SHANDONG UNIVERSITY

密码学引论实验: Padding Oracle 攻击

网络空间安全学院 (研究院)

2023 年 5 月 26 日

目录

1 实验分工 **2**

2 Padding Oracle 攻击介绍 **2**

2.1 PKCS#5 填充标准 2

2.2 PKCS#7 填充标准 2

2.3 CBC 加密模式 2

2.4 Padding Oracle 攻击原理 3

3 实验过程及实现 **5**

3.1 问题描述 5

3.2 解决步骤 5

3.2.1 找到满足上式的 r 5

3.2.2 判断填充的长度并得到 $a_j \ldots a_b$ 6

3.2.3 进一步得到 $a_1 \ldots a_{j-1}$ 6

3.2.4 恢复出本块的明文 m 及全部明文 7

4 数据与结果 **8**

参考文献 **10**

1 实验分工

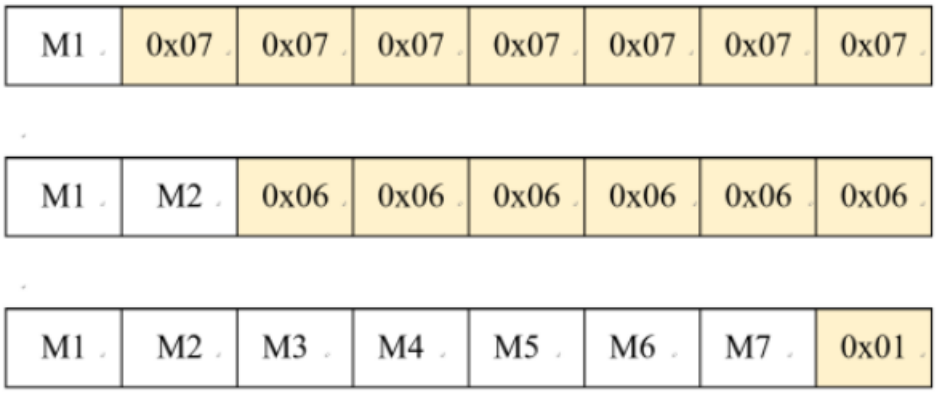
姓名	学号	任务
谢子洋	202100460116	代码实现、论文-实验数据与结果
冯大玮	202100460126	论文
程雨森	202100460090	论文-实验原理
刘志	202122460178	论文-实验过程

2 Padding Oracle 攻击介绍

Padding Oracle 攻击是一种基于 CBC 模式的加密算法的攻击，是一种选择密文攻击，它利用了加密算法中的“填充”机制，即在加密明文时，为了满足加密算法的要求，在明文末尾添加一些特定的填充字符，使得最终加密后的密文长度符合算法的要求。攻击者通过向目标系统发送特定的填充数据，然后观察该系统在解密填充数据时的错误提示信息，从而逐渐推断出加密密钥等关键信息。

2.1 PKCS#5 填充标准

以 8 个字节为一个“分组”，明文长度不足一个分组的，填充足够数量的字节，如填充 7 个字节，每个字节内容为 0x07；明文长度正好一个分组的，也要填充，填充一个分组（8 个字节），每个字节内容为 0x08。即填充的字节内容表示填充的字节数量。



2.2 PKCS#7 填充标准

规则类似于 PKCS#5，但分组长度不再限制为 8，而是为 1-255 字节，例如 AES-128 中 16 个字节一个分组。

2.3 CBC 加密模式

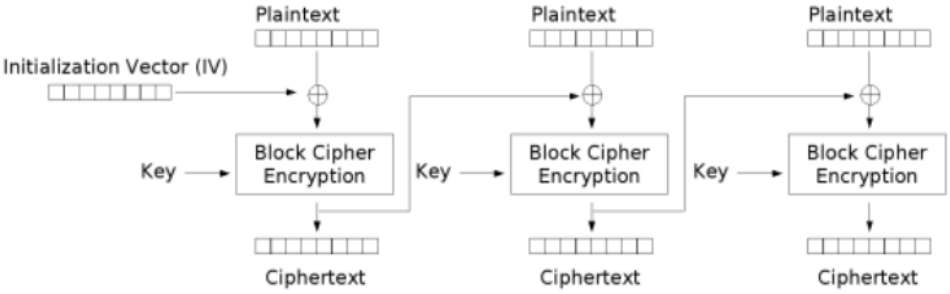
CBC 是一种常见的块加密模式，它可以将一个分组密码算法（如 DES、AES 等）转化为一个可以加密任意长度的消息的密码算法。在 CBC 模式中，每个明文块在加密前会与前

