Name: Kwa Li Ying
Student ID: 1003833

50.020 Network Security Lab 8: Wireless Security

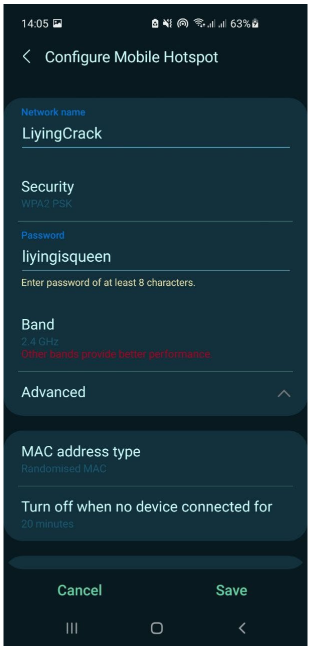
Task 1: Setup an Access Point

Step 1: Set up an access point

Using my android phone's mobile hotspot, I have set up the access point.

Step 2: Configure the SSID, username, and password

Unfortunately, I could not find a 'username' field in the hotspot configuration. The SSID is set to 'LiyingCrack' and the password is set to 'liyingisqueen' and as shown:



Network name
LiyingCrack

Security
WPA2 PSK

Password
liyingisqueen

Enter password of at least 8 characters.

Band
2.4 GHz
Other bands provide better performance.

MAC address type
Randomised MAC

Turn off when no device connected for
20 minutes

Step 3: Configure the security protocol to be WPA2

The security protocol is set to be WPA2 and it can be observed in the same screenshot above.

Name: Kwa Li Ying
Student ID: 1003833

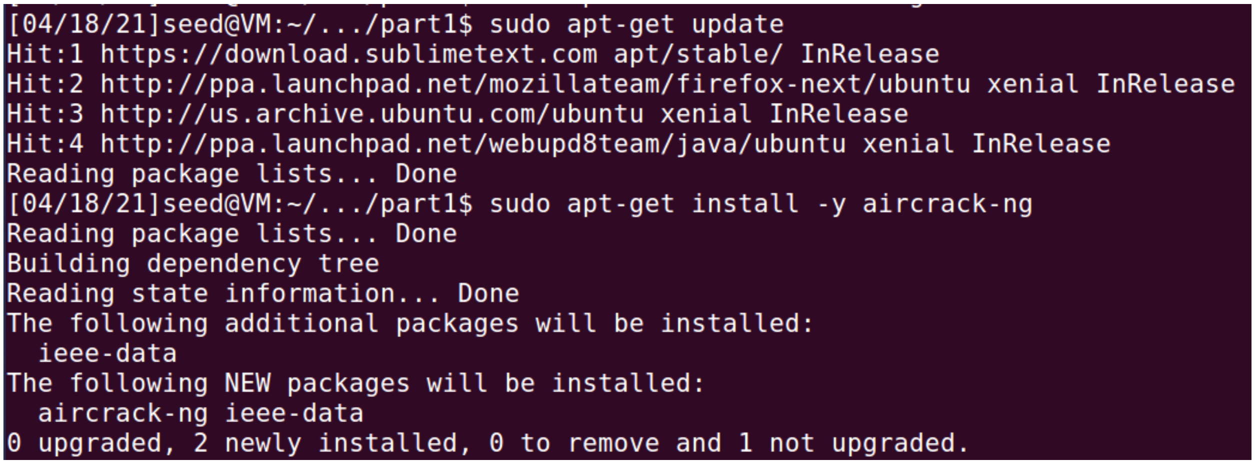
Task 2: Capturing Wireless Packets

This part is skipped due to hardware constraints.

Task 3: Capturing the Four-way Handshake

This part is largely skipped due to hardware constraints.

Aircrack-ng is installed on SEED using the following commands:



