Marvin is plain Jane WriteUp_实验吧_Crypto

转载

FrancisQiu 于 2019-03-10 23:23:49 发布 577 收藏

分类专栏： CTFwriteup crypto CTF 密码学 实验吧

 CTFwriteup 同时被 3 个专栏收录

8 篇文章 0 订阅

订阅专栏

 crypto

5 篇文章 0 订阅

订阅专栏

 CTF

7 篇文章 0 订阅

订阅专栏

Marvin is plain Jane WriteUp

由于本人并没有做出来，因此本WP参考了文章：

> https://stratum0.org/blog/posts/2013/10/26/hack-dot-lu-2013-marvin-is-plain-jane/
> http://www.shiyanbar.com/ctf/writeup/6162

题目：

> Hey mister super-duper robo-dabster. We need you to tell us, what Marvin is!
> What we know:
> Marvin is
> using brainpool p256r1.
> His friend is called meneze or something. Or was it van-stone?
> What we heard:
> (233720930783175516652161591397844134118067532292492016816473888277548274528 56 : 1)
> 71164450240897430648972143714791734771985061339722673162401654668605658194656
> 12951693517100633909800921421096074083332346613461419370069191654560064909824
> What we need to know:
> What Marvin is

思路分析：

本题中，根据“His friend is called meneze or something. Or was it van-stone?”，明显提示了“van-stone”这个词。根据密码学相关内容推断，本题使用了Menezes-Vanstone椭圆曲线加密算法， 因此本题考查了椭圆曲线加密。

其中相关数据为：

> x1 = txt(“Marvin is”)
> y1 = 71164450240897430648972143714791734771985061339722673162401654668605658194656
> y2 = 12951693517100633909800921421096074083332346613461419370069191654560064909824
> p = 0xA9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377
> A = 0x7D5A0975FC2C3057EEF67530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9
> B = 0x26DC5C6CE94A4B44F330B5D9BBD77CBF958416295CF7E1CE6BCCDC18FF8C07B6

椭圆曲线加密算法相关：

1、椭圆曲线上所有的点外加一个叫做无穷远点的特殊点构成的集合，连同一个定义的加法运算构成一个Abel群。在等式Q=kP中，已知k和点P求点Q比较容易，反之已知点Q和点P求k确是相当困难的，这个问题称为椭圆曲线上点群的离散对数问题ECDLP(Elliptic Curve Discrete LogarithmProblem)。椭圆曲线加密体制正是基于这个一个困难问题。
2、Menezes-Vanstone方法：

> 有限域Zq上的非奇异椭圆曲线方程E，公开基点P，点P的阶数为q（大于160位）。设明文m=（m1，m2）∈Zq×Zq，发方为A，收方为B。
> 加密操作如下：
> （1、A、B分别选取SA，SB∈Zq*为私钥，计算PA =SAP，PB = SBP并公开；
> （2、A计算SAPB=（x1，y1）。 (1)
> （3、A计算X2=m1xi modq，
> y2=m2y1 modq。
> 则明文朋加密厉生成的密文是(X2，y2)。
> 解密操作如下：
> （1、B计算SBPA=（x1，y1）。
> （2、B计算m1=X2X1-1modq，
> m2 = y2y1-1 modq 。
> 则密文（x2，y2）解密后的明文是(m1，m2)。

解题脚本：

```
# -*- coding: utf-8 -*-
def txt(istr):
 return int(istr.encode("hex"),16)
x1 = txt("Marvin is")
y1 = 71164450240897430648972143714791734771985061339722673162401654668605658194656
y2 = 12951693517100633909800921421096074083332346613461419370069191654560064909824
p = 0xA9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377
A = 0x7D5A0975FC2C3057EEF67530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9
B = 0x26DC5C6CE94A4B44F330B5D9BBD77CBF958416295CF7E1CE6BCCDC18FF8C07B6

# from: http://eli.thegreenplace.net/2009/03/07/computing-modular-square-roots-in-python/
def modular_sqrt(a, p):
 """
 求解二次同余： x^2 = a (mod p)
    返回解：x、p-x或者为0
 使用的求解算法：Tonelli-Shanks algorithm
 """
 # 简易情况：
 #
 if legendre_symbol(a, p) != 1:
  return 0
 elif a == 0:
  return 0
 elif p == 2:
  return p
 elif p % 4 == 3:
  return pow(a, (p + 1) / 4, p)
```

```
 # Partition p-1 to s * 2^e for an odd s (i.e.
 # reduce all the powers of 2 from p-1)
 #
 s = p - 1
 e = 0
 while s % 2 == 0:
  s /= 2
  e += 1

 # Find some 'n' with a legendre symbol n|p = -1.
 # Shouldn't take long.
 #
 n = 2
 while legendre_symbol(n, p) != -1:
  n += 1

 # information
 #

 # x is a guess of the square root that gets better
 # with each iteration.
 # b is the "fudge factor" - by how much we're off
 # with the guess. The invariant x^2 = ab (mod p)
 # is maintained throughout the loop.
 # g is used for successive powers of n to update
 # both a and b
 # r is the exponent - decreases with each update
 #
 x = pow(a, (s + 1) / 2, p)
 b = pow(a, s, p)
 g = pow(n, s, p)
 r = e

 while True:
  t = b
  m = 0
  for m in xrange(r):
   if t == 1:
    break
   t = pow(t, 2, p)

  if m == 0:
   return x

  gs = pow(g, 2 ** (r - m - 1), p)
  g = (gs * gs) % p
  x = (x * gs) % p
  b = (b * g) % p
  r = m

# from: http://stackoverflow.com/a/9758173
def legendre_symbol(a, p):
 #求（a, p）的勒让德符号
 ls = pow(a, (p - 1) / 2, p)
 return -1 if ls == p - 1 else ls


def egcd(a, b):
```

```
    #求解欧几里得算法逆过程：
 if a == 0:
  return (b, 0, 1)
 else:
  g, y, x = egcd(b % a, a)
  return (g, x - (b // a) * y, y)

def modinv(a, m):
    #调用欧几里得算法得到逆过程方程参数
 g, x, y = egcd(a, m)
 if g != 1:
  raise Exception('modular inverse does not exist')
 else:
  return x % m

def gety(x):
    #求解y、y2
 y = modular_sqrt(x**3 + A * x + B, p) % p
 y2 = -y % p
 return y,y2

def hextotext(nbr):
    #hex转化成string
 s = hex(nbr)[2:-1]
 if len(s) % 2 ==1:
  s = "0"+s
 return s.decode("hex")


x1_inv = modinv(x1, p)
c1 = (y1 * x1_inv) % p
c2_1, c2_2 = gety(c1)

print repr(hextotext(y2*modinv(c2_2, p)  % p))
```