一个密文块进行异或操作，然后再进行加密。

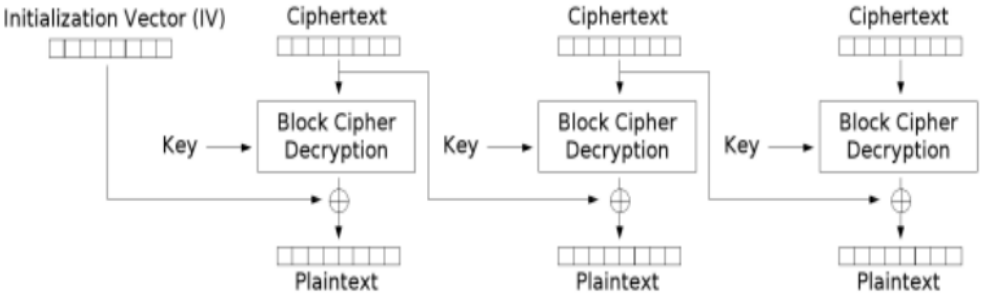
CBC 模式的加密过程：首先将明文分成固定长度的块，对于第一个明文块，将其与一个随机的初始化向量（IV）进行异或操作，对于后续的明文块，将其与上一个密文块进行异或操作，并对每个异或后的块进行加密，产生相应的密文块，再将所有的密文块组合成加密后的消息。

CBC 模式的解密过程与加密过程基本相同，只需将加密操作替换为解密操作即可。具体来说，对于每个密文块，需要进行解密操作，然后与前一个密文块进行异或操作，得到相应的明文块。

■ Cipher Block Chaining 密码分组链接模式



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

2.4 Padding Oracle 攻击原理

Padding oracle 攻击是一种选择密文攻击, 假设所截获密文的最后一个分组为 y，随机选择一个分组 r 作为密文的倒数第二个分组, 将 $r \| y$ 密文发送给解密服务器, 解密服务器运行 CBC 解密程序并判断填充是否合法, 根据解密服务器反馈信息，进一步确定 CBC 解密中间状态值 a.

(1) 找到满足条件的 r:

从最后一个分组入手，穷举猜测 r 使其满足 CBC 最后一个或多个分组的输出填充满足填充规则，穷举搜索 r 的末位，并将对应的 $r \| y$ 发送给解密服务器，直到 CBC 解密反馈填充正确信息。

(2) 判断填充的长度：

从 r 的首位开始修改，并将修改后对应的 $r \| y$ 发送给解密服务器，若服务器返回正确则说明长度不正确，开始修改下一位，重复该操作，直到修改 r_j 时，服务器返回填充错误，则说明填充的字节数量为 $b-j+1$。

(3) 求得 a_{j-1}:

通过调整 r，将 CBC 解密后的填充修改为长度为 $b-j+2$ 的形式，即让 a_{j-1} 对应的字节也成为填充, 将 r 最后 $b-j+1$ 字节重新设置为 $r_k = a_k \oplus (b-j+2)$ for k=j,...,b。穷举搜索 r_{j-1}，将 $r \| y$ 发送给解密服务器，直到服务器返回填充正确，此时输出 $a_{j-1} = r_{j-1} \oplus (b-j+2)$。

(4) 求明文：

利用上述方法可恢复任意一个分组 y 的解密中间状态 Dec(y)=$a_1...a_b$, 此时，再根据截获的上一个分组密文 y', 利用 $y' \oplus a_1...a_b$，可得到真实的明文。