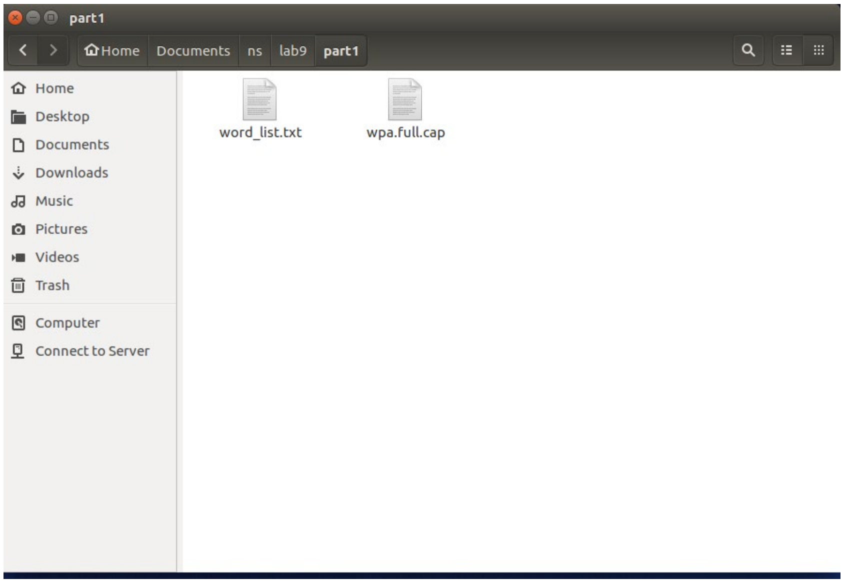
```
[04/18/21]seed@VM:~/.../part1$ sudo apt-get update
Hit:1 https://download.sublimetext.com apt/stable/ InRelease
Hit:2 http://ppa.launchpad.net/mozillateam/firefox-next/ubuntu xenial InRelease
Hit:3 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Hit:4 http://ppa.launchpad.net/webupd8team/java/ubuntu xenial InRelease
Reading package lists... Done
[04/18/21]seed@VM:~/.../part1$ sudo apt-get install -y aircrack-ng
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ieee-data
The following NEW packages will be installed:
  aircrack-ng ieee-data
0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.
```

Name: Kwa Li Ying
Student ID: 1003833

Task 4: Cracking WPA2 WiFi Passphrase Using Aircrack-ng

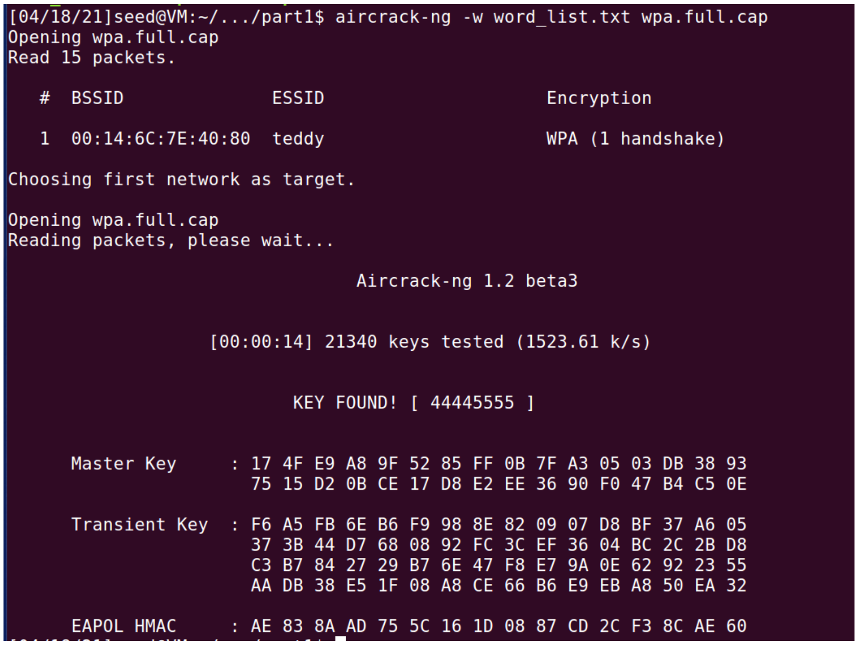
Step 1: Copy the cap/pcap file into the VM

The provided wpa.full.cap file word_list.txt word list is transferred to the VM:



Step 2: Use aircrack-ng to crack the passphrase

To crack the passphrase using the word list provided, the following command is run:



```
[04/18/21]seed@VM:~/.../part1$ aircrack-ng -w word_list.txt wpa.full.cap
Opening wpa.full.cap
Read 15 packets.

   #  BSSID              ESSID                     Encryption

   1  00:14:6C:7E:40:80  teddy                     WPA (1 handshake)

Choosing first network as target.

Opening wpa.full.cap
Reading packets, please wait...
                                 Aircrack-ng 1.2 beta3


                   [00:00:14] 21340 keys tested (1523.61 k/s)


                           KEY FOUND! [ 44445555 ]


      Master Key     : 17 4F E9 A8 9F 52 85 FF 0B 7F A3 05 03 DB 38 93
                       75 15 D2 0B CE 17 D8 E2 EE 36 90 F0 47 B4 C5 0E

      Transient Key  : F6 A5 FB 6E B6 F9 98 8E 82 09 07 D8 BF 37 A6 05
                       37 3B 44 D7 68 08 92 FC 3C EF 36 04 BC 2C 2B D8
                       C3 B7 84 27 29 B7 6E 47 F8 E7 9A 0E 62 92 23 55
                       AA DB 38 E5 1F 08 A8 CE 66 B6 E9 EB A8 50 EA 32

      EAPOL HMAC     : AE 83 8A AD 75 5C 16 1D 08 87 CD 2C F3 8C AE 60
```

The password set for the WiFi is shown to be '**44445555**'.

Name: Kwa Li Ying
Student ID: 1003833

Q: What is the difference between Monitor Mode and Promiscuous Mode?

Monitor mode: Sniffing the packets in the air without connecting (associating) with any access point.

Promiscuous mode: Sniffing the packets after connecting to an access point. This is possible because the wireless-enabled devices send the data in the air but only "mark" them to be processed by the intended receiver. They cannot send the packets and make sure they only reach a specific device, unlike with switched LANs.

Q: If the WiFi traffic is on-going, how to crack the WiFi password?

Force users to reconnect to the network so as to capture the 4-way handshakes during their reconnect. This can be done by using airodump-ng to monitor the target AP in the background, and then using aireplay-ng to deauthenticate the users. Reference: https://welkin.dev/2019/03/10/DOS-A-Router/

