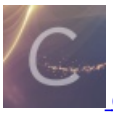
攻防世界crypto easy_ECC

原创

小元砸 于 2020-12-05 20:49:42 发布

分类专栏： ctf

本文链接：https://blog.csdn.net/qq_49480008/article/details/110725708



攻防世界crypto easy_ECC

刚开始看见6分就知道不简单

看见文本之后联想到了py脚本（但可惜卑微的我不会）所以就照搬了大佬的脚本

```
import collections
import random

EllipticCurve = collections.namedtuple('EllipticCurve', 'name p a b g n h')

curve = EllipticCurve(
   'secp256k1',
   # Field characteristic.
   p=int(input('p=')),
   # Curve coefficients.
   a=int(input('a=')),
```

```
    b=int(input('b=')),
    # Base point.
    g=(int(input('Gx=')),
       int(input('Gy='))),
    # Subgroup order.
    n=int(input('k=')),
    # Subgroup cofactor.
    h=1,
)


# Modular arithmetic ##########################################################

def inverse_mod(k, p):
    """Returns the inverse of k modulo p.
  This function returns the only integer x such that (x * k) % p == 1.
  k must be non-zero and p must be a prime.
  """
    if k == 0:
        raise ZeroDivisionError('division by zero')

    if k < 0:
        # k ** -1 = p - (-k) ** -1 (mod p)
        return p - inverse_mod(-k, p)

    # Extended Euclidean algorithm.
    s, old_s = 0, 1
    t, old_t = 1, 0
    r, old_r = p, k

    while r != 0:
        quotient = old_r // r
        old_r, r = r, old_r - quotient * r
        old_s, s = s, old_s - quotient * s
        old_t, t = t, old_t - quotient * t

    gcd, x, y = old_r, old_s, old_t

    assert gcd == 1
    assert (k * x) % p == 1

    return x % p


# Functions that work on curve points #########################################

def is_on_curve(point):
    """Returns True if the given point lies on the elliptic curve."""
    if point is None:
        # None represents the point at infinity.
        return True

    x, y = point

    return (y * y - x * x * x - curve.a * x - curve.b) % curve.p == 0


def point_neg(point):
    """Returns -point."""
```

```
    assert is_on_curve(point)

    if point is None:
        # -0 = 0
        return None

    x, y = point
    result = (x, -y % curve.p)

    assert is_on_curve(result)

    return result


def point_add(point1, point2):
    """Returns the result of point1 + point2 according to the group law."""
    assert is_on_curve(point1)
    assert is_on_curve(point2)

    if point1 is None:
        # 0 + point2 = point2
        return point2
    if point2 is None:
        # point1 + 0 = point1
        return point1

    x1, y1 = point1
    x2, y2 = point2

    if x1 == x2 and y1 != y2:
        # point1 + (-point1) = 0
        return None

    if x1 == x2:
        # This is the case point1 == point2.
        m = (3 * x1 * x1 + curve.a) * inverse_mod(2 * y1, curve.p)
    else:
        # This is the case point1 != point2.
        m = (y1 - y2) * inverse_mod(x1 - x2, curve.p)

    x3 = m * m - x1 - x2
    y3 = y1 + m * (x3 - x1)
    result = (x3 % curve.p,
              -y3 % curve.p)

    assert is_on_curve(result)

    return result


def scalar_mult(k, point):
    """Returns k * point computed using the double and point_add algorithm."""
    assert is_on_curve(point)



    if k < 0:
        # k * point = -k * (-point)
        return scalar_mult(-k, point_neg(point))
```

```
    result = None
    addend = point

    while k:
        if k & 1:
            # Add.
            result = point_add(result, addend)

        # Double.
        addend = point_add(addend, addend)

        k >>= 1

    assert is_on_curve(result)

    return result


# Keypair generation and ECDHE ################################################

def make_keypair():
    """Generates a random private-public key pair."""
    private_key = curve.n
    public_key = scalar_mult(private_key, curve.g)

    return private_key, public_key



private_key, public_key = make_keypair()
print("private key:", hex(private_key))
print("public key: (0x{:x}, 0x{:x})".format(*public_key))
```

解出X，Y的值

```
private key: 0x857d0
public key: (0xcb19fe553fa, 0x50545408eb4)
```

再16进制转10进制可得

2进制 4进制 8进制 10进制 16进制 32进制 16进制

转换数字 cb19fe553fa

2进制 4进制 8进制 10进制 16进制 32进制 10进制

转换结果 13957031351290

题目要求答案是x+y所以得出答案为

5520194834100+13957031351290

19 477 226 185 390

(	)	%	C
7	8	9	÷
4	5	6	x
1	2	3	-
0	.	=	+