3 实验过程及实现

3.1 问题描述

密文内容：本组组号为 10，即本组的目标密文 IV ‖ c_1 ‖ c_2 ‖ c_3 为：

6b10ced1a3f2134c972f5b84224c369a25c9403e3e0504853c04099b25a5d47ed43

91c0886ad76a90b930369b318addb1b7c352f3b15254a0385c5ac2c39792b

已有条件：敌手已截获 AES128-CBC 目标密文 IV ‖ c_1 ‖ c_2 ‖ c_3（采用 PKCS#7 填充标准），敌手可访问对应解密服务器，该服务器可反馈输入所对应明文的填充状态，即:

(1) 如果解密后明文不满足填充规则，则返回填充错误信息，如 HTTP 500 server error

(2) 如果解密后明文满足则填充规则，则返回正确信息，如 HTTP 200

根据以上条件破解 c_1 ‖ c_2 ‖ c_3 的明文.

3.2 解决步骤

基本思想：Padding oracle 攻击是一种选择密文攻击。假设所截获密文的最后一个分组为 y，随机选择一个分组 $r = r_1 r_2 r_b$ 作为密文的倒数第二个分组，其中 r_i 为一个字 (word)，将 rIly 作为密文发送给解密服务器。解密服务器运行 CBC 解密程序并判断填充是否合法。

在本题中：如果不满足填充规则，则返回填充错误信息，即 HTTP 500 server error ；如果满足则填充规则，则返回正确信息，即 HTTP 200 。根据解密服务器反馈信息，进一步确定 CBC 解密中间状态值 a。

其中 a,r,y 的含义如下图所示：

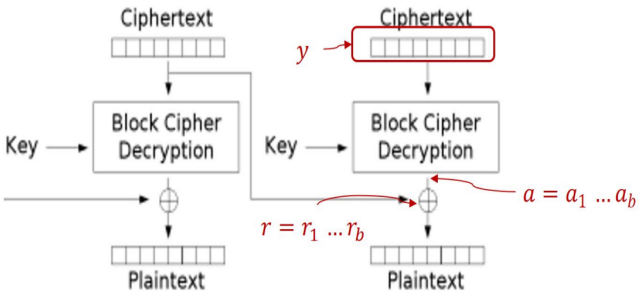


图 1: a,r,y

3.2.1 找到满足上式的 r

穷举搜索 r_b，并将对应的 rIly 发送给解密服务器，直到 CBC 解密反馈填充正确信息。此时对应的填充可能是 $0x01$ 或 $0x02||0x02$ 或者 $0x03||0x03||0x03$ 等情况。（注：当然，y 的真实明文填充可能并非此时对应的填充，该情况下很有可能找到的是只有 1 个字节填充，即 $a_b \oplus r_b = 0x01$。）

代码实现如下：

```
#1. 遍历找出解密后满足填充规则 r
rList=[0 for i in range(32)]
while(True):
  #找到服务器返回200的r
  (sta,output)=subprocess.getstatusoutput(path+List2Str(rList)+y)
  if(output==output1): break
  selfAdd(rList)
```

3.2.2 判断填充的长度并得到 $a_j \dots a_b$

首先修改 r_1，将 rlly 发送给解密服务器，如果服务器返回正确，说明填充长度不是 b，再修改 y_2；然后将 $r \| y$ 发送给解密服务器，如果服务器返回正确，说明填充长度不是 $b-1$；依此类推，直到解密服务返回填充错误，例如，修改 r_j 后，返回了错误，说明填充字节为 $a_j \dots a_b \oplus r_j \dots r_b$，填充字节数量为 $b-j+1$，此时 $a_j \dots a_b$ 可以根据填充规则计算得到。