Name: Kwa Li Ying
Student ID: 1003833

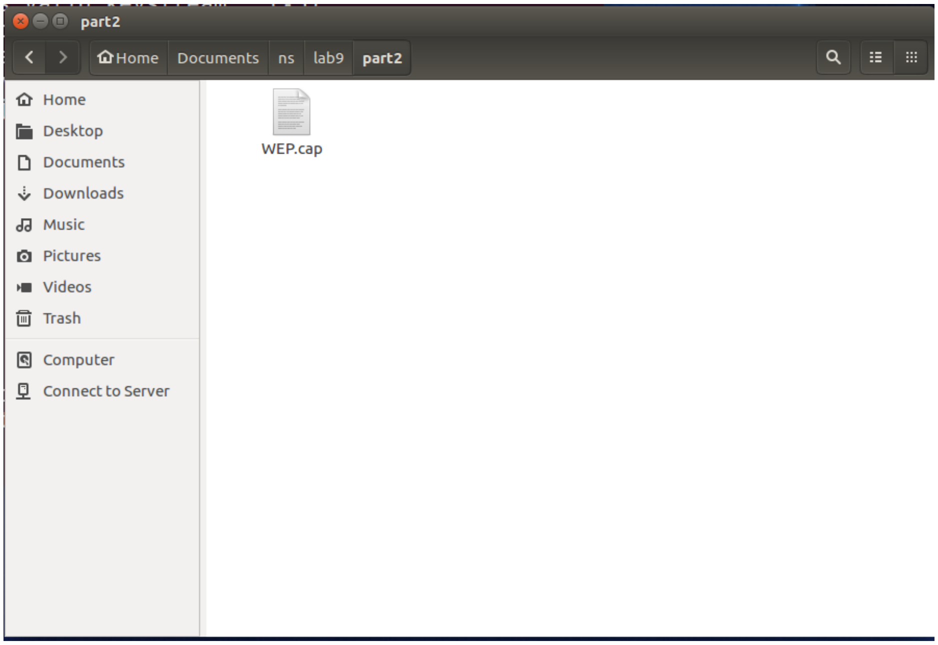
Task 5: Cracking the WEP Password

Step 1: Install Aircrack-ng

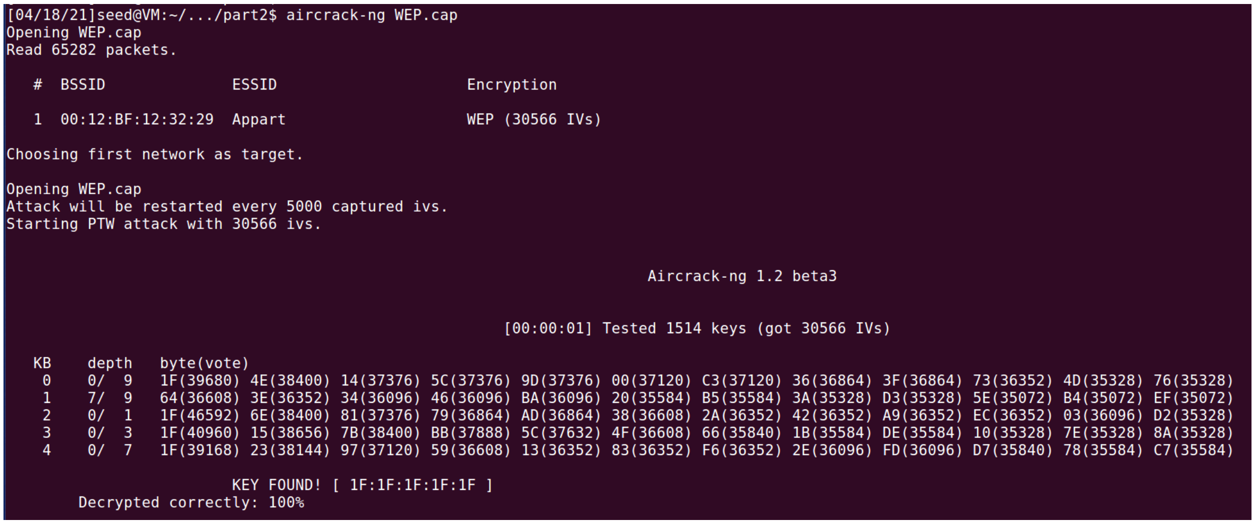
Done in the previous part.

Step 2: Cracking the WEP.cap

The provided WEP.cap file is transferred to the VM:



To crack the WEP protocol, the following command is run:



```
[04/18/21]seed@VM:~/.../part2$ 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) 00(37120) C3(37120) 36(36864) 3F(36864) 73(36352) 4D(35328) 76(35328)
    1    7/  9   64(36608) 3E(36352) 34(36096) 46(36096) BA(36096) 20(35584) B5(35584) 3A(35328) D3(35328) 5E(35072) B4(35072) EF(35072)
    2    0/  1   1F(46592) 6E(38400) 81(37376) 79(36864) AD(36864) 38(36608) 2A(36352) 42(36352) A9(36352) EC(36352) 03(36096) D2(35328)
    3    0/  3   1F(40960) 15(38656) 7B(38400) BB(37888) 5C(37632) 4F(36608) 66(35840) 1B(35584) DE(35584) 10(35328) 7E(35328) 8A(35328)
    4    0/  7   1F(39168) 23(38144) 97(37120) 59(36608) 13(36352) 83(36352) F6(36352) 2E(36096) FD(36096) D7(35840) 78(35584) C7(35584)

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

The cracked password/key is ‘**1F:1F:1F:1F:1F**’.

Name: Kwa Li Ying
Student ID: 1003833

Task 5: Cracking the WEP Packet

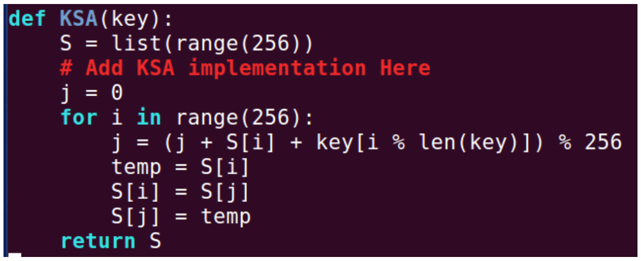
Step 1: Recall the WEP encryption process

Yup.

Step 2: Implement the RC4 Algorithm

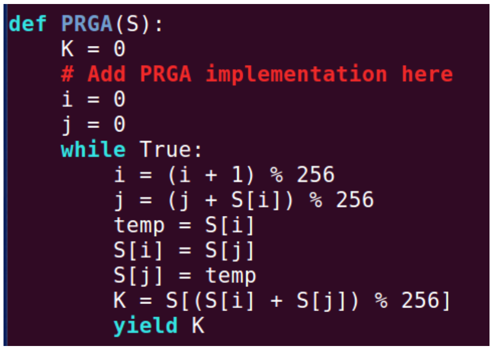
A full implementation of the code can be found in rc4.py.

The pseudocode given in the handout is used to fill in the code for KSA:



