CTF-CRYPTO-RSA-ECC

原创

大熊何在 于 2020-11-26 09:09:14 发布

分类专栏： CRYPTO

本文链接：https://blog.csdn.net/zippo1234/article/details/110101200

CTF-CRYPTO-RSA-ECC

这几天在准备一个培训，身心俱疲，略有懈怠。。。

每天一题，只能多不能少

RSA-ECC

题目分析

1. ECC椭圆曲线方程
2. sage求因式
3. gcd(p,n) != 1

1.题目

ecn.py

```
from fastecdsa.point import Point
from Crypto.Util.number import bytes_to_long, isPrime
from os import urandom
from random import getrandbits
from secret import flag, myCurve

def gen_rsa_primes(G):
 urand = bytes_to_long(urandom(521//8))
 while True:
  s = getrandbits(521) ^ urand
  Q = s*G
  if isPrime(Q.x) and isPrime(Q.y):
   print("ECC Private key:", hex(s))
   print("RSA primes:", hex(Q.x), hex(Q.y))
   print("Modulo:", hex(Q.x * Q.y))
   return (Q.x, Q.y)

ecc_p = myCurve.p
a = myCurve.a
b = myCurve.b

Gx = myCurve.gx
Gy = myCurve.gy
G = Point(Gx, Gy, curve=myCurve)

e = 65537
p, q = gen_rsa_primes(G)
n = p * q

m = bytes_to_long(flag)
c = pow(m, e, n)

print 'ECC Curve Prime = %s' % str(ecc_p)
print 'Curve a = %s' % str(a)
print 'Curve b = %s' % str(b)
print 'Gx = %s' % str(Gx)
print 'Gy = %s' % str(Gy)
print 'n = %s' % str(n)
print 'c = %s' % str(c)

# ECC Curve Prime = 6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661
4545549772963113914808580371219879997166438125740282911150571 51
# Curve a = 7
# Curve b = 3337486724173056198539806914083613591148552939440918108176506133639914043582629357942467392691514112323
3307681909882108412484322514730595838550808273964305469
# Gx = 3439826180868101672186760668868680735915910517718501487265676703411036750054361952397760187383847238746825
2514885848038126069940585715650333302214273760270 71
# Gy = 2288776024351564828297492412066489392882447049893573067242139452176063822294333628193334883272224755588185
2006134343434669930190504664976927518996851698121 77
# n = 1308369257971894646462515718786143078437830269208725214468305470613365791686873293775456017611973820715965 0
3015940802370892208678895927660773409759934008499011908437644773874436659983869706278841801493292171611572887778
7679665155926916609352983403723548767363393968581227377924761110324355375043017426708159 31
# c = 8547288982820508491394895629851960626273819485040364978041497022427459572335040036069238205313904702249964 3
8518826687593416911196766750309230674387306723634640023059664362869570772826995847369010147827885102683963541519
5419921408024221820474471055514101876316654080938989953688130277615746769833625205464295
```

2.分析

（1）加密过程分析

明显c是flag经过rsa加密后，所以问题就是分解n。但是n肯定不是直接能分解的（废话。。。。）那就来看p和q的产生过程。
根据

```
if isPrime(Q.x) and isPrime(Q.y):
  print("ECC Private key:", hex(s))
  print("RSA primes:", hex(Q.x), hex(Q.y))
  print("Modulo:", hex(Q.x * Q.y))
  return (Q.x, Q.y)
```

p和q就是Q.x和Q.y。这里

```
Gx = Curve.gx
Gy = Curve.gy
G = Point(Gx, Gy, curve=Curve)
```

也就是说G是ECC的一组解。s是随机产生的ECC私钥。
那么p和q就是ECC里的公钥。

（2）理论基础

$$y = x + a*$$

（3）sage分解

用sage分解多项式。

$$f = p + a*$$

脚本

```
#!python2
# -*- coding: utf-8 -*-
# @Time : 2020/11/25 0:10
# @Author : A.James
# @FileName: exp.py
from binascii import *

P = 68647976601306097149819007990813932172694353001433054093944634591855431833976560521225596406614545549772963113914808580371219879997166438125740282911150571 51
e = 65537
a = 7
b = 33374867241730561985398069140836135911485529394409181081765061336399140435826293579424673926915141123233076819098821084124843225147305958385508082739643054 69
n = 13083692579718946464625157187861430784378302692087252144683054706133657916868732937754560176119738207159650301594080237089220867889592766077340975993400849901190843764477387443665998386970627884180149329217161157288777876796651559269166093529834037235487673633939685812273779247611103243553750430174267081593 1
c = 85472889828205084913948956298519606262738194850403649780414970224274595723350400360692382053139047022499643851882668759341691119676675030923067438730672363464002305966436286957077282699584736901014782788510268396354151954199214080242218204744710555141018763166540809389899536881302776157467698336252054642 95

#TODO:构建多项式
R.<x> = PolynomialRing(GF(P))
f = x**5 + a*x**3 + b*x**2 - n**2
#TODO：分解多项式
factors = f.factor()
print factors

#TODO：求解p
for i in factors:
    #print i
    i = i[0]
    print i
    #度为1的因式
    if i.degree()==1:
        #度为1的因式的解
        print i[0]
        #解决负数解的问题
        tmp = Integer(mod(-i[0],P))
        #tmp = Integer(i[0])
        print tmp
        print gcd(tmp,n)
        #找到p
        if gcd(tmp,n) != 1:
            p = tmp
            q = n/tmp
#TODO：常规模式rsa
print p,q
phi = (p-1)*(q-1)
d = inverse_mod(e,phi)
m = pow(c,d,n)
print unhexlify(hex(Integer(m)))
```

4.get flag

```
flag{fake_ECC_but_polynomial_factor}
```

结语

天知道发生了什么。。。