Crypto Lab 2

1. CBC Byte Flip

1.1. 题目特点

题目的载体是AES加密中的CBC模式，题目会给出一个50字节长度的字符串和对应的密文（前十六字节是IV），要求通过CBC字节翻转修改密文，使其解密出的明文变成另一个47字节的字符串。

在不触发程序错误的情况下，服务器可以不断地返回做题者输入的密文所对应的明文（有两个，分别是未取消padding和取消padding的明文），我们可以根据给出的明文进一步修正，直至得到与题目中要求一致的明文。

1.2. 思路

解决这道题的方法就在标题中，即CBC字节反转，其原理是通过修改前一个密块中的值，从而改变下一个密块解出的明文的值，其中IV字段可以看成是第0个密块。例如，修改IV的值，就能使紧邻的第一个密块解出的明文的值发生变化，同时后面的密块解出的明文也会不同。

```
original_iv = ciphertext[:16]
change_for_first_block = bytes([original_plaintext[i] ^ target_plaintext[i] for i
in range(16)])
modified_iv = bytes([original_iv[i] ^ change_for_first_block[i] for i in
range(16)])
original_first_ciphertext_block = ciphertext[16:32]
```

如上面代码展示的，取出IV块，用异或运算来计算第一个原始明文块与预期明文块的差距，然后将这个差距和IV再做一次异或运算，就能修改IV块使得第一个明文块修改成我们想要的明文。以此类推，修改第一个密文块来修改第二个明文块，直至所有明文块都修改过一次。但此时会发现，除了最后一个明文块，前面修改过的明文块都变成了乱码，这是因为我们修改了这些明文块对应的密文块。此时只需要从后往前再计算一次就行，即从倒数第三个密文块开始，通过修改倒数第三个密文块来修改倒数第二个明文块，通过修改倒数第四个密文块来修改倒数第三个明文块，最后修改IV来修改第一个明文块。

在代码实现过程中还有一些需要注意的：

- 文本末尾的emoji表情每一个都对应着三个字节，可以通过在线的十六进制转换得到
- 目标文本是47字节，补位的字节值和距离下一个块所缺少的值一致，比如这里47字节，距离下一个16字节的整数倍差1，因此补1个\x01，同理差两个的话就补2个\x02
- 为了避免不必要的麻烦，可以把目标文本改成50字节，即补3个\x01，最后再删掉一个块即可

