【CTF WriteUp】201909广东强网杯部分题解

原创

零食商人 于 2019-09-10 16:23:11 发布 1678 收藏 2

文章标签： WriteUp

本文链接：https://blog.csdn.net/cccchhhh6819/article/details/100702691

Crypto

老王的秘密

根据题目提示区块链与数据格式，本题为秘密共享，使用github下现成库可解决：

https://github.com/blockstack/secret-sharing

解题代码如下：

```
shares=['1-fddc7d57594928fb74a507ab9cba0b28b92bb6e7b36a9925a105eeddac020e64',
'3-84f82314003c9690eeacd823b22680ccbe93ac098cabdd0a992c095dde0031cf',
'5-b0e2e8d2cadc91f8f2f357a42e26aeabaccbfa7731437298ca23d8a4a5424ce4',
'7-810e7545213971a3c7c2dce3d0998764d0bc1e3b866b15ad0deebaa7abcf64c5',
'9-b4da0bd03394e4bdfef92f16365e8811d9614f11b99111bcf8a4e68ba79626a2',
'b-661069e7d491719759a3199be1f65ffb6db92d1b014abb4e33ca7e32f85ee276',
'd-1f84ab9b467a4ec4de4451ed187987785b567bbdde0126d0722e3335a5307d68',
'f-9001dc36dd28c5c5dd7333968e7263986f55dd79cd9be286d21f45e46f53c399']

from secretsharing import PlaintextToHexSecretSharer

PlaintextToHexSecretSharer.recover_secret(shares[0:8])
```

遗失的秘密

本题为PCTF原题改编，解题原理如下：

```
* given a candidate for (p mod 16**(t - 1)), generate all possible candidates for (p mod 16**t) (check against m
ask for prime1)
* calculate q = n * invmod(p, 16**t) (and check against mask for prime2)
* calculate d = invmod(e, 16**t) * (1 + k * (N - p - q + 1)) (and check against mask for private exponent)
* calculate d_p = invmod(e, 16**t) * (1 + k_p * (p - 1)) (and check against mask for exponent1)
* calculate d_q = invmod(e, 16**t) * (1 + k_q * (q - 1)) (and check against mask for exponent2)
* if any of checks failed - check next candidate
```