```
def KSA(key):
    S = list(range(256))
    # Add KSA implementation Here
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        temp = S[i]
        S[i] = S[j]
        S[j] = temp
    return S
```

The pseudocode given in the handout is used to fill in the code for PRGA:



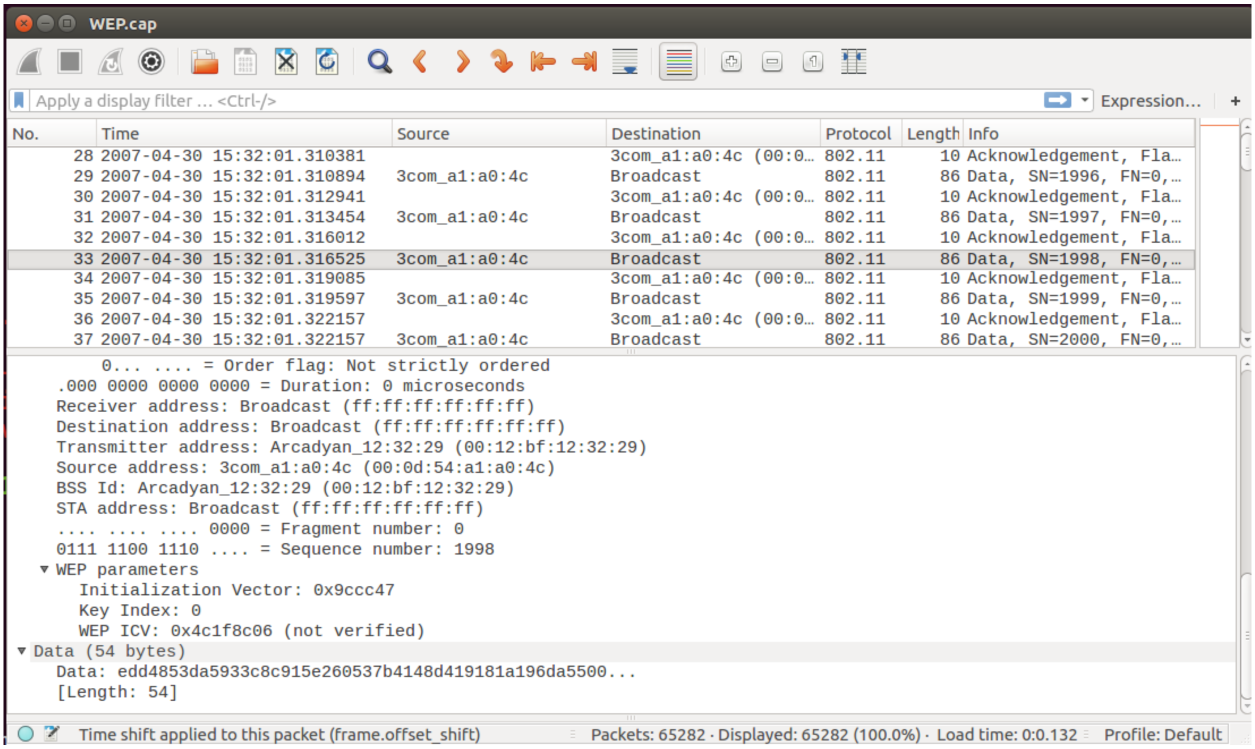
```
def PRGA(S):
    K = 0
    # Add PRGA implementation here
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        temp = S[i]
        S[i] = S[j]
        S[j] = temp
        K = S[(S[i] + S[j]) % 256]
        yield K
```

Step 3: Verify Your Results

The test cases given are used to verify that the RC4 algorithm is correct. Each key and ciphertext pair (edit line 41 for key and line 42 for ciphertext) give the correct plaintext (printed on line 52) stated.

For the CRC, the sequence number chosen is SN1998 (my birth year!). The information is as shown on Wireshark:

Name: Kwa Li Ying
Student ID: 1003833



