Crypyto HW1 Write UP

COR

- 一層LFSR不夠,那就兩層吧XD
  - 再不夠就三層(X
- Src code分析

1. LFSR(第一層) ```python= class LFSR: def **init**(self, init, feedback): self.state = init self.feedback = feedback def getbit(self): nextbit = reduce(lambda x, y: x ^ y, [i & j for i, j in zip(self.state, self.feedback)]) self.state = self.state[1:] + [nextbit] return nextbit # nextbit = ((s0&p0 ^ s1&p1) ^ ...) ^ ... # state每次拿掉LSB,shift right 1 bit並在MSB補上nextbit

```
2. MYLFSR(第二層)
```python= 13
class MYLFSR:
# input一個list,每個list都有2bytes(16bit),將其每個element都轉成bit array
# 初始化l1, l2, l3的init跟feedback值
def __init__(self, inits):
    inits = [[int(i) for i in f"{int.from_bytes(init, 'big'):016b}"] for init in inits]
    self.l1 = LFSR(inits[0], [int(i) for i in f'{39989:016b}'])
    self.l2 = LFSR(inits[1], [int(i) for i in f'{40111:016b}'])
    self.l3 = LFSR(inits[2], [int(i) for i in f'{52453:016b}'])

# 取得nextbit x1,x2,x3, 再進行第二層LFSR
def getbit(self):
    x1 = self.l1.getbit()
    x2 = self.l2.getbit()
    x3 = self.l3.getbit()
    return (x1 & x2) ^ ((not x1) & x3)
# 取得output
def getbyte(self):
    b = 0
    for i in range(8):
        b = (b << 1) + self.getbit()
    return bytes([b])
```

```python=33 FLAG = open('./flag', 'rb').read() assert len(FLAG) == 12 # FLAG lenth = 12 assert FLAG.startswith(b'FLAG{') assert FLAG.endswith(b'}') # FLAG = 'FLAG{xxxxxx}' FLAG = FLAG[5:-1] # FLAG = 'xxxxxx'

lfsr = MYLFSR([FLAG[0:2], FLAG[2:4], FLAG[4:6]]) print([lfsr.getbit() for _ in range(100)]) ```

3. Summery

```
把input拆成三份R1, R2, R3,分別做lfsr得到x1, x2, x3
for i in range(100):
    output[i] = (x1 & x2) ^ ((~x1) & x3)
```

4. 想法: 雖然多了一層,但還是可以用correlation attack處理
   - Correlation Attack
     - 建立一個x1.x2.x3.output的table `output = (x1&x2) ^ ((~x1)&x3)`

x1	x2	x3	output
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0

x1	x2	x3	output
1	1	0	1
1	1	1	1

- 可以寫個script來計算xi跟output的相關度 ```python=

cal similarity between xi & output

cnt1 = 0 cnt2 = 0 cnt3 = 0 for x1, x2, x3 in list(product([0,1], repeat=3)): out = (x1 & x2) ^ ((not x1) & x3) if (x1 == out): cnt1 += 1 if (x2 == out): cnt2 += 1 if (x3 == out): cnt3 += 1 print(f'x1:{cnt1/8}') # x1 = 0.5 print(f'x2:{cnt2/8}') # x2 = 0.75
print(f'x3:{cnt3/8}') # x3 = 0.75

- 因此可以先嘗試用暴力枚舉R3,如果R3產生的x3跟output的相似度接近75,就有可能是正確的R3
```python=53
# cal similarity between 2 lists
def correlation(a,b):
    cnt = 0
    for i, j in zip(a,b):
        if (i == j):
            cnt += 1
    return cnt

# brute force R3 (by cmp x3 with output)
for R3 in list(product([0,1], repeat=16)): #產生所有可能性
    R3 = [R3[i] for i in range(16)]
    lfsr = MYLFSR([[0]*16, [0]*16, R3])
    # I use bit array as input instead
    x3 = []
    for i in range(100):
        x3.append(lfsr.l3.getbit())
    if (correlation(x3, output) >= 70): #概率訂為70%
        print(f'R3 : {R3}')
        R3_ = chr(int('0b' + ''.join([str(i) for i in R3[:8]]), 2)) + chr(int('0b' + ''.join([str(i) for i in R3[8
        print(f'R3_ : {R3_}')
```

```
R3只有一個解
    R3 : [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0]
    R3_ : hj
同理爆破R2, R2有兩個解
    R2 : [0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1]
    R2_ : Vi
    R2 : [0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1]
    R2_ : ui
```

