ByteCTF2021 Crypto - easyxor writeup

原创

风好衣轻 于 2021-10-21 12:00:27 发布 146 收藏

分类专栏： CTF 文章标签： python 安全

easyxor

`shift` 函数是个常见的移位异或操作， `convert` 是对一个数字使用不同的key和mask进行4次移位异或，这个函数在已知key的情况下是可逆的。

`encrypt` 函数是对明文块进行两种模式（CBC和OFB）的块加密，块长度为8，对于每一块的加密使用的就是上面的 `convert` 函数。

首先通过密文的长度可以得知一共被分成了6块；前3块明文使用OFB模式，后三块明文使用CBC模式；keys是一个长度为4的列表，列表中每个值的范围是(-32, 32)，64 爆破也是可以接受的。

读完题目代码之后可以想到其实我们已经知道第一块明文了，就是flag的格式 `ByteCTF{` ，而OFB模式实际上是加密的key，最终结果和明文块异或，所以第一个明文块异或第一个密文块就可以知道第一个key加密的结果，也就是 `cur_c = convert(last, k)` 的 `cur_c` ，这样就可以得到第二块的last。

现在对于第二块，已知IV（last），未知keys，已知明文是可显示字符，所以可以爆破keys了，把能解出可显示字符明文的keys都保留出来，发现有4836个keys是满足的，那么我们还要借助第三块再筛一次，最终只得到一组keys。

```
from itertools import product
from tqdm import tqdm
from Crypto.Util.number import bytes_to_long, long_to_bytes
def check(s):
    return min([((i<129) and (i>31)) for i in s])

c = "89b8aca257ee2748f030e7f6599cbe0cbb5db25db6d3990d3b752eda9689e30fa2b03ee748e0da3c989da2bba657b912"
c_list = [int(c[i*16:i*16+16], 16) for i in range(len(c)//16)]
known_m = bytes_to_long(b'ByteCTF{')
range64 = list(range(-32, 33))
cur_c = known_m^c_list[0]
print(cur_c)
k_cnt = 0
for a,b,c,d in tqdm(product(range64, range64, range64, range64)):
    last = cur_c
    k = [a, b, c, d]
    try_cur_c = convert(last, k)
    m1 = long_to_bytes(try_cur_c ^ c_list[1])
    if check(m1): # 只筛选这第一轮的话，4836个k是满足条件的，所以得筛第二轮
        last = try_cur_c
        try_cur_c = convert(last, k)
        m2 = long_to_bytes(try_cur_c ^ c_list[2])
        if check(m2):
            k_cnt += 1
            try:
                print(m1.decode() + m2.decode(), k)
            except:
                print("error")
print(k_cnt)
# keys = [-12, 26, -3, -31]
# ByteCTF{5831a241s-f30980
```

现在已经得到了keys和前三块的明文，可以接着解后三块明文了。

```
from Crypto.Util.number import bytes_to_long, long_to_bytes

k = [-12, 26, -3, -31]
c = "89b8aca257ee2748f030e7f6599cbe0cbb5db25db6d3990d3b752eda9689e30fa2b03ee748e0da3c989da2bba657b912"
cl = [int(c[i*16:i*16+16], 16) for i in range(len(c)//16)]
cur_c = bytes_to_long(b'ByteCTF{') ^ cl[0]

def shift(m, k, c):
    if k < 0:
        return m ^ m >> (-k) & c
    return m ^ m << k & c

def convert(m, key):
    c_list = [0x37386180af9ae39e, 0xaf754e29895ee11a, 0x85e1a429a2b7030c, 0x964c5a89f6d3ae8c]
    for t in range(4):
        m = shift(m, key[t], c_list[t])
    return m

def unshift_right(value, key, mask=None, nbits=32):
    if not mask: mask = (1 << (nbits + 1)) - 1
    i = 0
    while i * key < nbits:
        part_mask = ((((1 << nbits)-1) << (nbits - key)) & ((1 << nbits)-1)) >> (i * key)
        part = value & part_mask
        value ^= (part >> key) & mask
        i += 1
```

```
        i += 1
    return value

def unshift_left(value, key, mask=None, nbits=32):
    if not mask: mask = (1 << (nbits + 1)) - 1
    i = 0
    while i * key < nbits:
        part_mask = ((((1 << nbits)-1) >> (nbits - key)) & ((1 << nbits)-1)) << (i * key)
        part = value & part_mask
        value ^= (part << key) & mask
        i += 1
    return value

def my_unshift(m, k, c):
    if k < 0:
        tmp = unshift_right(m, -k, c, 64)
        return tmp
    tmp = unshift_left(m, k, c, 64)
    return tmp

def re_convert(m, key):
    c_list = [0x37386180af9ae39e, 0xaf754e29895ee11a, 0x85e1a429a2b7030c, 0x964c5a89f6d3ae8c]
    for t in range(3, -1, -1):
        m = my_unshift(m, key[t], c_list[t])
    return m

IV = re_convert(cur_c, k)
assert IV.bit_length() == 64

last = IV
cur = re_convert(cl[3], k)
m3 = long_to_bytes(cur ^ last)
print(m3)

last = cl[3]
cur = re_convert(cl[4], k)
m4 = long_to_bytes(cur ^ last)
print(m4)

last = cl[4]
cur = re_convert(cl[5], k)
m5 = long_to_bytes(cur ^ last)
print(m5)
print(m3 + m4 + m5)
# q535af-2156547475u2t}$$$
```

拼接起来得到完整flag: ByteCTF{5831a241s-f30980q535af-2156547475u2t}