D^3CTF 2019 KeyGenMe WP

那道题稍微看了一下，无符号、大数运算，strings出了一些字符串，估计是用了库。

搜github找到用的是miracl，开始了漫长的恢复符号表之旅...试了rizzo、flair，恢复率很低...

后来搜某字符串搜到了源码...

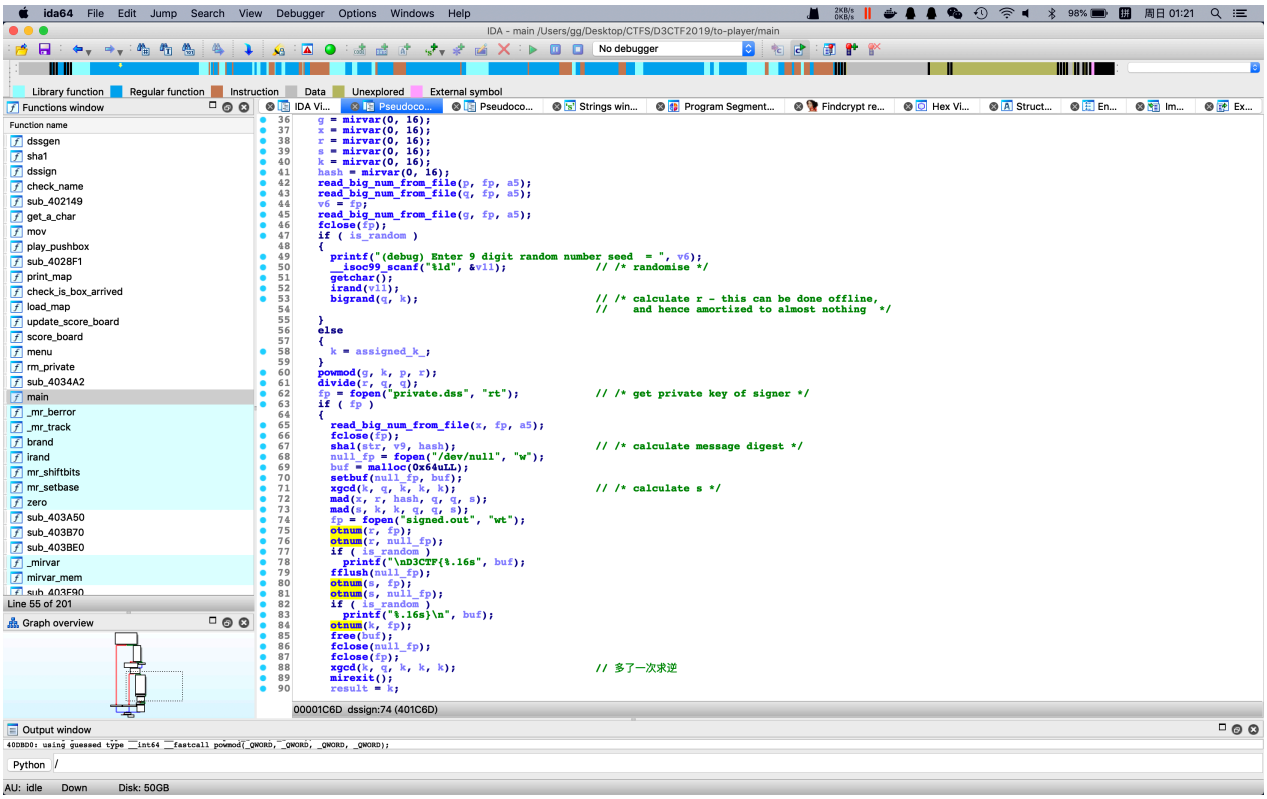
https://github.com/BingyZhang/EC-ElGamal/blob/252d57cae7a72019fbc08ce644b60a54014761be/dssver.c

