BearcatCTF2026 WriteUp

by zijeff

Kidds Crypto

打开附件一看，很显然啊，加密指数 $e=3$ 是很小的，所以我们考虑一下常见的 **small e** 攻击方法，比如直接开根和遍历开根。验证了一下之后发现不行，但是我们又注意到了 n 也比较小，试了一下，果然能够被完全分解，分解如下：

$$n=\prod_{i=1}^{13}p_i$$

同时，我们还发现：

$$\gcd(e,p_i-1)=e=3$$

所以，我们考虑使用**有限域开根**做，即在每个素因数下求方程的解，再用 **CRT** 组合求解，最多有 3^{13} 种可能，还是比较快得到结果的。

具体代码如下：

```
from sage.all import *
from Crypto.Util.number import long_to_bytes
import itertools

n =
1474111061582870416227845935016137758737406981820743001973218021092761520
1634788092144486300 7
e = 3
c =
1142677571544928565139938771959629860224897700091202003678114408529652390
5985415731346239935 1

# 素因子列表
primes = [
    8532679, 9613003, 9742027, 10660669, 11451571,
    11862439, 11908471, 13164721, 13856221, 14596051,
    15607783, 15840751, 16249801
]

roots_per_prime = []
print("[*] 正在计算每个素数模下的立方根...")

for i, p in enumerate(primes):
    # 在有限域 GF(p) 下求 e 次方根
    # nth_root(e, all=True) 会返回该模下所有可能的根
    try:
        # 转化为 Sage 的整数类型
```

```
        p = Integer(p)
        c_mod = Mod(c, p)

        # 求解 x^3 = c mod p
        # nth_root 返回所有根的列表
        roots = c_mod.nth_root(e, all=True)

        # 将结果转回普通整数，方便后续处理
        roots = [Integer(r) for r in roots]

        print(f"    Prime {p} (mod 3 = {p%3}): Found {len(roots)}
roots.")

        if len(roots) == 0:
            print(f"[!] 错误：在素数 {p} 下无解，无法继续。")
            exit()

        roots_per_prime.append(roots)

    except Exception as err:
        print(f"[!]计算出错：{err}")
        exit()

# 组合爆破 (CRT)
total_combinations = 1
for r in roots_per_prime:
    total_combinations *= len(r)

print("[*] 开始爆破组合并筛选 Flag...")

count = 0
found_flag = False

# 使用 itertools.product 生成所有根的组合
for roots_comb in itertools.product(*roots_per_prime):
    count += 1

    # CRT_list 是 Sage 内置函数，比手动 crt 快且方便
    # x = r1 mod p1, x = r2 mod p2 ...
    m = CRT_list(list(roots_comb), primes)

    # 转为 bytes
    m_bytes = long_to_bytes(m)
    try:
        # 尝试解码为 ASCII
        m_str = m_bytes.decode('ascii')

        # 检查是否全部由可打印字符组成
        if all(0x20 <= ord(c) <= 0x7e for c in m_str):
            print(f"[+] Found Candidate ({count}/{total_combinations}):")
```

```python
            print(f"    Str: {m_str}")

            if "CTF{" or "ctf{" in m_str: found_flag = True; break

    except UnicodeDecodeError:
        # 包含乱码，通常不是 Flag
        pass

print("[*] 完成。")
```

运行结果如图:

```
[*] 开始爆破组合并筛选 Flag...
[+] Found Candidate (104964/1594323):
    Str: BCCTF{y0U_h4V3_g0T_70_b3_K1DD1n9_Me}
[*] 完成。
```

Twisted Pair

啊，熟悉的 **RSA**，我们重点关注一下 pair 的生成过程，记其中元素为 D, E，做如下推导。

首先，我们已知如下式子成立:

$$
\begin{aligned}
\phi &= r\varphi \\
DE &\equiv 1 \mod \phi \\
de &\equiv 1 \mod \varphi
\end{aligned}
$$

进而，我们得到:

$$DE = k\phi + 1 = kr\varphi + 1 = K\varphi + 1$$

接着，我们计算一个特殊的 d'，使其满足:

$$
\begin{aligned}
ed' &\equiv 1 \mod (DE-1) \\
ed' &= 1 + m(DE-1) = 1 + mK\varphi \\
ed' &\equiv 1 \mod \varphi
\end{aligned}
$$