1.3. 实现

声明：为了调试方便，没有做全自动化（没有连接靶机和自动读取的操作），最后拿到flag了也没有改动，代码中保留的ciphertext和下面的图片一致。代码输出的前四个字符串分别从后往前地完成了明文块的修改，最后一个输出删掉了多余的块，如下图所示：

```
Welcome to SECURE Crypto System
No way to hack😜
Your Crypto System is HACKED BY AAA🤣🤣🤣
ciphertext:  504a17a9a1135ee44382748a211a388744ce2fbbaed197df312777c88980d5212570ad9a66
fe373b8de7fb0fe0225835524a4758bcace1a2521f153c29caada6ae999061ecf74e587a97073fd5c9e9b0
5e400eb8ee3d49bd47993bf937201ea164834ca0a481abf1523f4bffdda7e10b2a5ecc1d9823b4bf7d63309
e1ced0babcbd74758bcace1a2521f153c29caada6ae999061ecf74e587a97073fd5c9e9b0
b'u\x05\xa4\xbf\x19@\x9c\xe7[\xa7\x8b\xf8\xe1W\x1f\xb4\xe6@t\xf8\x8b\xcd`\xe3\x86\xc5\x
d5j\x82F\xd57\x032\x1c)\xe1\xcf\x01\x18\x99\xecKvp\x92\xd1\xc2\x01\x01\x0e\x0e\x0e\x0e\
x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'
plaintext:  7505a4bf19409ce75ba78bf8e1571fb4e64074f88bcd60e386c5d56a8246d53703321c29e1c
f011899ec4b767092d1c20101
5e400eb8ee3d49bd47993bf937201ea164834ca0a481abf1523f4bffdda7e10b682d91c4e64816577b2bd81
8f3db7968cbd74758bcace1a2521f153c29caada6ae999061ecf74e587a97073fd5c9e9b0
b'u\x05\xa4\xbf\x19@\x9c\xe7[\xa7\x8b\xf8\xe1W\x1f\xb4\tN\xb3\xf7/&\xab\xd8\xde\x88/a\x
99\xdf\x8eMAAA\xf0\x9f\xa4\xa3\xf0\x9f\xa4\xa3\xf0\x9f\xa4\xa3\x01\x01\x01\x0e\x0e\x0e\
x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'
plaintext:  7505a4bf19409ce75ba78bf8e1571fb4094eb3f72f26abd8de882f6199df8e4d414141f09fa
4a3f09fa4a3f09fa4a3010101
5e400eb8ee3d49bd47993bf937201ea108a0df3ef8874868cffc21da643a3666682d91c4e64816577b2bd81
8f3db7968cbd74758bcace1a2521f153c29caada6ae999061ecf74e587a97073fd5c9e9b0
b'\xb7yV\xc9M\xc5\xb0_a\x8b\x91\x18jlt\x89em is HACKED BY AAA\xf0\x9f\xa4\xa3\xf0\x9f\x
a4\xa3\xf0\x9f\xa4\xa3\x01\x01\x01\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\
x0e'
plaintext:  b77956c94dc5b05f618b91186a6c7489656d20697320484143 4b4544204259204141 41f09fa
4a3f09fa4a3f09fa4a3010101
b0562d0383bb8b9b5666c5c10e35195c08a0df3ef8874868cffc21da643a3666682d91c4e64816577b2bd81
8f3db7968cbd74758bcace1a2521f153c29caada6ae999061ecf74e587a97073fd5c9e9b0
b'Your Crypto System is HACKED BY AAA\xf0\x9f\xa4\xa3\xf0\x9f\xa4\xa3\xf0\x9f\xa4\xa3\x
01\x01\x01\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'
plaintext:  596f75722043727970746f2053797374656d20697320484143 4b4544204259204141 41f09fa
4a3f09fa4a3f09fa4a3010101
b0562d0383bb8b9b5666c5c10e35195c08a0df3ef8874868cffc21da643a3666682d91c4e64816577b2bd81
8f3db7968cbd74758bcace1a2521f153c29caada6
b'Your Crypto System is HACKED BY AAA\xf0\x9f\xa4\xa3\xf0\x9f\xa4\xa3\xf0\x9f\xa4\xa3\x
01'
plaintext:  596f75722043727970746f2053797374656d20697320484143 4b4544204259204141 41f09fa
4a3f09fa4a3f09fa4a3
AAA{Wow_7h1siS_CbcFLIP@ttacK^uL0ps8|5fMj5oAd2}
```

2. Padding Oracle

2.1. 题目特点

同样是CBC，同样是发送密文，但这次不返回明文，只返回三个状态：200表示密文对应的与明文一致，500表示padding错误，403表示padding正确但不是明文。这么看来返回的信息量比之前的题目少了很多，需要的爆破次数大幅增加，必须要用全自动交互了。

服务器只在最开始给我们密文，密文所对应的明文就是flag，找到明文就解出了flag。

2.2. 思路

根据我在第一题里提到的padding规则，离完整的密文块缺n个，就补n个十六进制的n。以IV为例，修改IV的最后一个字节，从0x00遍历到0xff，其中会有一个在解密后使得第一个明文块的最后一个字节为0x01，也就是正确的padding，此时返回403（padding正确但不是明文），这个时候我们就可以通过此时IV的最后一个字节与0x01进行异或得到这个位置对应的中间值（即代码中的intermediate_bytes）。下一个操作就是想办法让明文块的最后两个字节都是0x02，即下一个padding，此时IV的最后一个字节是中间值的最后一个字节和0x02异或后的结果，这样能保证最后一个字节解出来的明文一定是0x02，只需要遍历IV块的倒数第二个字节来使明文块的倒数第二个字节也是0x02即可。将以上规则遍历整个IV块就能得到第一个明文块的全部内容了。部分代码如下：

```
# 逐字节破解,初始化iv=[0]*16，从最后一个字节开始，逐个字节破解，直到第一个字节，得到中间值
#当返回403时，说明有一位破解成功，记录中间值，继续破解下一位
#当返回200时，说明iv破解完成，记录中间值
iv = [0] * 16
for i in range(15, -1, -1):
    #将已确定的iv部分与填充字符个数进行异或，得到新的iv
    for k in range(15, i, -1):
        iv[k]=intermediate_bytes[k]^(16-i)
    for j in range(256):
        iv[i] = j
        conn.sendline((bytes(iv) + enc_msg[:16]).hex())
        result = conn.recvline().decode()
        if result == "403\n" or result == "200\n":
            intermediate_bytes[i] = j ^ (16-i)
            break
        if j == 255:
            print("error")
            exit()
```