利用公钥和私钥内的信息可以拼出n，但是这个n是不完整的，其后8位为空。幸好我们知道p和q的后8位，因此可以计算出完整的n。解题代码如下：

```
#!/usr/bin/env python
#-*- encoding: utf8 -*-

import re
import pickle
from itertools import product
from libnum import invmod, gcd


def solve_linear(a, b, mod):
```

```
    if a & 1 == 0 or b & 1 == 0:
        return None
    return (b * invmod(a, mod)) & (mod - 1)  # hack for mod = power of 2


def to_n(s):
    s = re.sub(r"[^0-9a-f]", "", s)
    return int(s, 16)


def msk(s):
    cleaned = "".join(map(lambda x: x[-2:], s.split(":")))
    return msk_ranges(cleaned), msk_mask(cleaned), msk_val(cleaned)


def msk_ranges(s):
    return [range(16) if c == " " else [int(c, 16)] for c in s]


def msk_mask(s):
    return int("".join("0" if c == " " else "f" for c in s), 16)


def msk_val(s):
    return int("".join("0" if c == " " else c for c in s), 16)


E = 65537

N = to_n("""00:c4:9d:36:a4:77:76:12:12:85:24:6c:74:1d:7d:
    b3:ce:f4:c3:a4:69:cd:0b:2e:8f:d6:75:e3:80:b8:
    e8:1c:ce:e8:60:90:45:56:73:ab:32:32:00:7f:6a:
    76:3e:b6:10:d3:a2:74:da:f9:4e:a5:7e:ae:ef:f4:
    da:82:57:6d:68:82:50:d8:b1:fc:92:b1:5c:7d:54:
    f5:7e:d0:06:8a:60:ff:82:70:72:20:68:4b:71:ba:
    87:44:57:c1:97:a0:8a:2d:53:93:f3:0a:60:87:a3:
    85:c8:45:e6:0a:88:85:b5:ff:c7:09:9a:76:03:fe:
    99:b6:fb:8a:1e:9f:a8:42:3a:0a:c9:a9:bf:1c:87:
    2c:c4:99:10:db:46:e3:a9:a5:79:93:8c:75:71:ec:
    c6:3b:af:44:dc:60:c4:53:f6:3c:e8:73:2f:50:10:
    38:e7:6f:d0:a5:4b:ae:e3:1e:43:11:42:2c:a2:38:
    e6:3f:0b:13:54:63:e8:2f:9e:61:ab:08:65:97:e0:
    27:30:19:fd:a7:fe:5c:d8:11:b8:34:87:ad:02:c2:
    bc:cd:73:d3:86:be:fd:2a:b4:fe:7d:7e:d3:64:bb:
    6f:63:ed:a6:1d:ee:f2:80:da:9d:7a:23:7f:c1:39:
    b0:98:0c:85:8f:d0:4b:9f:e4:1a:26:fc:44:d1:67:
    03:32:03:0c:91:61:23:4c:81:6f:42:18:88:41:dc:
    27:55:a3:07:7c:a1:ad:f3:58:4d:91:07:65:f1:63:
    f2:34:d5:17:0e:59:c6:bb:b6:6d:7d:0c:d2:64:4b:
    b9:9c:52:59:03:8e:2a:43:23:76:33:c3:e8:72:3b:
    1c:e0:40:97:36:5f:ae:00:d7:e3:09:eb:df:55:44:
    22:b4:09:00:b5:09:41:70:6c:5c:3b:98:d3:34:7e:
    60:a2:b8:93:bd:af:32:77:48:48:8a:a5:9c:0e:6a:
    a1:79:36:86:8c:e9:3f:b1:a2:a7:4a:3a:d8:d6:f6:
    dd:62:d8:ae:9e:13:bb:0c:6b:b1:65:68:0d:7e:58:
    3f:68:1e:91:49:13:19:68:2b:fd:3c:5e:52:fa:76:
    b0:57:fc:0e:35:d8:71:56:41:06:ef:50:99:56:dd:
    d4:9a:1f:d3:46:26:12:9c:15:4b:43:fc:1b:de:c9:
    06:ad:82:56:63:c8:a4:83:32:d2:35:05:23:15:52:
```

```
    d9:0a:73:85:5e:c9:c2:56:af:69:d2:5f:77:04:28:
    c8:4c:b9:a6:d4:15:15:b5:15:99:13:ef:a9:a5:de:
    5a:74:b1:03:cf:32:a5:03:69:f8:e9:bb:7e:16:31:
    5e:43:e7:02:51:ac:c5:f6:bf:ef:1c:74:f7:13:0c:
    19:ad:b7""")
# prime1 p
p_ranges, pmask_msk, pmask_val = msk("""00:  :05:89:  :bd:35:  :  :23:  :  :  :  :84:
      :  :ed:  :70:14:  :  :  :10:  :  :87:  :51:
    ea:  :97:69:  :52:  :  :  :  :  :ea:  :  :15:
      :  :34:  :be:11:23:  :  :34:14:  :94:  :10:
      :  :74:87:37:ee:81:62:ee:95:  :  :dc:49:dd:
      :  :35:  :81:  :fa:  :  :  :86:  :  :  :fb:
      :93:  :  :12:  :14:  :ab:76:  :96:  :  :27:
      :21:  :04:01:41:  :98:  :ff:  :  :12:dc:  :
    cd:  :39:95:30:  :47:  :fa:ff:  :34:  :ad:  :
      :52:02:fa:bc:14:22:22:48:61:62:bd:53:  :  :
    72:08:cb:41:88:  :  :  :63:91:30:fe:  :  :42:
    87:  :18:52:  :39:dd:  :68:  :fe:06:88:81:  :
      :  :  :ae:fd:  :  :fb:21:37:59:  :53:  :fa:
      :07:40:eb:33:77:51:64:10:dd:  :73:  :86:62:
      :bf:  :79:  :34:  :bb:  :44:ff:  :46:fe:90:
    ef:  :52:ad:  :  :fe:  :69:18:89:bd:cd:09:46:
      :  :74:71:  :  :  :41:66:  :  :11:  :25:  :
    39:8b""")
# prime2 q
q_ranges, qmask_msk, qmask_val = msk("""00:ce:43:ef:  :76:58:17:43:31:  :  :32:70:  :