表明这个特殊的 d' 是和解密密钥等价的，从而我们并不需要求解原本的私钥。

具体代码如下:

```python
from Crypto.Util.number import long_to_bytes, inverse

e = 65537
```

```
n =
194395490167361579944453216937160403044969334509088228307855464473566409385118485152145321223729524413463823481280901743850870177549980042633819038841727957987578969268262604498463119101808531515382593096724925313249121315778604492700206887883661376224422610779691019368568864684274035033651705257201812007549073852032810192980133892651443534978128684013951895626347289984570806857631029024881106220421834143833761337841852275290471831457061733224084691685950670020505301600032509656507355378457201412741479346600479913794377766257739638757559228475965125419779623350566548092284489390092481396680552238435284531936 27
c =
108654862507494521428291477704827320124879697088212385166934483303494684182274596414755470811027020866775316530955771562804338330073012710647360894027359618634634994258353539383223671866313869549105713575425223528516067342671128624256753628308029308951164905073681259836680709785693845030578624313147543542501418977683323905999843762792503924104919110907453193958452168655524046835648623974291751777426526760967743372249074732536479042028404355004151582874770603651360085648907412374050763441818072932338232683808674322774892555607409859412777466688799746879629699213229154496816285701663956601160636675384376431917 78
```

```python
pair =
(787659315728378901847324855738550403557349066676167323654259759613721767
3449090876058614021324769584425816469011385676444996263350751408687085597
7086816171599249980551485754833342744585856747583660831419118402998816441
0311865292559161917784860025999883657833012008840746285622713539300521002
6809140160100135202940618182077923566002250650074432348355719020221456485
2496130042887915901484203538953743898572705208120100144410589773705113057
0431198008338321110603144288623275203357262214458710131793541826457046769
6233867679311457516362160176776050716530172188662425182443285823761225420
5394349688833372550507942811289139413264033829882510339925802192503391563
4440457039099872182278450963934788385352690708170598264630847369610895729
7801438243003578131451520832474703622986804229078485203304287455860443222
4005163718694172245343564311670051223512443733554163748501818786399835388
4051064276366623132961559570348860402351397818790669609760912865119380177
8455004869881947996739890529901318613579454613730106304382929543992095447
5639248584023875352900082592641833682772678691258949438442520763882466710
2214805279464558488708854392657444595333096712723633939618134980322402939
23368716112266816632737047499214760445112990128047541791833572827,
1319706040894277810931686068376736369926300154709149361031239204132078529
2670988925664638342683661554015926284641801175088273463019331735285060032
2481025539933491985114268193122631390578816208240929415962476199734150525
1024089415920183080438925951000160340819738717386652432697815506162152117
9296883159621119079434392945282458104640771598948552745225621896863196137
5664818092224014538346509000511329897864091996499571026139363240809226838
9031486695429612667557235032752659681591301252147351235782688455423784123
6175012931996972061495801750037935059967354308251368865571004611146028641
8503919155149746284910623921263483191314244456818012541093775511309319094
0663272720657864269981040159865171239696418937738638263395454560496970910
1983483373008625052431124973346424033571286909188910380586734124040489812
6314487597920288551355801575807245535302964160133333369247197055552045510
1473102549047755570900115358053077678926727745553760874799679702729262180
7642412447721177658752615125664107284786499808211869276279440959337839734
9174173985530922396585882206274522841399151027762258839530878512906205295
1985729346228575950562903375619826089961884936117151712872287560063571702
86646088568403471562908235292011333236920822539972541202731306163)

def solve():
    re, rd = pair
    # 核心原理: re * rd - 1 是 phi 的倍数
    # 令 mul_phi = k * phi
    mul_phi = re * rd - 1


    # 求 e 关于 mul_phi 的逆元
    d_fake = inverse(e, mul_phi)
    # print(d_fake)
    # 直接解密
    m = pow(c, d_fake, n)
    # 转换为字节并打印 Flag
    flag = long_to_bytes(m)
```

```
        print(f"[+] Flag found: {flag.decode(errors='ignore')}")

if __name__ == '__main__':
    solve()
```

运行结果如图:

```
[+] Flag found: BCCTF{D0n7_g37_m3_Tw157eD}
smith@jeff-ciallo:~/Tasks/BearcatCTF2026/Twisted_Pair$
```