具体代码实现如下：

```
#2. 确定该 rList 对应解密结果的填充数 length
length=16
for i in range(16):
  #2.1 修改rList
  rListChanged=rList.copy()
  rListChanged[2*i+1]+= 1 if rList[2*i+1]!=15 else -1

  #2.2 测试修改后rList是否满足要求
  (sta,output)=subprocess.getstatusoutput(path+List2Str(rListChanged)+y)
  if(output==output2):
    # 服务器第一次返回500 , 此时可计算出填充长度 length=16-i
    length-=i
    j=i
    break

#2.3 此时可计算中间值 a 的后length个Byte
aList=[0 for i in range(32)]
#对后length个字节:  a[]=r[] ^ length
for i in range(16,16-length,-1):
  XOR(aList,rList,i,length)
```

3.2.3 进一步得到 $a_1 \dots a_{j-1}$

具体方法：将 r 最后 $b-j+1$ 字节重新设置为 $r_k = a_k \oplus (b-j+2)$, for k= j, $\cdots$, b. 穷举搜索 r_{j-1}，将 $r \| y$ 发送给解密服务器，直到服务器返回填充正确，此时输出 $a_{j-1} = r_{j-1} \oplus (b-j+2)$. 依此类推，求出 $a_{j-1} \dots a_1$。

具体代码实现如下：

```
#3. 计算a的 (16-length)Byte部分,每次循环计算一个Byte
while(j!=0):
  #3.1 r最后 b-j 字节重设
  for i in range(16,16-(b-j),-1):
```

```
        XOR(rList,aList,i,(length+1))

    #3.2 穷搜r的前 j 字节,直到找到一个 r 令服务器返回200
    while(True):
        (sta,output)=subprocess.getstatusoutput(path+List2Str(rList)+y)
        if(output==output1): break
        selfAdd(rList,poi=2*(16-length)-1)

    #3.3计算a的第j字节:
    #a的第j字节=r的第j字节 ^ (length+1)
    XOR(aList,rList,j,(length+1))
    print("aList: ",aList)

    #准备进入下一轮, length+j==16 始终成立
    j-=1
    length+=1
```

3.2.4 恢复出本块的明文 m 及全部明文

根据截获的上一个分组密文 y'，计算 $y' \oplus a_1 a_b$，可得到本块的明文。而后通过相同的方法求出其他明文块从而级联出整个明文串。

具体代码实现如下：

```
#4. 当前块的中间值 a 已经求出, 可计算出明文m
print(f"Current Block Plaintext:")
print([int(ActualIv[i],16)^aList[i] for i in range(len(ActualIv))])
print(f"\n\n")
```

4 数据与结果

中间结果如下:

表 1: 密文 c3 破解中间结果

r1...r16	a1...a16	填充长度
000000000000000000000000000000d0	000000000000000000000000000000d1	1
0000000000000000000000000000a5d3	0000000000000000000000000000a7d1	2
00000000000000000000000000011a4d2	00000000000000000000000000012a7d1	3
000000000000000000000000bd16a3d5	000000000000000000000000b912a7d1	4
0000000000000000000000066bc17a2d4	0000000000000000000000063b912a7d1	5
00000000000000000000f65bf14a1d7	00000000000000000000963b912a7d1	6
000000000000000009e0e64be15a0d6	0000000000000000000990963b912a7d1	7
00000000000000000991016bb11aafd9	00000000000000000001990963b912a7d1	8
00000000000000aa0890006ab01baed8	00000000000000a301990963b912a7d1	9
00000000000076a90b930369b318addb	0000000000007ca301990963b912a7d1	10
0000000000db77a80a920268b219acda	0000000000d07ca301990963b912a7d1	11
00000000eedc70af0d95056fb51eabdd	00000000e2d07ca301990963b912a7d1	12
00000031efdd71ae0c94046eb41faadc	0000003ce2d07ca301990963b912a7d1	13
00002032ecde72ad0f97076db71ca9df	00002e3ce2d07ca301990963b912a7d1	14
00002133eddf73ac0e96066cb61da8de	000f2e3ce2d07ca301990963b912a7d1	15
a01f3e2cf2c06cb311891973a902b7c1	b00f2e3ce2d07ca301990963b912a7d1	16