    89:  :  :36:55:06:  :79:66:78:  :  :  :  :  :
      :85:  :  :  :  :  :33:bb:  :  :56:  :66:cb:
      :08:  :  :90:cb:  :  :24:fa:ca:47:  :  :  :
      :88:  :83:01:  :62:  :  :  :  :  :  :ad:ae:
      :  :  :58:  :ec:  :  :  :09:04:86:  :05:00:
      :df:50:84:81:80:  :ae:  :24:  :94:da:  :04:
    ce:  :ef:  :  :ed:be:bf:43:78:  :  :05:93:  :
    08:52:05:  :  :  :  :ae:  :  :  :  :ab:  :  :
      :76:ce:  :  :  :  :19:bd:22:  :ef:dc:bf:ea:
    ab:78:01:  :  :85:  :  :  :ea:  :  :fb:  :  :
    92:66:19:  :  :ab:  :  :82:  :  :31:  :  :da:
    82:  :13:82:43:  :  :94:13:41:  :  :  :37:  :
      :04:56:02:87:dd:  :58:27:  :  :24:  :  :  :
    28:  :  :09:14:89:  :  :  :49:59:  :16:eb:65:
      :01:22:  :  :dd:  :78:  :  :db:90:  :ac:  :
      :fd:  :03:74:  :  :  :  :92:  :00:ba:  :  :
      :05""")
# privateExponent d
_, dmask_msk, dmask_val = msk("""11:  :  :69:62:64:  :  :  :  :15:  :13:de:de:
    cf:  :  :17:  :  :75:  :98:42:fc:  :12:15:08:
      :  :  :  :  :  :36:  :be:25:48:  :  :19:  :  :
      :47:11:19:  :03:  :49:fc:da:  :96:45:eb:  :
      :  :  :91:  :ea:  :  :55:ff:  :37:58:  :  :
    19:  :  :73:40:  :91:15:01:da:91:22:fd:32:  :
      :  :50:  :  :66:  :  :  :42:  :  :ef:  :  :
    df:42:  :97:30:  :39:  :  :  :  :  :  :dc:  :
      :  :  :  :  :  :38:  :  :  :88:28:  :05:  :  :
    78:59:fa:  :86:  :19:24:  :  :  :  :da:cf:15:
    39:  :  :  :  :ef:55:  :ce:47:  :58:89:  :fb:
      :24:  :  :  :92:  :  :ee:  :  :db:67:31:ce:
      :28:  :72:ec:89:  :04:  :  :50:  :  :  :  :
      :37:  :44:  :  :  :  :56:  :38:  :bb:47:bb:
    66:83:99:22:07:72:  :  :48:52:02:  :  :  :29:
      :82:56:  :67:  :95:  :  :56:94:  :  :71:  :
```

```
    bf:27:98:  :  :54:98:26:06:87:  :ae:  :53:be:
      :  :80:37:60:61:ea:ef:de:  :  :df:90:81:  :
    70:  :06:33:26:  :75:fe:95:  :92:  :78:cd:05:
    64:cc:68:  :  :36:54:  :bd:16:90:ee:60:  :  :
      :  :41:  :  :91:  :79:58:06:50:  :46:  :  :
    45:  :09:ca:ac:16:  :27:98:  :  :ba:82:  :77:
    93:98:ad:  :15:  :67:53:97:ad:ee:50:44:  :31:
    07:  :ff:01:  :09:  :  :  :  :  :46:  :  :42:
    15:  :db:df:42:be:  :  :  :78:  :41:  :  :  :
      :14:  :  :25:fc:  :84:  :  :  :  :  :  :20:
    da:46:01:eb:87:  :12:57:  :  :56:af:  :87:93:
    60:  :02:  :18:89:63:72:ad:  :ed:cf:  :  :84:
      :22:  :13:  :  :dd:  :ff:  :  :  :de:62:37:
      :19:66:  :  :86:02:  :38:  :  :  :  :ec:14:
    12:  :43:93:19:65:98:  :  :03:  :  :  :ef:  :
      :  :ca:07:92:22:  :  :bb:15:eb:  :  :  :35:
      :72:29:cd:  :  :99:  :  :  :  :41:06:  :  :
      :43:33:  :32:  :  :54:be:92:62:  :78:59:42:
    79:89"""")
# exponent1
_, dpmask_msk, dpmask_val = msk("""  :39:  :28:16:02:89:ce:11:fe:  :  :  :  :af:
      :  :  :ed:97:  :  :11:20:ba:ae:98:ad:  :  :
      :10:87:ac:07:  :  :  :  :50:  :  :70:50:52:
    df:89:eb:02:  :  :  :  :93:11:  :  :12:  :56:
      :08:  :  :ea:  :10:fa:19:  :  :  :54:45:07:
      :  :bc:ff:33:  :db:63:49:fe:52:  :33:  :  :
    bf:cd:45:91:  :10:  :  :92:81:40:03:  :80:  :
    29:  :30:  :ed:43:64:ca:  :bf:64:  :  :bf:  :
      :  :  :24:72:84:  :  :ff:  :  :24:  :81:27:
    db:23:  :64:  :67:  :ba:  :  :bc:  :  :  :  :
      :ae:88:  :  :  :  :  :91:  :  :14:  :ba:ef:
      :89:  :  :  :  :  :  :  :  :05:  :75:52:  :
      :  :  :be:ad:df:  :02:88:00:  :  :15:45:  :
    cf:32:  :ca:  :93:  :32:  :40:  :27:dd:  :19:
    73:dc:  :  :  :  :  :cf:  :  :dd:  :  :ca:  :
    ee:  :ca:  :  :  :49:  :27:  :58:53:  :64:25:
      :22:06:16:ff:62:bc:  :  :  :  :24:fc:  :  :
    df""")
