This writeup explains how I was able to complete a crypto challenge during the 2022 CrikeyCon CTF.

Setup

The zip file for the challenge contained three files:

```
$ cat flag.enc
0x089f3beba3fa4caf6627f0229b870f80af3cf9307b652bc064a45394566252bc800702fdfe6a7be20ca12db833b

$ xxd AAAAAA.der
00000000: 3082 04a5 0201 0002 8201 0100 d807 c986  0...............
00000010: cf04 2ab4 2a0e 2a24 a1b8 44ea be79 bd3e  ..*.*.*$..D..y.>
00000020: 66b8 f4dc 753a 283f 0d59 bd63 9256 1f34  f...u:(?.Y.c.V.4
00000030: d26a aaaa aaaa aaaa aaaa aaaa aaaa aaaa  .j..............
00000040: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000050: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000060: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000070: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000080: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000090: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000000a0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000000b0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000000c0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000000d0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000000e0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000000f0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000100: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000110: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000120: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000130: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000140: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000150: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000160: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000170: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000180: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000190: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000001a0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000001b0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000001c0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000001d0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000001e0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000001f0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000200: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000210: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000220: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000230: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000240: aaaa f9d3 e7fa d7b2 5117 d047 88d9 d049  ........Q..G...I
00000250: 3ec9 81db d24d cbe4 e565 20a3 addc 1b9c  >....M...e .....
00000260: d0ee 806f 9cab 3830 f69e 8e05 4958 da48  ...o..80....IX.H
00000270: 2235 297f 7df1 5875 a509 285d e0f0 1d67  "5).}.Xu..(]...g
00000280: 95ae 0aea dff6 b93a 1bbe a4c5 9493 b752  .......:.......R
00000290: ef3d f9e4 24e2 f45d 15b1 0281 8100 e263  .=..$..].......c
000002a0: 42ee 56f0 3d3e 3da9 65f6 8398 cab4 4da6  B.V.=>=.e.....M.
000002b0: f71b 0ab6 d8c2 cb3a d253 8eac b31d 0baf  .......:.S......
000002c0: c292 a96f 8b74 cea2 2c7f 36a4 7064 4c43  ...o.t..,.6.pdLC
000002d0: 5d94 058b 3028 252e ab70 5c61 0832 bf0d  ]...0(%..p\a.2..
000002e0: 92c1 564d fa41 3c11 f981 1642 fe53 1990  ..VM.A<....B.S..
000002f0: 8e04 7ed3 601c b87f 60a6 e8aa a457 04c3  ..~.`...`....W..
00000300: e743 8964 8556 6544 933f 9e3d 6ef8 ac58  .C.d.VeD.?.=n..X
00000310: 7ea6 8081 6981 bfbf 16a7 2581 c7d9 0281  ~...i.....%.....
00000320: 8100 f2c0 8f33 1b49 fa3e c1e0 6299 b0b4  .....3.I.>..b...
00000330: 5794 5445 543c 8179 5f50 4590 e2c9 f616  W.TET<.y_PE.....
00000340: d327 6989 f5bd 133e cb9f 2d4e a376 c483  .'i....>..-N.v..
00000350: 414c 5dbf c841 5256 86db 2d7d 28e6 0018  AL]..ARV..-}(...
00000360: 8cd0 4a77 fc9b fea0 7d8a 93fc d57d dbcf  ..Jw....}....}..
00000370: e432 aaaa aaaa aaaa aaaa aaaa aaaa aaaa  .2..............
```

```
00000380: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000390: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000003a0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000003b0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000003c0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000003d0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000003e0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
000003f0: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000400: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000410: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000420: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000430: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000440: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000450: aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa  ................
00000460: aaaa aaaa aaaa aaa0 e20d d1e3 82bb 2728  ..............'(
00000470: d664 1756 f25e 8a7f 2e6f 91ce 905d 2295  .d.V.^...o...]".
00000480: fddc 160d 1514 1692 21a3 5bce fa1b 84ef  ........!.[.....
00000490: b92e 84da 9dc9 2dc4 966d 95c8 9382 acad  ......-..m......
000004a0: d805 d9a2 543b 5580 76                   ....T;U.v

$ xxd public.der

00000000: 3082 0122 300d 0609 2a86 4886 f70d 0101  0.."0...*.H.....
00000010: 0105 0003 8201 0f00 3082 010a 0282 0101  ........0.......
00000020: 00d8 07c9 86cf 042a b42a 0e2a 24a1 b844  .......*.*.*$..D
00000030: eabe 79bd 3e66 b8f4 dc75 3a28 3f0d 59bd  ..y.>f...u:(?.Y.
00000040: 6392 561f 34d2 6dfb 8d2d 8db5 d1f6 88cb  c.V.4.m..-......
00000050: c154 08c4 9340 7151 011d 1d35 010e f7f7  .T...@qQ...5....
00000060: 54f1 abb7 4a05 fede 3fc4 f32f d8bb 7800  T...J...?../..x.
00000070: 5dd6 1732 7272 d93e 531e 8abb 3fce d7f5  ]..2rr.>S...?...
00000080: 2cf8 2b2e 6c98 6c84 fce8 af70 6175 f2ce  ,.+.l.l....pau..
00000090: 8ecc 2975 6cfc 1c9a 2a70 04ea 6f05 aaed  ..)ul...*p..o...
000000a0: ecc7 2c4d cc67 660e 9501 0204 5035 46b4  ..,M.gf.....P5F.
000000b0: d79d cff7 472b c2c7 3f4e 7a94 13a7 4db4  ....G+..?Nz...M.
000000c0: d5c7 3591 8825 c9e9 97bc 5acf 58f5 3265  ..5..%....Z.X.2e
000000d0: c1c2 878f 3a47 0c34 e350 3f4d 9fe9 78e8  ....:G.4.P?M..x.
000000e0: dd51 19eb 15d4 3598 2965 c3b9 bd60 fd3c  .Q....5.)e...`.<
000000f0: 166b 474b 7c6d 22bb 4c78 f95d ce99 5a83  .kGK|m".Lx.]..Z.
00000100: 77a7 42d6 a05e 5346 e1a3 383a 9ecf 67c5  w.B..^SF..8:..g.
00000110: d223 c4b8 d898 42ba f535 63a3 d701 f4fa  .#....B..5c.....
00000120: 0902 0301 0001
```