Pickme

远程题目来了，服务器端脚本理解一下逻辑。简单来说就是，我们上传一个 RSA 加密的私钥，服务器会用这个私钥加密 flag ，然后解密验证，当验证结果不相等的时候输出密文。

所以我们的目标在于构造特殊的私钥，使服务器能够输出密文，从而解密密文得到答案。显然，我们考虑更改加密中的条件，令:

$$p = q$$

那么，ϕ 的计算就变为:

$$\phi = p(p-1) \neq (p-1)^2$$

因此，题目验证结果不会相等，而私钥是我们已知的，进而解出。需要注意的是构造过程由于不符合标准加密过程，需要我们手动生成私钥文件。

构造代码如下:

```
import base64
from Crypto.Util.number import *

# --- ASN.1 编码辅助函数 ---
def encode_len(length):
    if length < 0x80:
        return bytes([length])
    else:
        s = hex(length)[2:]
        if len(s) % 2: s = '0' + s
        b = bytes.fromhex(s)
        return bytes([0x80 | len(b)]) + b

def encode_integer(n):
    b = long_to_bytes(n)
    # ASN.1 INTEGER 是有符号的，如果最高位是1，需要补00
    if b[0] & 0x80:
        b = b'\x00' + b
    return b'\x02' + encode_len(len(b)) + b

def encode_sequence(content):
```

```python
    return b'\x30' + encode_len(len(content)) + content

def generate_malicious_pem():
    print("[*] Generating malicious p=q key manually...")
    e = 65537
    # 题目要求 p >= 512 bits
    p = getPrime(512)
    q = p  # 设置 p = q
    n = p * q

    # 1. 计算能骗过服务器的 d
    # 服务器逻辑: phi = (p-1)*(q-1) = (p-1)^2
    phi_fake = (p - 1) * (q - 1)
    d_fake = inverse(e, phi_fake)

    # 2. 计算 CRT 参数 (用于通过服务器检查)
    dmp1 = d_fake % (p - 1)
    dmq1 = d_fake % (q - 1)
    # iqmp = q^-1 mod p。因为 p=q，不存在逆元。
    # 但服务器只检查 dmp1/dmq1，不检查 iqmp。随便填一个。
    iqmp = 1

    # 3. 手动构造 ASN.1 序列
    # RSAPrivateKey ::= SEQUENCE {
    #   version           Version, (0)
    #   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, (inv(q) mod p)
    # }
    content = b''
    content += encode_integer(0)        # Version
    content += encode_integer(n)        # n
    content += encode_integer(e)        # e
    content += encode_integer(d_fake)   # d (fake)
    content += encode_integer(p)        # p
    content += encode_integer(q)        # q
    content += encode_integer(dmp1)     # dmp1
    content += encode_integer(dmq1)     # dmq1
    content += encode_integer(iqmp)     # iqmp

    der = encode_sequence(content)

    # 4. 封装成 PEM
    b64 = base64.b64encode(der).decode()
    pem = "-----BEGIN RSA PRIVATE KEY-----\n"
```

```python
    # 每 64 字符换行
    for i in range(0, len(b64), 64):
        pem += b64[i:i+64] + "\n"
    pem += "-----END RSA PRIVATE KEY-----"

    return pem.encode(), p, n

pem, p, n = generate_malicious_pem()
print(p)
print(pem.decode())
```