标记对应的函数，看与标准实现之间的区别。

发现在生成的时候，dssign签名了两次，

```
unsigned __int64 __usercall sub_4034A2@<rax>(int a1@<r12d>)
{
  int len_1; // eax
  __int64 v2; // ST08_8
  int len_2; // eax
  char flag_str[4]; // [rsp+10h] [rbp-20h]
  char str_d3ctf[4]; // [rsp+20h] [rbp-10h]
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  dssetup();
  dssgen(a1);
  strcpy(flag_str, "flag");
  strcpy(str_d3ctf, "d3ctf");
  len_1 = strlen(flag_str);
  v2 = dssign(flag_str, len_1, 1, 0LL, a1);
  len_2 = strlen(str_d3ctf);
  dssign(str_d3ctf, len_2, 0, v2, a1);
  rm_private();
  return __readfsqword(0x28u) ^ v7;
}
```



发现k重复使用，且输出到了signed.out

根据DSA签名算法

签名 ¶

签名步骤如下

1. 选择随机整数数 k 作为临时密钥，$0 < k < q$ 。
2. 计算$r \equiv (g^k \bmod p) \bmod q$
3. 计算$s \equiv (H(m) + xr)k^{-1} \bmod q$

签名结果为 (r,s)。需要注意的是，这里与 Elgamal 很重要的不同是这里使用了哈希函数对消息进行了哈希处理。

其他信息都有，我们可以恢复出x了。然后根据x求出'flag'的s

```
from Crypto.Util.number import *
import hashlib

def hex2int(x):
    return int(x, 16)

# common.dss
```

```
p =
hex2int("DECD51284F2A90629A979EC21D02831B22A3B91C48A914A50DA604F5136F52E108179
821885A7D8F78C248D25953EB70B84DCA4AC037891BB8DB58C3F4AFCFB87ACBA48065E9982DA03
C6CD89C57A8A4133A66DD28C0E9EDC4D2FC0B4F328C93D81078428C6A27A971B5A61058F021186
504157F5FDCCE4ACCBD7EC0B7244BE7")
q = hex2int("74B097D575A3FADAA4392962FEAB0566922E47CB")
g =
hex2int("3E01480D6DE590269FC9D27322E27791979DE767BD9EB57FB16D2884393D1E89C0ED0
9413611F810CA02230CD818D4ECFEC38969F9722202627DF695E4256E7FC237685C9F88AEFF31D
0B0FEA1ACF1A7E93A28442E7E1419D2BA66AB034BE3F70F2F813F116FE2D184A0494DD516C8ABC
02786B34B0EBEB57ADEB8369CE6D1A0")

# public.dss
y =
hex2int("44ED3FB1DD0BD0BDB9A15CD817A8AC6820AB189C33468FA74F9180842B0F6B2B17EEB
CE91C670AAA71F44643EA128709A0BEAC51219B80452C6C670B07DD99F7F92AAC19345F4DB94BC
58DF3BCAF1C8BAF90514AC18F4BAB13D1E3BED6DB1C9321EE12FEFD71C8BEE1CACC9F64F043708
A6ED90B8B6295BA07ECDAE2C9CC58D3")

# signed.out
r = hex2int("68C0403C1A7495A678B519F5E048D37D7DC87E8")
s = hex2int("189A6502999B68C09F06385054DFD444F33F3EA5")
k = hex2int("6E08556F52B1256DDF51517D7F2FA0B2650A89FE")


k = inverse(k, q)

# hash('d3ctf')
md1 = int(hashlib.sha1('d3ctf').hexdigest(), 16)

x = (s * k - md1) * inverse(r, q) % q
print "x: " + hex(x)

# check origin r
print "old r: " + hex(pow(g, k, p) % q)
# check s
print "old s: " + hex((md1 + x * (pow(g, k, p) % q)) * inverse(k, q) % q)

# k = inverse(k, q)

# hash('flag')
md2 = int((hashlib.sha1('flag').hexdigest()), 16)

r = pow(g, k, p) % q
s = (md2 + x * r) * inverse(k, q) % q

print "r: " + hex(r)
print "s: " + hex(s)
print "k: " + hex(k)
```

```
# x: 0xf7225f04b1ff287ea82522d6791f80e3ce238fdL
# old r: 0x68c0403c1a7495a678b519f5e048d37d7dc87e8L
# old s: 0x189a6502999b68c09f06385054dfd444f33f3ea5L
# r: 0x68c0403c1a7495a678b519f5e048d37d7dc87e8L
# s: 0x2afa13f8ad3a9c362b1678a0f7f188e5b735976dL
# k: 0x2537954df1e74bb0f6ef9b710d39ee4cd2296a06L
```