From this, we can see that it appears that the private key has had many sections removed and replaced with long chains of hexadecimal As, and we have to work out how to decrypt the flag despite this limitation.

DER files

The DER file format is a way of storing certificates and private keys. Unlike other formats, (such as PEM), DER files are completely binary and do not contain plaintext information, which is why we need special tools to view them.

As always, OpenSSL is able to provide us with the assistance we need. The asn1parse command of OpenSSL can be used to read these files. Since public.der does not appear to have been changed, we will read the data from that first:

```
$ openssl asn1parse -inform DER -in public.der
    0:d=0  hl=4 l= 290 cons: SEQUENCE
    4:d=1  hl=2 l=  13 cons: SEQUENCE
    6:d=2  hl=2 l=   9 prim: OBJECT            :rsaEncryption
   17:d=2  hl=2 l=   0 prim: NULL
   19:d=1  hl=4 l= 271 prim: BIT STRING
```

This implies that the key is an RSA key. We will now decode the BIT STRING section:

```
$ openssl asn1parse -inform DER -in public.der -strparse 19
```

```
    0:d=0  hl=4 l= 266 cons: SEQUENCE
    4:d=1  hl=4 l= 257 prim: INTEGER           :D807C986CF042AB42A0E2A24A1B844EABE79BD3E66B8F
  265:d=1  hl=2 l=   3 prim: INTEGER           :010001
```

To interpret this, we will look up the conventional ASN1 grammar for RSA. In RFC3447, we find

```
RSAPublicKey ::= SEQUENCE {
    modulus           INTEGER,  -- n
    publicExponent    INTEGER   -- e
}
```

This implies that the RSA modulus is:

```
0xD807C986CF042AB42A0E2A24A1B844EABE79BD3E66B8F4DC753A283F0D59BD6392561F34D26DFB8D2D8DB5D1F68
```

And the RSA exponent is:

```
0x010001
```

Brief explanation of RSA

Before we go any further, it will be useful to give a brief overview of how the RSA public key cryptosystem works._

Key Generation

To generate keys, the party that wants to receive messages should:

1. Choose two prime numbers p and q.
2. Compute the product n = pq.
3. Compute $\phi(n) = (p - 1)(q - 1)$, where ϕ is the Euler totient function).
4. Choose an e such that $1 < e < \phi(n)$ and $\gcd(e,\phi(n)) = 1$.
5. Calculate $d \equiv e^{-1} \bmod \phi(n)$.
6. d is the private key exponent which should be kept secret, and e and n together make up the public key.

Encryption

To send an encrypted message, a sender should:

1. Find the recipient's public key, which contains values for e and n.
2. Use an agreed scheme to represent the message they would like to send as an integer m.
3. Calculate $c = m^e \bmod n$ and send this ciphertext to the recipient.