# exponent2
_, dqmask_msk, dqmask_val = msk("""02:  :bd:  :19:25:98:75:  :65:  :55:28:33:bc:
    34:84:91:01:96:  :  :08:  :32:45:  :27:  :  :
      :fe:  :bb:63:32:68:  :51:bd:75:40:  :52:52:
      :  :  :78:85:fc:94:  :07:  :14:  :  :  :  :
    15:dd:  :  :93:  :01:  :  :77:ca:  :40:  :da:
      :89:bc:87:62:dc:ac:61:88:  :  :70:  :69:  :
      :36:  :  :21:08:  :dc:73:  :ad:da:ee:fe:  :
    96:  :58:  :  :46:  :29:ff:97:ce:  :  :  :cb:
    51:  :  :81:  :22:  :  :19:  :10:69:41:36:ca:
      :22:49:  :cc:cf:06:  :  :08:  :76:  :  :45:
    98:  :  :45:  :  :  :69:13:65:  :  :da:54:  :
    19:  :ee:24:  :73:  :  :  :  :  :  :18:53:40:
    21:25:  :  :84:52:cd:  :49:33:78:  :  :ed:  :
    25:27:  :  :  :ca:  :  :  :ca:  :  :bc:  :02:
    31:70:  :10:ca:84:59:  :  :  :52:  :27:76:  :
    47:  :66:bf:ff:  :03:  :99:ff:  :df:  :  :  :
      :46:27:45:  :65:07:  :48:da:dc:  :80:  :  :
    f9""")
```

```
def search(K, Kp, Kq, check_level, break_step):
    max_step = 0
    cands = [0]
    for step in range(1, break_step + 1):
        #print " ", step, "( max =", max_step, ")"
        max_step = max(step, max_step)

        mod = 1 << (4 * step)
        mask = mod - 1

        cands_next = []
        for p, new_digit in product(cands, p_ranges[-step]):
            pval = (new_digit << ((step - 1) * 4)) | p

            if check_level >= 1:
                qval = solve_linear(pval, N & mask, mod)
                if qval is None or not check_val(qval, mask, qmask_msk, qmask_val):
                    continue

            if check_level >= 2:
                val = solve_linear(E, 1 + K * (N - pval - qval + 1), mod)
                if val is None or not check_val(val, mask, dmask_msk, dmask_val):
                    continue

            if check_level >= 3:
                val = solve_linear(E, 1 + Kp * (pval - 1), mod)
                if val is None or not check_val(val, mask, dpmask_msk, dpmask_val):
                    continue

            if check_level >= 4:
                val = solve_linear(E, 1 + Kq * (qval - 1), mod)
                if val is None or not check_val(val, mask, dqmask_msk, dqmask_val):
                    continue

                if pval * qval == N:
                    print "Kq =", Kq
                    print "pwned"
                    print "p =", pval
                    print "q =", qval
                    p = pval
                    q = qval
                    d = invmod(E, (p - 1) * (q - 1))
                    coef = invmod(p, q)

                    from Crypto.PublicKey import RSA
                    print RSA.construct(map(long, (N, E, d, p, q, coef))).exportKey()
                    quit()

            cands_next.append(pval)

        if not cands_next:
            return False
        cands = cands_next
    return True



def check_val(val, mask, mask_msk, mask_val):
    test_mask = mask_msk & mask
    test_val = mask_val & mask
```

```
    test_val = mask_val & mask
    return val & test_mask == test_val


# K = 4695
# Kp = 15700
# Kq = 5155

for K in range(1, E):
    if K % 100 == 0:
        print "checking", K
    if search(K, 0, 0, check_level=2, break_step=20):
        print "K =", K
        break

for Kp in range(1, E):
    if Kp % 1000 == 0:
        print "checking", Kp
    if search(K, Kp, 0, check_level=3, break_step=30):
        print "Kp =", Kp
        break

for Kq in range(1, E):
    if Kq % 100 == 0:
        print "checking", Kq
    if search(K, Kp, Kq, check_level=4, break_step=9999):
        print "Kq =", Kq
        break
```