表 2: 密文 c2 破解中间结果

r1...r16	a1...a16	填充长度
00000000000000000000000000000048	00000000000000000000000000000049	1
0000000000000000000000000000b44b	0000000000000000000000000000b649	2
00000000000000000000000000095b54a	00000000000000000000000000096b649	3
000000000000000000000000001792b24d	000000000000000000000000001396b649	4
0000000000000000000000ae1693b34c	0000000000000000000000ab1396b649	5
00000000000000000000037ad1590b04f	00000000000000000000031ab1396b649	6
0000000000000000003436ac1491b14e	0000000000000000003331ab1396b649	7
000000000000000053b39a31b9ebe41	0000000000000000d3331ab1396b649	8
00000000000000b9043a38a21a9fbf40	00000000000000b00d3331ab1396b649	9
0000000000006aba07393ba1199cbc43	00000000000060b00d3331ab1396b649	10
00000000006a6bbb06383aa0189dbd42	00000000006160b00d3331ab1396b649	11
000000000b6d6cbc013f3da71f9aba45	00000000076160b00d3331ab1396b649	12
000000030a6c6dbd003e3ca61e9bbb44	0000000e076160b00d3331ab1396b649	13
00002d00096f6ebe033d3fa51d98b847	0000230e076160b00d3331ab1396b649	14
00f52c01086e6fbf023c3ea41c99b946	00fa230e076160b00d3331ab1396b649	15
54ea331e177170a01d2321bb0386a659	44fa230e076160b00d3331ab1396b649	16

表 3: 密文 c1 破解中间结果

r1...r16	a1...a16	填充长度
000000000000000000000000000000a3	000000000000000000000000000000a2	1
0000000000000000000000000000001a0	0000000000000000000000000000003a2	2
00000000000000000000000000007700a1	00000000000000000000000000007403a2	3
000000000000000000000000177007a6	000000000000000000000000137403a2	4
0000000000000000000000b8167106a7	0000000000000000000000bd137403a2	5
00000000000000000000006abb157205a4	00000000000000000000006cbd137403a2	6
0000000000000000004d6bba147304a5	0000000000000000004a6cbd137403a2	7
000000000000000a94264b51b7c0baa	000000000000000a14a6cbd137403a2	8
00000000000007da84365b41a7d0aab	000000000000074a14a6cbd137403a2	9
0000000000007f7eab4066b7197e09a8	0000000000007574a14a6cbd137403a2	10
0000000000c97e7faa4167b6187f08a9	0000000000c27574a14a6cbd137403a2	11
00000000d4ce7978ad4660b11f780fae	00000000d8c27574a14a6cbd137403a2	12
000000bbd5cf7879ac4761b01e790eaf	000000b6d8c27574a14a6cbd137403a2	13
0000a1b8d6cc7b7aaf4462b31d7a0dac	0000afb6d8c27574a14a6cbd137403a2	14
0073a0b9d7cd7a7bae4563b21c7b0cad	007cafb6d8c27574a14a6cbd137403a2	15
1d6cbfa6c8d26564b15a7cad036413b2	0d7cafb6d8c27574a14a6cbd137403a2	16

C3 中间值: b00f2e3ce2d07ca301990963b912a7d1

C3 明文值: 64363234647d0a0a0a0a0a0a0a0a0a0a

C2 中间值: 44fa230e076160b00d3331ab1396b649

C2 明文值: 61336330396464353137383036336237

C1 中间值: 0d7cafb6d8c27574a14a6cbd137403a2

C1 明文值: 666c61677b3066383665373931383538

综上:

1. 带填充明文 (ASCII 码值)

666c61677b376233653835313035616364613537336235 61

64636337303536383136333039 7d0a0a0a0a0a0a0a0a0a

2. 去填充明文 (ASCII 码值)

666c61677b376233653835313035616364613537336235 61

6463633730353638313633303 97d

3. 明文 (译码后):

flag{7b3e85105acda573b5adcc7056816309}

参考文献

[1] Dougals R.Stinson. 密码学原理与实践: 第三版. 北京: 电子工业出版社,2009.7.