Decryption

To decrypt a message c:

1. Consider the identity $m^{\phi(n)} \equiv 1 \bmod n$ where n is relatively prime to m.
2. Observe that this identity gives us $m^{p-1} \equiv 1 \bmod p$. Similarly, $m^{q-1} \equiv 1 \bmod q$.

3. Combining these, we have $m^{(p-1)(q-1)} \equiv 1 \bmod pq$, which is equivalent to $m^{\phi(n)} \equiv 1 \bmod n$.
4. More algebraic manipulations give us $m^{k\phi(n)+1} \equiv m \bmod n, k \in \mathbb{Z}$.
5. By definition, $ed \equiv 1 \bmod \phi(n) \leftrightarrow ed = k\phi(n) + 1, k \in \mathbb{Z}$.
6. Substituting this into our previous expression, $m^{k\phi(n)+1} \equiv m^{ed} \equiv c^d \bmod n$.
7. This means that all the recipient needs to do to decrypt a message sent to them is calculate $c^d \bmod n$.

Analysing the private key

Just as in the previous section, we will try to analyse the private key using OpenSSL:

```
$ openssl asn1parse -inform DER -in AAAAAA.der
    0:d=0  hl=4 l=1189 cons: SEQUENCE
    4:d=1  hl=2 l=   1 prim: INTEGER           :00
    7:d=1  hl=4 l= 257 prim: INTEGER           :D807C986CF042AB42A0E2A24A1B844EABE79BD3E66B8F
Error in encoding
139877235652416:error:0D07207B:asn1 encoding routines:ASN1_get_object:header too long:crypto/
```

So, we can see that because of the missing sections of the file, the parser isn't working correctly. To get a better idea of what we should be expecting in terms of the file format, we again look at RFC3447:

```
RSAPrivateKey ::= SEQUENCE {
    version           Version,
    modulus           INTEGER,  -- n
    publicExponent    INTEGER,  -- e
    privateExponent   INTEGER,  -- d
    prime1            INTEGER,  -- p
    prime2            INTEGER,  -- q
    exponent1         INTEGER,  -- d mod (p-1)
    exponent2         INTEGER,  -- d mod (q-1)
    coefficient       INTEGER,  -- (inverse of q) mod p
    otherPrimeInfos   OtherPrimeInfos OPTIONAL
}
```

From some more research about how DER encoding works, we can see that the bytes "02 82 01 01" specify an integer stored in 257 bytes. The parser was expecting another similar header after this, but the header was also destroyed.

Since we already know the modulus from the public key, we will move past this section and see what other information we can gather from the private key. Since the byte "02" is the marker for INTEGER encoding, we search through the remainder of the private key file to find this, and find a match at byte 0x29a.

We re-run the asn1parse command, but this time starting from the offset 0x29a:

```
$ openssl asn1parse -inform DER -in AAAAAA.der -offset 0x29a
    0:d=0  hl=3 l= 129 prim: INTEGER           :E26342EE56F03D3E3DA965F68398CAB44DA6F71B0AB6D
  132:d=0  hl=3 l= 129 prim: INTEGER           :F2C08F331B49FA3EC1E06299B0B457945445543C8179
Error in encoding
140655835830080:error:0D07207B:asn1 encoding routines:ASN1_get_object:header too long:crypto/
```

From the RFC3447 structure, we can see that

```
0xE26342EE56F03D3E3DA965F68398CAB44DA6F71B0AB6D8C2CB3AD2538EACB31D0BAFC292A96F8B74CEA22C7F36A
```

is one of the primes used in generating the modulus!

Decrypting the flag

All we need to decrypt the flag is the modulus and a single prime. We will do this in Python below:

```
#!/usr/bin/python3
prime1 = 0xE26342EE56F03D3E3DA965F68398CAB44DA6F71B0AB6D8C2CB3AD2538EACB31D0BAFC292A96F8B74CE
modulus = 0xD807C986CF042AB42A0E2A24A1B844EABE79BD3E66B8F4DC753A283F0D59BD6392561F34D26DFB8D2
e = 0x10001
ciphertext = 0x089f3beba3fa4caf6627f0229b870f80af3cf9307b652bc064a45394566252bc800702fdfe6a7b

prime2 = modulus//prime1
totient = (prime1-1)*(prime2-1)
d = pow(e, -1, totient)
print(hex(pow(ciphertext, d, modulus)))
```

When this script is run, we get the output:

```
0x464c41477b53484f57424147537d
```

Converting this into ASCII characters, we get the flag:

```
FLAG{SHOWBAGS}
```