Background Reading

The flag is the name we give groups with a commutative operation.

> crypto{Abelian}

Point Negation

$Q = -P = (8045, -6936)$

$-6936 mod p = 2803$

> flag:crypto{8045,2803}

Point Addition

根据公式

(a) If P = O, then P + Q = Q.

(b) Otherwise, if Q = O, then P + Q = P.

(c) Otherwise, write P = (x_1, y_1) and Q = (x_2, y_2).

(d) If $x_1 = x_2$ and $y_1 = -y_2$, then P + Q = O.

(e) Otherwise:

  (e1) if P ≠ Q: $\lambda = (y_2 - y_1) / (x_2 - x_1)$

  (e2) if P = Q: $\lambda = (3x_1^2 + a) / 2y_1$

(f) $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$

(g) P + Q = (x_3, y_3)

可通过如下代码解出flag

注意的点：除法是F_p下的除法，运用到模的逆运算

```
from Crypto.Util.number import inverse

a = 497
b = 1768
prime = 9739

def point_addition(p,q):
    if(p==(0,0)):
        return q
```

```python
    if(q==(0,0)):
        return p
    x1,y1 = p
    x2,y2 = q
    if x1==x2 and y1+y2==0:
        return (0,0)
    if p==q:
        s1 = (3*x1**2+a)%prime
        s2 = 2*y1
        s = s1*inverse(s2,prime)
    else:
        s1 = (y2-y1)%prime
        s2 = (x2-x1)%prime
        s = s1*inverse(s2,prime)
    x3 = (s*s-x1-x2)%prime
    y3 = (s*(x1-x3)-y1)%prime
    return (x3,y3)

p=(493,5564)
q=(1539,4742)
r=(4403,5202)
ans = point_addition(point_addition(point_addition(p,p),q),r)
print(ans)
```

> flag:crypto{4215,2162}

Scalar Multiplication

根据cryptohack给出的计算方法（快速幂的思想）

Input: P in E(F_p) and an integer n > 0
1. Set Q = P and R = O.
2. Loop while n > 0.
   3. If n ≡ 1 mod 2, set R = R + Q.
   4. Set Q = 2 Q and n = ⌊n/2⌋.
   5. If n > 0, continue with loop at Step 2.
6. Return the point R, which equals nP.

·便容易解出flag,（下面的代码运用了上题的point_addition函数）

```python
def scalar_mul(p,n):
    q = p
    r = (0,0)
    while n>0:
        if n%2 == 1:
            r = point_addition(r,q)
        q = point_addition(q,q)
        n = n//2
    return r
p = (2339,2213)
n = 7863
print(scalar_mul(p,n))
```

> flag:crypto{9467,2742}

Curves and Logs

首先根据cryptohack上的介绍

Alice generates a secret random integer n_A and calculates $Q_A = n_A G$

Bob generates a secret random integer n_B and calculates $Q_B = n_B G$

Alice sends Bob Q_A, and Bob sends Alice Q_B. Due to the hardness of ECDLP, an onlooker Eve is unable to calculate $n_{A/B}$ in reasonable time.

Alice then calculates $n_A Q_B$, and Bob calculates $n_B Q_A$.

Due to the associativity of scalar multiplication, $S = n_A Q_B = n_B Q_A$.

Alice and Bob can use S as their shared secret.

可由题目给的Q_A, n_B计算出the shared secret S,将S的x坐标按题目要求先转化为十进制，再转化为字符，最后化为十六进制表示，即可得到flag，代码如下（接在前面的加法和乘法后）

```python
QA = (815,3190)
nb = 1829
G = (1804,5368)
S = scalar_mul(QA,nb)
key = hashlib.sha1()
key.update(str(S[0]).encode())
print(key.hexdigest())
```

> flag:crypto{80e5212754a824d3a4aed185ace4f9cac0f908bf}

Efficient Exchange

- 首先求出Q的y坐标
  - 由椭圆曲线方程和`q_x`可以得出y^2的值
  - 又由于p是奇素数，且$y^2 == y \times ymod(p)$满足欧拉准则，所以该方程存在二次剩余，$x^2 \equiv a(modp)$的解为$a^{(m+1)/4}$
- 得到Q的坐标后，可由Q和n_B的值算出公钥S的值（1791，2181），取其x坐标用题目所给的程序decrypt可得到flag，代码如下

计算S

```
qx = 4726
nb = 6534
y2 = (qx**3+497*qx+1768)%prime
y = pow(y2,(prime+1)//4,prime)
print(y)
S = scalar_mul((4726,6287),6534)
print(S)
```