运行结果如下图:

```
[*] Generating malicious p=q key manually...
8664599897181828105768700145910830040465303865998233185749388220210735896059664821298925145211253607
-----BEGIN RSA PRIVATE KEY-----
MIICHQIBAAKBgGrpJ88ms5wk7DyXcug2TPjROXKKS7eWRxNdHKbaXU9sbn6BqL4x
Z8kf4nbkaSRAzcgPVC/3//HOr1Tdf18etK4spq7akTY2TOR6SvXiTWt3uPmX5fun
yiIU9G8VXucOtDCxxEsXxibGJnVQII82LVSP6/grl6NhIW4e4sPcA34hAgMBAAEC
gYAEsKPBb6YzdF80rUQqMa+gvrTxdGWr8Ri/HbeTDkNHp68VN6TqawpthUuPAagn
IfylCR8dV5jPx8xjQbBDwPoWF3WB/DZVcwYfTRPDmjaUma+LHYpHmeNnMi7Cyih4
Tnb97RpA16ypzNCQVi7UjVgNyl3eEmYdSoP+4sZP1YwkrQJBAKVvrpzoXW1yB5m0
RpXfnfKHDyLLI2WGdED5U+fbmJ5EcKyThhKhvcAb/kB0PijRmx/+3pmSfSjORVFI
INheyW8CQQClb66c6F1tcgeZtEaV353yhw8iyyNlhnRA+VPn25ieRHCsk4YSob3A
G/5AdD4o0Zsf/t6Zkn0ozkVRSCDYXslvAkEAhKm8cmL7mKpaYe1otQKvHC8l+ha/
+cgKLZhCqqk0B6aGXD2oxAQjr4xDJNiSiBUdulidlLUd3L0cZQPcIuZPzQJBAISp
vHJi+5iqWmHtaLUCrxwvJfoWv/nICi2YQqqpNAemhlw9qMQEI6+MQyTYkogVHbpY
nZS1Hdy9HGUD3CLmT80CAQE=
-----END RSA PRIVATE KEY-----
```

上传服务器后，我们得到密文回显如下图:

```
smith@jeff-ciallo:~$ nc chal.bearcatctf.io 56025
Never worry about your RSA keys again!
Let me test them for you

Enter your key in pem format:
-----BEGIN RSA PRIVATE KEY-----
MIICHQIBAAKBgGrpJ88ms5wk7DyXcug2TPjROXKKS7eWRxNdHKbaXU9sbn6BqL4x
Z8kf4nbkaSRAzcgPVC/3//HOr1Tdf18etK4spq7akTY2TOR6SvXiTWt3uPmX5fun
yiIU9G8VXucOtDCxxEsXxibGJnVQII82LVSP6/grl6NhIW4e4sPcA34hAgMBAAEC
gYAEsKPBb6YzdF80rUQqMa+gvrTxdGWr8Ri/HbeTDkNHp68VN6TqawpthUuPAagn
IfylCR8dV5jPx8xjQbBDwPoWF3WB/DZVcwYfTRPDmjaUma+LHYpHmeNnMi7Cyih4
Tnb97RpA16ypzNCQVi7UjVgNyl3eEmYdSoP+4sZP1YwkrQJBAKVvrpzoXW1yB5m0
RpXfnfKHDyLLI2WGdED5U+fbmJ5EcKyThhKhvcAb/kB0PijRmx/+3pmSfSjORVFI
INheyW8CQQClb66c6F1tcgeZtEaV353yhw8iyyNlhnRA+VPn25ieRHCsk4YSob3A
G/5AdD4o0Zsf/t6Zkn0ozkVRSCDYXslvAkEAhKm8cmL7mKpaYe1otQKvHC8l+ha/
+cgKLZhCqqk0B6aGXD2oxAQjr4xDJNiSiBUdulidlLUd3L0cZQPcIuZPzQJBAISp
vHJi+5iqWmHtaLUCrxwvJfoWv/nICi2YQqqpNAemhlw9qMQEI6+MQyTYkogVHbpY
nZS1Hdy9HGUD3CLmT80CAQE=
-----END RSA PRIVATE KEY-----
Key looks good, testing encryption capabilities...
An error occurred:
Some unknown error occurred! Maybe you should take a look: 676853105382496763523993258343573133039198054208201582833439304870479173659557928690176583855874819969152215836044767761557180365498932324698987238178760090787711339252961447066312721433517405997302838007047218126274348030986135526889520357371666827839193871736877105329796190120119035926338275573285373702 70
```

解密代码如下:

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

# p = q
e = 65537
p =
866459989718182810576870014591083004046530386599823318574938822021073589605966482129892514521125360764548752057211944430781292382434281643895079523521572 7
n = p * p
phi = p * (p - 1)
c =
676853105382496763523993258343573133039198054208201582833439304870479173659557928690176583855874819969152215836044767761557180365498932324698987238178760090787711339252961447066312721433517405997302838007047218126274348030986135526889520357371666827839193871736877105329796190120119035926338275573285373702 70
d = inverse(e, phi)
m = pow(c, d, n)
print(long_to_bytes(m))
```

运行得到最终答案:

```
b'BCCTF{R54_Br0K3n_C0nF1rm3d????}'
```