这里为了测试代码逻辑是否正确，如果遍历完一个字节还没有答案就说明写错了，实际使用可以删掉。enc_msg已经把IV剔除了，这里是纯密文块。intermediate_bytes并不是明文，想要得到明文，还需要将中间值和真正的IV做异或才能得到，即：

```
#中间值和已知的IV做异或，则得到第一组密文的明文
plaintext1 = [0] * 16
ciphertext1 = enc_msg[:16]
print(ciphertext1)
for i in range(16):
    plaintext1[i] = intermediate_bytes[i] ^ IV[i]
#转成字符串输出
print("明文第一部分:", ''.join([chr(c) for c in plaintext1]))
```

```
第 6 位中间值: 74
第 5 位中间值: 240
第 4 位中间值: 234
第 3 位中间值: 96
第 2 位中间值: 63
第 1 位中间值: 120
第 0 位中间值: 0
[*] Closed connection to 127.0.0.1 port 60208
wrong
中间值: [0, 120, 63, 96, 234, 240, 74, 69, 92, 246,
 58, 112, 161, 190, 118, 61, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0]
明文: AAA{P@dD1Ng0racL
PS D:\CS\MD\Doc1\site1>
```

将以上的部分复制粘贴略作修改，就能得到完整的明文了。

2.3. 实现

为方便调试，输出了每一个阶段的明文和对应的中间值，最后再写一句代码把三部分都合起来就得到了flag。

```
加密消息: b'920d672bf6a2dc6c832c97bae0613349333149dc645aeee3007832be5c453e86f29a916fb458404303accfd170b05688e76c6b7868b104b474ff98a302a62b88\n'
加密消息字节: b'\x92\rg+\xf6\xa2\xdcl\x83,\x97\xba\xe0a3I31I\xdcdZ\xee\xe3\x00x2\xbe\\E>\x86\xf2\x9a\x91o\xb4X@C\x03\xac\xcf\xd1p\xb0V\x88\xe7lkxh\xb1\x04\xb4t\xff\x98
\xa3\x02\xa6+\x88'
d:\CS\MD\Doc1\site1\docs\CTF\Crypto\CBC1.py:38: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  conn.sendline((bytes(iv) + enc_msg[:16]).hex())
b'31I\xdcdZ\xee\xe3\x00x2\xbe\\E>\x86'
明文第一部分: AAA{P@dD1Ng0racL
d:\CS\MD\Doc1\site1\docs\CTF\Crypto\CBC1.py:65: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  conn.sendline((bytes(iv) + enc_msg[:16]).hex())
[86, 21, 109, 248, 49, 55, 143, 144, 84, 75, 64, 252, 48, 42, 93, 237]
明文第二部分: e$$$UmasT3rBlock
d:\CS\MD\Doc1\site1\docs\CTF\Crypto\CBC1.py:91: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  conn.sendline((bytes(iv) + enc_msg[:16]).hex())
[191, 170, 245, 42, 201, 83, 75, 72, 8, 167, 196, 218, 123, 187, 93, 131]
明文第三部分: M0dE}

全部明文:  AAA{P@dD1Ng0racLe$$$UmasT3rBlockM0dE}
```

