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.

Note that $e_i = H(m_i)$ are known. It is easy to solve the above equations as the following

$$\begin{cases} k = & (s_0 - s_1)^{-1}(e_0 - e_1) \bmod n \\ d = & (s_0 k - e_0) r_0^{-1} \bmod n \end{cases}$$

Therefore, the private key d can be determined. This very attack was used

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)