- 最後再爆破R1,利用R2跟R3算出out跟題目給的output做比較 `python=92 for i in printable: for j in printable: FLAG = bytes(i.encode()) + bytes(j.encode()) lfsr = MYLFSR([FLAG, b'ui', b'hj']) x1 = [lfsr.l1.getbit() for _ in range(100)] x2 = [lfsr.l2.getbit() for _ in range(100)] x3 = [lfsr.l3.getbit() for _ in range(100)] out = [((x1[i] & x2[i]) ^ ((not x1[i]) & x3[i])) for i in range(100)] if correlation(out, output) == 100: print('FLAG{' + FLAG.decode() + 'uihj' + '}')` > FLAG = FLAG{'dfuihj'}

完整code

```python=

# !/usr/bin/env python3

from functools import reduce from itertools import product from string import printable

class LFSR: def **init**(self, init, feedback): self.state = init self.feedback = feedback def getbit(self): nextbit = reduce(lambda x, y: x ^ y, [i & j for i, j in zip(self.state, self.feedback)]) self.state = self.state[1:] + [nextbit] return nextbit

class MYLFSR: def **init**(self, inits): inits = [[int(i) for i in f"{int.from_bytes(init, 'big'):016b}"] for init in inits] self.l1 = LFSR(inits[0], [int(i) for i in f'{39989:016b}']) self.l2 = LFSR(inits[1], [int(i) for i in f'{40111:016b}']) self.l3 = LFSR(inits[2], [int(i) for i in f'{52453:016b}']) def getbit(self): x1 = self.l2.getbit() x2 = self.l2.getbit() x3 = self.l3.getbit() return (x1 & x2) ^ ((not x1) & x3)
def getbyte(self): b = 0 for i in range(8): b = (b << 1) + self.getbit() return bytes([b])
```

def xor(a, b): return bytes([i ^ j for i, j in zip(a, b)])

output = [1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1]

cal similarity between xi & output

cnt1 = 0 cnt2 = 0 cnt3 = 0 for x1, x2, x3 in list(product([0,1], repeat=3)): out = (x1 & x2) ^ ((not x1) & x3) if (x1 == out): cnt1 += 1 if (x2 == out): cnt2 += 1 if (x3 == out): cnt3 += 1 print(f'x1:{cnt1/8}') print(f'x2:{cnt2/8}')
print(f'x3:{cnt3/8}')

cal similarity between 2 lists

def correlation(a,b): cnt = 0 for i, j in zip(a,b): if (i == j): cnt += 1 return cnt

brute force R3 (by cmp x3 with output)

for R3 in list(product([0,1], repeat=16)): R3 = [R3[i] for i in range(16)] lfsr = MYLFSR([[0]*16, [0]*16, R3]) x3 = [] for i in range(100): x3.append(lfsr.l3.getbit()) if (correlation(x3, output) >= 70): print(f'R3 : {R3}') R3_ = chr(int('0b' + ''.join([str(i) for i in R3[:8]]), 2)) + chr(int('0b' + ''.join([str(i) for i in R3[8:]]), 2)) print(f'R3_ : {R3_}')

brute force R2

for R2 in list(product([0,1], repeat=16)): R2 = [R2[i] for i in range(16)] if (R2[0] or R2[8]): continue lfsr = MYLFSR([[0]*16, R2 , [0]*16]) x2 = [] for i in range(100): x2.append(lfsr.l2.getbit()) if (correlation(x2, output) >= 70): #print(f'x2 : {x2}') print(f'R2 : {R2}') R2_ = chr(int('0b' + ''.join([str(i) for i in R2[:8]]), 2)) + chr(int('0b' + ''.join([str(i) for i in R2[8:]]),2)) print(f'R2_ : {R2_}')

R2 = [0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1] # Vi

R2 = [0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1] # ui

R3 = [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0] # hj

for i in printable: for j in printable: FLAG = bytes(i.encode()) + bytes(j.encode()) lfsr = MYLFSR([FLAG, b'ui', b'hj']) x1 = [lfsr.l1.getbit() for _ in range(100)] x2 = [lfsr.l2.getbit() for _ in range(100)] x3 = [lfsr.l3.getbit() for _ in range(100)] out = [((x1[i] & x2[i]) ^ ((not x1[i]) & x3[i])) for i in range(100)] if correlation(out, output) == 100: print(FLAG + b'ui' + b'hj')

POA
- Source Code分析
    - 有點不一樣的padding oracle 攻擊
    其實就是padding方式不一樣,跟MD5類似的padding方式(不夠的byte先補一個0x80剩下補0x00)
    ```python=
    def pad(data):
    padlen = 16 - len(data) % 16
    return data + int('1' + '0' * (padlen * 8 - 1), 2).to_bytes(padlen, 'big')
    # data + 0x80 0x00 ... 0x00
    ```

