the private key.

Une image contenant texte

Description générée automatiquement

And the result in Python is :

s = s0 - s1  
k = (modinv(s, ORDER\_N) \* (m0\_hash - m1\_hash)) % ORDER\_N  
private\_key = (s0\*k - m0\_hash) \* modinv(r0, ORDER\_N) % ORDER\_N

As the attack function returns the computed private key, I just need to compare it the private key constant of the program and print a message if it is the same:

result = attack(list, list2)  
if result == PRIVATE\_KEY:  
 print(f"Private key found : {PRIVATE\_KEY}")

# Time processing

The total time for one signature and one verification is about more or less:

*Processing time: 0.0005640983581542969 second(s)*