附录

实验代码, 使用 Python 实现.

```python
#orgaworl@outlook.com
import subprocess
path="./解密/Project2.exe "
output1="HTTP 200."
output2="HTTP 500 server error."
c="6b10ced1a3f2134c972f5b84224c369a25c9403e3e0504853c04099b25a5d47ed43
    91c0886ad76a90b930369b318addb1b7c352f3b15254a0385c5ac2c39792b"
t=len(c)

c3=c[len(c)-32:len(c)]
c2=c[len(c)-64:len(c)-32]
c1=c[len(c)-96:len(c)-64]
iv=c[0:len(c)-96]

def List2Str(rList):
    input=""
    for j in range(32):
        input+=hex(rList[j])[-1]
    return input

def selfAdd(rList,poi=31):
    while(True):
        if(poi==-1):
            break
        rList[poi]+=1
        if rList[poi]==16:
            rList[poi]=0
            poi-=1
        else:
            break
    return rList

def XOR(DST,SRC,ORDER,A):
    #按字节索引:  DST[ORDER]=SRC[ORDER]^A;
    if(A==16):
        DST[2*(ORDER)-1]=SRC[2*(ORDER)-1]
        DST[2*(ORDER)-2]=SRC[2*(ORDER)-2]^1
    else:
        DST[2*(ORDER)-1]=SRC[2*(ORDER)-1]^A
        DST[2*(ORDER)-2]=SRC[2*(ORDER)-2]

def lastBlock(ActualIv,y):
    print("new block: ")
    b=16
    j=15
    #1. 遍历找出解密后满足填充规则 r
    rList=[0 for i in range(32)]
```

```
48      while(True):
49          #找到服务器返回200的r
50          (sta,output)=subprocess.getstatusoutput(path+List2Str(rList)+y)
51          if(output==output1): break
52          selfAdd(rList)
53      print(List2Str(rList)," & ",end="")
54      #2. 确定该 rList 对应解密结果的填充数 length
55      length=16
56      for i in range(16):
57          #2.1 修改rList
58          rListChanged=rList.copy()
59          rListChanged[2*i+1]+= 1 if rList[2*i+1]!=15 else -1
60
61          #2.2 测试修改后rList是否满足要求
62          (sta,output)=subprocess.getstatusoutput(path+List2Str(rListChanged)+y)
63          if(output==output2):
64              # 服务器第一次返回500 , 此时可计算出填充长度 length=16-i
65              length-=i
66              j=i
67              break
68      #print(length)
69
70      #2.3 此时可计算中间值 a 的后length个Byte
71      aList=[0 for i in range(32)]
72      #对后length个字节:  a[]=r[] ^ length
73      for i in range(16,16-length,-1):
74          XOR(aList,rList,i,length)
75      print(List2Str(aList),f" & {length} \\\\")
76
77      #3. 计算a的 (16-length)Byte部分,每次循环计算一个Byte
78      while(j!=0):
79          #3.1 r最后 b-j 字节重设
80          for i in range(16,16-(b-j),-1):
81              XOR(rList,aList,i,(length+1))
82
83          #3.2 穷搜r的前 j 字节,直到找到一个 r 令服务器返回200
84          while(True):
85              (sta,output)=subprocess.getstatusoutput(path+List2Str(rList)+y)
86              if(output==output1): break
87              selfAdd(rList,poi=2*(16-length)-1)
88
89          #3.3计算a的第j字节:
90          #a的第j字节=r的第j字节 ^ (length+1)
91          XOR(aList,rList,j,(length+1))
92          print(List2Str(rList)," & ",end="")
93          print(List2Str(aList),f" & {length+1} \\\\")
94
95          #准备进入下一轮, length+j==16 始终成立
96          j-=1
97          length+=1
98
99      #4. 当前块的中间值 a 已经求出, 可计算出明文m
100     print(f"Current Block Plaintext:")
101     print([int(ActualIv[i],16)^aList[i] for i in range(len(ActualIv))])
102     print(f"\n\n")
103 lastBlock(c2,c3)
104 lastBlock(c1,c2)
105 lastBlock(iv,c1)
106 #50 48 50 49 48 48 52 54 48 49 49 54
```