前两题的通过截图：

3. Republican Signature Agency

3.1. 题目特点

连接服务器端口，先要做一个小小的pow，找到哈希值后六位与给定的相同的字符串。听助教说是防止服务器被爆破的手段。

通过pow后，有四个选项，0展示说明，1对输入的明文进行RSA数字签名，2是验证签名解密后是否是给定字符串，3是退出。

本题需要我们伪造数字签名，使其明文和要求的字符串一致。

3.2. 思路

RSA是一个安全性很高的算法，直接爆破不太现实。

但是题目能给任意给定字符串的hex返回签名，大可利用一番。

根据RSA选择明文爆破的思路，输入2，4，3，9分别得到c2，c4，c3，c9，可以根据`n=gcd(c2*c2-c4,c3*c3-c9)`来得到n（原理略）。这里得到的n也有一定的概率会出错，但是我们总能多次爆破，总有一次的n恰好是给定值。注意，程序限定了输入的类型是hex，且不小于两个字节，实际真正输入的是0002，0004，0003，0009。

仅仅知道n并不能得到数字签名，我们要诱导服务器帮我们生成签名的一部分。服务器并不能直接生成我们需要的签名，因为限定输入字符串的hex不能大于15个字节，但是我们可以组合签名。

利用RSA的乘法同态的特性，可以组合出需要的结果。例如，给定的字符串的数值为X，能找到X=X1*X2*，*那么X对应的密文C=(C1C2)%n*（原理略）。据此我们只需要找到

若干个子字符串，长度小于等于15（小于2字节补0即可），其乘积为X，就能得到数字签名了。

3.3. 实现

具体实现时遇到了不少阻力：

- 明文转数字并质因数分解后为：117539425503417846151799038508858073088368534710049 (51 digits) = 3 × 127 × 281 × 269323062708893 × 407641812133389400827972056131 3 (31 digits)，这五个质数并不能通过简单组合的形式来生成在ASCII码范围内的字符串，不能通过简单的乘法来解决。打x代表无法表示为字符串，因为ASCII码在0x00到0x7f之间。

```
3
7f
119
x f4f2a97b8a9d
x 33739e09e3796d57c0ebb48aa1
17d
34b
x 2ded7fc729fd7
x 9a5ada1daa6c480742c31d9fe3
x 8b67
x 7984621449c3e3
x 19865b66e7d93d3e88b4ee90c5df
x 10cde5c089b2655
x 3879e878dab0490552c2b92c2ab9
x 313af7fcf9e61c05b60b74d374bfd4c4cecebd
x 1a235
x 16c8d263cdd4ba9
x 4c931234b78bb7bb9a1ecbb2519d
x 3269b1419d172ff
x a96db96a9010db0ff8482b84802b
x 93b0e7f6edb2541122225e7a5e3f7e4e6c6c37
x 85624fa844f8042b
x 1c047a53f47d7439a40e99dce931c7
x 186c41067ff927e6d54faef4e6eb2a8da2988fc3
x 3609ba34ae4d94c244d2933c1b26908c0700ed75
x 19026eef8cee80c81
x 540d6efbdd785cacec2bcd96bb9555
x 4944c3137feb77b47fef0cdeb4c17fa8e7c9af49
x a21d2e9e0ae8be46ce77b9b45173b1a41502c85f
x 1aced36022787ccc6024770ad17821b5777975cd0b
506c7a2067697665206d6520746865206666c616721
```

解决思路：将不能用字符串表示的质数成倍增加，直至能够用字符串表示为止，以下代码能够实现这一功能。

```python
i = 1
factor = 2 #这里写无法用字符串表示的质数
while True:
    try:
```