美好的回忆

设c[i]为密文按8位分割的段，m[i]为明文按8位分割的段，根据加密算法，有

c[0] = m[0] ^ iv ^ key

c[1] = m[1] ^ c[0] ^ key

…

c[n] = m[n] ^ c[n-1] ^ key

我们已知c[i]和m[0]、m[1]，根据c[1] = m[1] ^ c[0] ^ key可求出key，再根据c[0] = m[0] ^ iv ^ key求出iv，即可依次求出明文。解题代码如下：

```
#!/usr/bin/python
#encoding=utf-8

def strxor(a, b):
    c = ''
    for i in xrange(len(a)):
        c += chr(ord(a[i])^ord(b[i%len(b)]))
    return c

start = 'have a good time.'
ciphertext = open('flag.txt.encrypted','rb').read()
c = []
for i in xrange(8):
    c.append(ciphertext[i*8:i*8+8])

m = []
for i in xrange(8):
    m.append('1')
m[0] = start[0:8]
m[1] = start[8:16]

key = strxor(strxor(c[1],m[1]),c[0])
iv = strxor(strxor(c[0],key),m[0])
result = ''
for i in xrange(8):
    m[i] = strxor(strxor(c[i],key),iv)
    iv = c[i]
    result += m[i]
print result
```

悲伤的结局

本题加密算法与上题相同，但是仅知道明文尾部部分信息，因此需要判断填充位。填充位有8种可能，分别对应8个不同的m[n]和m[n-1]，由

```
c[n] = m[n] ^ c[n-1] ^ key
c[n-1] = m[n-1] ^ c[n-2] ^ key
```

两式可以算出两个key。对于8种不同的填充，如果两个key相同，说明此时填充位正确且key正确。
但是本题实际上有个花招，即最后一部分加密结果被篡改，所以如果要求完全相同是求不出解的。这时可以输出每种填充时求得的两个key进行观察，发现当填充位为4时两个key比较相像。将此时的key当作真key，即可解出除第一段外的所有明文。
解题代码如下：

```
#!/usr/bin/python
#encoding=utf-8

def strxor(a, b):
    c = ''
    for i in xrange(len(a)):
        c += chr(ord(a[i])^ord(b[i%len(b)]))
    return c

last = 'keep away from xiaocui!'
ciphertext = open('flag.txt.encrypted','rb').read()
pad = -1
key = ''
for i in xrange(8):
    tmpm = last[i-23:]+chr(i+1)*(i+1)
    k1 = strxor(strxor(ciphertext[-16:-8],tmpm[8:16]),ciphertext[-24:-16])
    k2 = strxor(strxor(ciphertext[-8:],tmpm[16:24]),ciphertext[-16:-8])
    print i
    print k1.encode('hex')
    print k2.encode('hex')
    if(k1[:3]==k2[:3]):
        pad = i+1
        key = k1
        break
plaintext = ''
count = len(ciphertext)/8
for i in xrange(count-1):
    plaintext += strxor(strxor(ciphertext[i*8+8:i*8+16],ciphertext[i*8:i*8+8]),key)
print plaintext
```