    - padding error的判斷方式也很簡單,就是從還原的data最後面往回找

        > = 0x00就繼續往前找
        > = 0x80就代表找到data尾,回傳0x80之前的byte
        > = else: padding error
    ```python=
    def unpad(data):
    for i in range(len(data) - 1, len(data) - 1 - 16, -1): # 由最後一byte往前找
        if data[i] == 0x80:  # [raw_data] + '10000000(bit)'
            return data[:i]  # ret [raw_data]
        elif data[i] != 0x00: # if no 00 exist >> 代表padding錯誤
            raise PaddingError
    raise PaddingError
    ```

    - 加密方式
        - 採用CBC mode,iv亂數產生,回傳值為iv + cipher,代表題目印出的東西前16byte就是iv
        ```python=
        def encrypt(plain):
        # random iv
        iv = os.urandom(16)
        # encrypt
        aes = AES.new(key, AES.MODE_CBC, iv)
        cipher = aes.encrypt(pad(plain))
        return iv + cipher #cipher前16byte為iv
        ```
    - main(): 會嘗試decrypt收到的新cipher,decrypt之後會拿去unpad,此時如果有padding error就會print NOOOOO
    ```python=
    while True:
        try:
            decrypt(bytes.fromhex(input('cipher = ')))
            print('YESSSSSSSS')
        except PaddingError:
            print('NOOOOOOOOO')
        except:
            return
    ```
- 已知:
    :::info
    1. 題目給的密文
    2. 傳入的cipher是否造成padding error
    3. padding = b'\x80' + b'\x00' * n (n>=0) --> 長度未知
    :::

    - 想法:
        1. 先找出padding有幾個,因為padding只會在最後一個block,因此先用一個迴圈跑,要注意的是
        ```python=
        ans = b''
        iv = cipher[-32:-16]
        block = cipher[-16:]
        for b in range(16):
            for c in range(256):
                if b == 0 and c == iv[-1]:
                    continue
                my_c = iv[:15-b] + bytes([c]) \ # 偽造的Ci-1 byte
                     + xor(bytes([0] * b), xor(iv[-b:], ans)) \
                       # Pi ^ (Ci-1 ^ Ci -1')
                     + block # Ci

                if (oracle(my_c)): # YESSSSSS
                    if c==0: #開始重複惹
                        break
                    else:
                        ans = bytes([0]) + ans
                        break

        padding = b'\x80' + ans[:-1] #換掉第一個byte
        ```

        2. 接下來就可以用正常的方式去跑
來不及寫完的codeQQ
```python=
from pwn import *

r = process('./server.py')

r.recvuntil('cipher = ')
cipher = r.recvline().decode() # ...\n -> ...
cipher = bytes.fromhex(cipher)  # hex str to byte
print(cipher)
```

```
def oracle(cipher):
    r.sendlineafter('cipher = ', cipher.hex())
    if b'YESSSSSSSS' in r.recvline():
        return True
    else:
        return False


b_num = len(cipher)//16 - 1
print(f'block數 : {b_num}')

# padding可能為
# 0x80
# 0x80 0x00
# 0x80 0x00 ....

# find padding
ans = b''
iv = cipher[-32:-16]
block = cipher[-16:]
for b in range(16):
    for c in range(256):
        if b == 0 and c == iv[-1]:
            continue
        my_c = iv[:15-b] + bytes([c]) + xor(bytes([0] * b), xor(iv[-b:], ans)) + block
        print(my_c.hex())
        if (oracle(my_c)):
            if c==0:
                break
            else:
                ans = bytes([0]) + ans
                break

padding = b'\x80' + ans[:-1]


#find the other byte in the last block
for b in range(len(padding),16):
    for c in range(256):
        if b == 0 and c == iv[-1]:
            continue
        my_c = iv[:15-b] + bytes([c]) + xor(bytes([0] * b), xor(iv[-b:], padding)) + block
        if (oracle(my_c)):
            padding = bytes([iv[15 - b] ^ c ^ (b + 1)]) + padding
            break

print(padding.hex())

# find the other bytes in other blocks
flag = b''
for i in range(16, len(cipher)-16, 16):
    ans = b''
    iv, block = cipher[i-16:i], cipher[i:i+16]
    for b in range(16):
        for c in range(256):
            if b == 0 and c == iv[-1]:
                continue
            my_c = iv[:15-b] + bytes([c]) + xor(bytes([0] * b), xor(iv[-b:], padding)) + block
            if (oracle(my_c)):
                padding = bytes([iv[16 - 1 - j] ^ k ^ (j + 1)]) + padding
                break
    flag += padding
print(flag)

r.interactive()
```

```
.interactive()
```