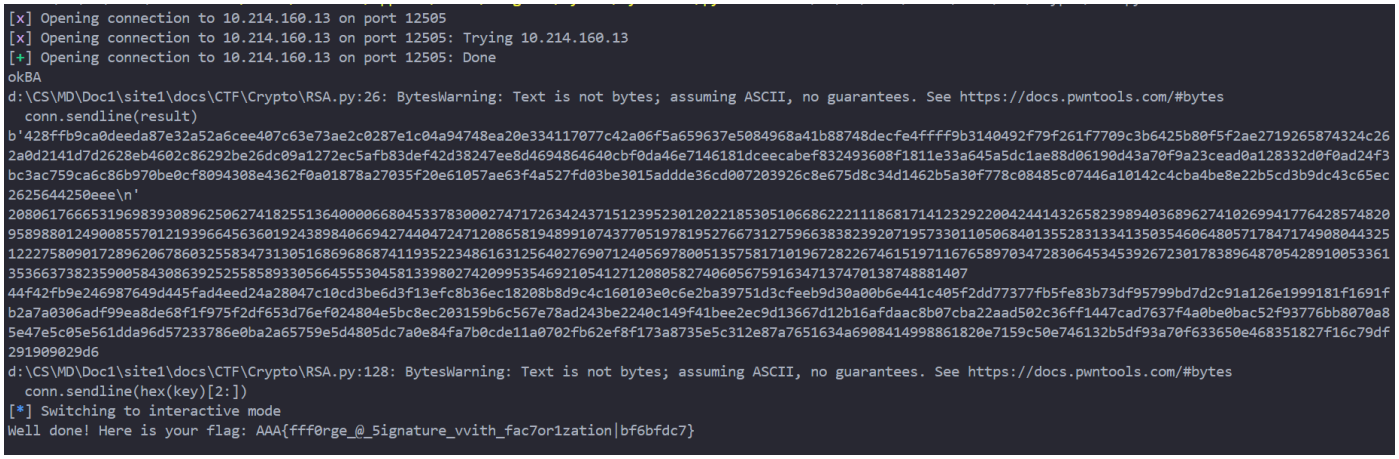
```
            # 计算factor的i倍对应的十六进制字符串，并尝试转换为ASCII码
            hex_str = hex(i * factor)[2:]
            ascii_str = bytes.fromhex(hex_str).decode('ascii')
            print(f"倍数: {i}, 十六进制: {hex_str}, ASCII: {ascii_str}")
            break  # 成功转换后退出循环
        except ValueError:
            # 如果转换失败，增加i的值继续尝试
            i += 1
        except UnicodeDecodeError:
            # 如果转换的字符串不是有效的ASCII字符，增加i的值继续尝试
            i += 1
```

根据以上代码，最终构造了以下组合形式：

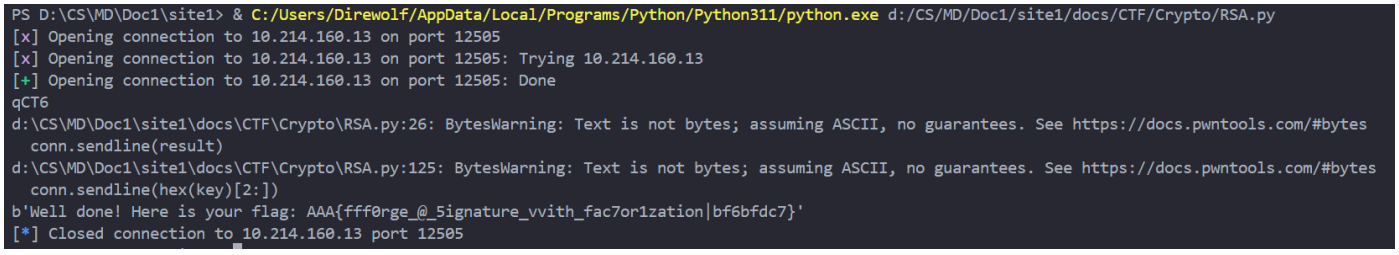
```
factor1 = 3
factor2 = 127
factor3 = 281
factor4 = 269323062708893
factor5 = 407641812133389400827972056131 3
m = factor1*factor2*factor3*((factor4*0x11)//0x11)*
((factor5*25745)//((25745*0x29)//0x29)
```

这么处理需要使用模意义的除法，即求逆，直接调包使用inverse()函数就行。

将以上实现写入代码，最后完成执行得到以下截图：



上面的代码输出了较多的调试用的中间量，以下为修改后的flag纯享版：（忽略一点点warnings和前面的pow）



通过截图：


Republican Signature Agency
Description
Can you get the flag again? nc 10.214.160.13 12505
Hint
Your Answer
AAA{fff0rge_@_5ignature_vvith_fac7or1zation|bf6bfd
Solved
Completed
SuperGay 4qwerty7 Esifiel Das
Schloss 4ncly XnHvHv zjlhhh123 huajuan ddzj 45gfg9