Writeup

1. 与服务器程序进行交互，了解Crypto程序的功能。

```
sha256(XXXX+KKMzFFD2cxBuBHdd) == a0f3e398815d486d1782e0df01fcb001781f1030eca497af5c14ede3b835df23
Give me XXXX:
```

开始是一个简单的proof_of_work，爆破一下即可。接着，就进入到程序的主页面：

```
Hello,guys!Welcome to my ECC Signature System!I promise no one can exploit it!
Howevers if you can exploit it in 10 times,I will give what you want!
Here is the frist message(64 bytes):\Iy1$N%osh&qBiSL'KxR4XHbfk(P]EWM6{nm|8CVQ-<D5Fj0}d2;/t)Yg[!T_+*u
Here is the second message(64 bytes):B*ImvOHyM'gCDb{EqGZnd2N.zJVYP=~5Q7Wk[/^"K:%3f\j1]UT&wAar $-o;8?4
Try to calculate the same signature for this two messages~
(((Notice: curve = SECP256k1, hashfunc = sha1)))

ECC Signature System:
    1. Show your pubkey
    2. Generate new prikey
    3. Update your pubkey
    4. Sign a message
    5. Verify a message
    6. Exploit
    7. Exit

You have only 10 times to operate!
Please choice your options:$
```

题目实现的是一个ECDSA签名与验证的功能，给我们提供了6个功能选项：

1）获取服务器给我们分配的公钥。

2）重新生成一个新的私钥，并分配新的公钥。

3）我们自定义自己的公钥。

4）对消息进行签名。

5）对消息和对应的签名进行认证。

6）破解ECDSA，获取flag。

此外，题目提供给了我们两条64bytes的消息，要求我们提供相同的签名能够验证通过，并且提示我们题目使用的是 `SECP256k1` 曲线和 `sha1`哈希算法。那么，思路就很清晰了：**我们需要计算得到两条消息相同的签名，使得他们都可以通过验证，即可得到flag。**

2. 这道题目，来源于ECDSA算法的一个特性：**我们可以利用两个消息的hash值，构造出特定的私钥来生成满足不同消息的重复签名。**

关于ECDSA的原理，可以参考这里。

设服务器生成椭圆曲线为`(a,b,p,G,n,h)`，对我们签名使用的私钥为x，验证公钥为h，那么有

```
h = xG
```

利用私钥，我们对消息m的签名为

```
s = k^(-1)*(H+x*r) mod p
```

则消息m的签名值为(r,s)。上述式子中，k为小于曲线阶n的随机值，r则是随机生成的y值与G点相乘的x坐标。

验证签名的公式如下：

```
r = H*s^(-1)*G+r*s^-1*h mod p
↓↓↓
r = H*s^(-1)*G+r*s^(-1)*x*G mod p
↓↓↓
r = s^(-1)*G*(H+r*x) mod p
```

对于不同的消息m1、m2，对应的哈希值为H1、H2，则有

```
r = s1^(-1)*G*(H1+r*x) mod p
r = s2^(-1)*G*(H2+r*x) mod p
```

将上述两式联立，得到符合条件的x值：

```
x = -(H1+H2)/2r mod p
```

我们将x带入m1对应的式子中：

```
r = s1^-1*G*(h1-(h1+h2)/2) mod p
↓↓↓
r = G*(h2-h1)/2s1 mod n
```

接着，将x带入前面的s1公式中，有

```
s1 = (h2-h1)/2k1 mod p
```

两者联立，可得：

```
r = k1*G mod p
```

等式成立，同理将其带入s2，结论也是成立的，这样我们就找到了所需的私钥x。

3. 根据前面的思路，我们计算得到所需私钥x，并生成对应的公钥值，发送给服务端更新pubkey，接着将生成的签名发给服务端即可得到flag，详见exp.py。