题目所给代码decrypt.py

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import hashlib


def is_pkcs7_padded(message):
    padding = message[-message[-1]:]
    return all(padding[i] == len(padding) for i in range(0, len(padding)))


def decrypt_flag(shared_secret: int, iv: str, ciphertext: str):
    # Derive AES key from shared secret
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode('ascii'))
    key = sha1.digest()[:16]
    # Decrypt flag
    ciphertext = bytes.fromhex(ciphertext)
    iv = bytes.fromhex(iv)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)

    if is_pkcs7_padded(plaintext):
        return unpad(plaintext, 16).decode('ascii')
    else:
        return plaintext.decode('ascii')


shared_secret = 1791
```

```python
iv = 'cd9da9f1c60925922377ea952afc212c'
ciphertext = 'febcbe3a3414a730b125931dccf912d2239f3e969c4334d95ed0ec86f6449ad8'

print(decrypt_flag(shared_secret, iv, ciphertext))
```

> flag:crypto{3ff1c1ent_k3y_3xch4ng3}

Montgomery's Ladder

本题就是代码实现cryptohack上所给的算法

首先实现Montgomery Curve上的addition和doubling

Addition formula for Montgomery Curve (Affine)

Input: P, Q in $E(F_p)$ with P != Q
Output: R = (P + Q) in $E(F_p)$

$\alpha = (y_2 - y_1) / (x_2 - x_1)$
$x_3 = B\alpha^2 - A - x_1 - x_2$
$y_3 = \alpha(x_1 - x_3) - y_1$

Doubling formula for Montgomery Curve (Affine)

Input: P in $E(F_p)$
Output: R = [2]P in $E(F_p)$

$\alpha = (3x^2_1 + 2Ax_1 + 1) / (2By_1)$
$x_3 = B\alpha^2 - A - 2x_1$
$y_3 = \alpha(x_1 - x_3) - y_1$

代码如下

```python
def addition(P,Q):
    x1,y1 = P
    x2,y2 = Q
    s1 = (y2-y1)%p
    s2 = (x2-x1)%p
    s = s1*inverse(s2,p)
    x3 = (b*s*s-a-x1-x2)%p
    y3 = (s*(x1-x3)-y1)%p
    return (x3,y3)

def doubling(P):
    x1,y1 = P
    s1 = (3*x1**2+2*a*x1+1)%p
    s2 = (2*b*y1)%p
    s = s1*inverse(s2,p)
    x3 = (b*s*s-a-2*x1)%p
    y3 = (s*(x1-x3)-y1)%p
    return (x3,y3)
```

然后就是**Montgomery's binary algorithm**

Montgomery's binary algorithm in the group $E(F_p)$

Input: P in $E(F_p)$ and an l-bit integer $k = \Sigma\ 2^i\ k_i$ where $k_{l-1} = 1$
Output: [k]P in $E(F_p)$

1. Set (R_0, R_1) to (P, [2]P)
2. for i = l - 2 down to 0 do
3. If $k_i = 0$ then
4. Set (R_0, R_1) to $([2]R_0, R_0 + R_1)$
5. Else:
6. Set (R_0, R_1) to $(R_0 + R_1, [2]R_1)$
7. Return R_0

代码如下

```
def mon_bin(P,list):
    R0,R1 = P,doubling(P)
    for i in list[1:]:
        if i == 0:
            R0,R1 = doubling(R0),addition(R0,R1)
        else:
            R0,R1 = addition(R0,R1),doubling(R1)
    return R0
```

以上就是本题的主要逻辑。

剩余的一些点:

- 将十六进制表示转化为二进制表示，并把每一位存入列表，便于算法的实现
- 二次剩余的计算，由于p模四余1，刚开始还写了代码求解二次剩余，flag正确后查看题解发现python直接有个库函数sqrt_mod可以直接求解二次剩余,大大简化工作量

完整代码如下

```
from Crypto.Util.number import inverse
from sympy.ntheory import sqrt_mod

b = 1
a = 486662
p = pow(2,255) - 19

def addition(P,Q):
    x1,y1 = P
    x2,y2 = Q
    s1 = (y2-y1)%p
    s2 = (x2-x1)%p
    s = s1*inverse(s2,p)
    x3 = (b*s*s-a-x1-x2)%p
    y3 = (s*(x1-x3)-y1)%p
    return (x3,y3)
```

```python
def doubling(P):
    x1,y1 = P
    s1 = (3*x1**2+2*a*x1+1)%p
    s2 = (2*b*y1)%p
    s = s1*inverse(s2,p)
    x3 = (b*s*s-a-2*x1)%p
    y3 = (s*(x1-x3)-y1)%p
    return (x3,y3)

def mon_bin(P,list):
    R0,R1 = P,doubling(P)
    for i in list[1:]:
        if i == 0:
            R0,R1 = doubling(R0),addition(R0,R1)
        else:
            R0,R1 = addition(R0,R1),doubling(R1)
    return R0

Gx = 9
y2 = (Gx**3 + a*Gx**2 + Gx)%p
y = sqrt_mod(y2,p)
G = (Gx,y)

hex_number = "1337c0decafe"
binary = bin(int(hex_number, 16))
bin_string = binary[2:]
bin_list = [int(bit) for bit in bin_string]
ans = mon_bin(G,bin_list)[0]%p
print(ans)
```

> flag:crypto{4923135046278601606433675697741265479338396472677189298250742092156302378152}

Smooth Criminal

首先分析题目给的加密代码source.py

该代码的基本作用:

- 定义了一个Point类来表示椭圆曲线上的点，并定义曲线的O
- 接下来实现了椭圆曲线上基本的运算函数并定义参数 (p,a,b,G)
- 对于加密通信的环节，代码首先生成了私钥n,并通过G计算得到公钥public，将公钥发送给Bob，又Bob的公钥B已知，通过Bob的公钥B和自己的私钥n可以得到shared_secret，并将其通过加密函数得到加密后的flag
- 而该加密函数采用的是AES加密，密钥由SHA-1散列算法由shared_secret生成

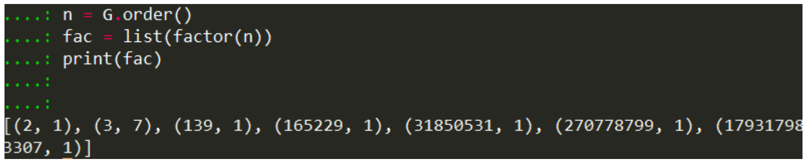
由output可知:

> {'iv': 'e0a6489405dc407dbe1823c304c929c3', 'encrypted_flag': '340e889c40ac663991b0b452da29183e90b0e83b1afd557087d04374ade26fa20a388dbd75bde36acd5f01ffe992eb7c'}

> Point(x=28081018213141489873037898276610121O916,
> y=29150649076805447815983560463271O368904),即Alice的公钥

由题目函数shared_secret的生成采用Q_B和n_A生成，所以要首先求出Alice的私钥，转化为离散对数问题

本题题目的smooth让我想到了求解离散对数的Pohlig-hellman algorithm，因此首先验证G是否是光滑的



```
....: n = G.order()
....: fac = list(factor(n))
....: print(fac)
....:
....:
[(2, 1), (3, 7), (139, 1), (165229, 1), (31850531, 1), (270778799, 1), (17931798
3307, 1)]
```

得到该曲线光滑，且分解产生的素因数都不是很大，因此可以用此方法求出n_A,由此得到$S = n_A \times Q_B$

而得到shared_secret之后，便由source中的加密函数方法得到解密函数，解密回去便可得到flag。

完整代码：

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad,unpad
from sage.all import *
import hashlib

p = 310717010502520989590157367261876774703
a = 2
b = 3
F = GF(p)
E = EllipticCurve(F,[a,b])
g_x = 179210853392303317793440285562762725654
g_y = 105268671499942631758568591033409611165
G = E.point((g_x,g_y))

b_x = 272640099140026426377756188075937988094
b_y = 51062462309521034358726608268084433317
QB = E.point((b_x,b_y))

a_x=280810182131414898730378982766101210916
a_y=291506490768054478159835604632710368904
QA = E.point((a_x,a_y))

n = G.order()
fac = list(factor(n))
# print(fac)

module = []
remain = []
for i,j in fac:
    mod = i**j
    g = G*ZZ(n/mod)
    q = QA*ZZ(n/mod)
    dl = discrete_log(q,g,operation = "+")
    module.append(mod)
    remain.append(dl)
```

```
nA = crt(remain,module)
S = QB*nA

sha1 = hashlib.sha1()
sha1.update(str(S[0]).encode('ascii'))
key = sha1.digest()[:16]

iv=bytes.fromhex('07e2628b590095a5e332d397b8a59aa7')
encrypted_flag=bytes.fromhex('8220b7c47b36777a737f5ef9caa2814cf20c1c1ef496ec21a9b483
3da24a008d0870d3ac3a6ad80065c138a2ed6136af')
cipher = AES.new(key,AES.MODE_CBC,iv)
flag = unpad(cipher.decrypt(encrypted_flag),16).decode()
print(flag)
```

> flag:crypto{n07_4ll_curv3s_4r3_s4f3_curv3s}

Curveball

我的思路

```
def search_trusted(self, Q):
        for host, cert in self.trusted_certs.items():
            if Q == cert['public_key']:
                return True, host
        return False, None

    def sign_point(self, g, d):
        return g * d

    def connection_host(self, packet):
        d = packet['private_key']
        if abs(d) == 1:
            return "Private key is insecure, certificate rejected."
        packet_host = packet['host']
        curve = packet['curve']
        g = Point(*packet['generator'])
        Q = self.sign_point(g, d)
        cached, host = self.search_trusted(Q)
        if cached:
            return host
        else:
            self.trusted_certs[packet_host] = {
                "public_key": Q,
                "curve": "secp256r1",
                "generator": G
            }
            return "Site added to trusted connections"
```

从上面的检查代码可以得到，服务器并没有正确检查证书的所有参数，想到它不检查用于生成`public`的生成点，因此或许可以向服务器发送我们自己的生成元以通过`search_trust`中的检查。但是对于具体实现不太有思路/(ㄒoㄒ)/~~