```
No.  Time                          Source            Destination              Protocol Length Info
  28 2007-04-30 15:32:01.310381                      3com_a1:a0:4c (00:0…     802.11      10 Acknowledgement, Fla…
  29 2007-04-30 15:32:01.310894    3com_a1:a0:4c     Broadcast                802.11      86 Data, SN=1996, FN=0,…
  30 2007-04-30 15:32:01.312941                      3com_a1:a0:4c (00:0…     802.11      10 Acknowledgement, Fla…
  31 2007-04-30 15:32:01.313454    3com_a1:a0:4c     Broadcast                802.11      86 Data, SN=1997, FN=0,…
  32 2007-04-30 15:32:01.316012                      3com_a1:a0:4c (00:0…     802.11      10 Acknowledgement, Fla…
  33 2007-04-30 15:32:01.316525    3com_a1:a0:4c     Broadcast                802.11      86 Data, SN=1998, FN=0,…
  34 2007-04-30 15:32:01.319085                      3com_a1:a0:4c (00:0…     802.11      10 Acknowledgement, Fla…
  35 2007-04-30 15:32:01.319597    3com_a1:a0:4c     Broadcast                802.11      86 Data, SN=1999, FN=0,…
  36 2007-04-30 15:32:01.322157                      3com_a1:a0:4c (00:0…     802.11      10 Acknowledgement, Fla…
  37 2007-04-30 15:32:01.322157    3com_a1:a0:4c     Broadcast                802.11      86 Data, SN=2000, FN=0,…
```

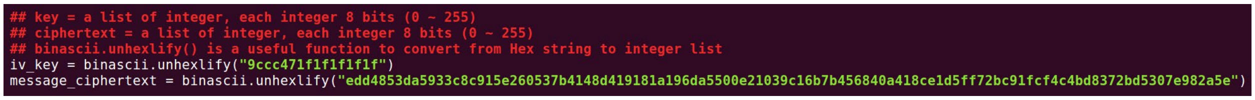
```
      0... .... = Order flag: Not strictly ordered
    .000 0000 0000 0000 = Duration: 0 microseconds
    Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
    Transmitter address: Arcadyan_12:32:29 (00:12:bf:12:32:29)
    Source address: 3com_a1:a0:4c (00:0d:54:a1:a0:4c)
    BSS Id: Arcadyan_12:32:29 (00:12:bf:12:32:29)
    STA address: Broadcast (ff:ff:ff:ff:ff:ff)
    .... .... .... 0000 = Fragment number: 0
    0111 1100 1110 .... = Sequence number: 1998
  ▼ WEP parameters
      Initialization Vector: 0x9ccc47
      Key Index: 0
      WEP ICV: 0x4c1f8c06 (not verified)
▼ Data (54 bytes)
    Data: edd4853da5933c8c915e260537b4148d419181a196da5500...
    [Length: 54]
```

```
Time shift applied to this packet (frame.offset_shift)    Packets: 65282 · Displayed: 65282 (100.0%) · Load time: 0:0.132    Profile: Default
```

The information is recorded down properly using (right-click) > Copy > …as a Hex Stream:

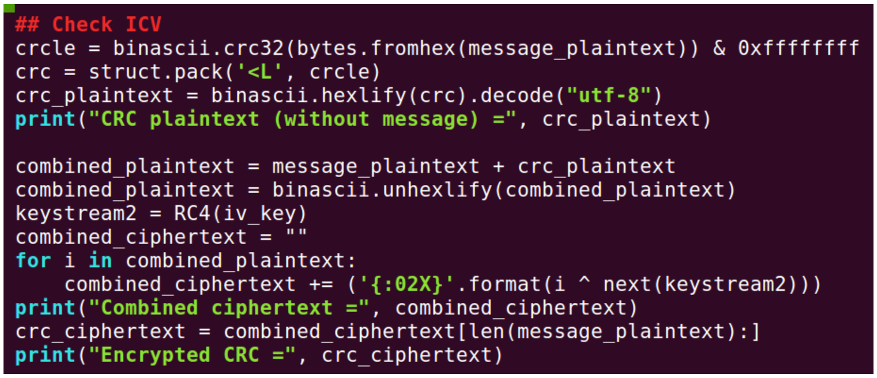
- IV: **9ccc47**
- Data (encrypted message): **edd4853da5933c8c915e260537b4148d419181a196da5500e21039c16b7b456840a418ce1d5ff72bc91fcf4c4bd8372bd5307e982a5e**
- Encrypted ICV: **4c1f8c06**

The IV is concatenated with the key (**1f1f1f1f1f**) to give the combined iv_key (**9ccc471f1f1f1f1f**) and the message_ciphertext is replaced with the data as shown:



```
## 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 = binascii.unhexlify("9ccc471f1f1f1f1f")
message_ciphertext = binascii.unhexlify("edd4853da5933c8c915e260537b4148d419181a196da5500e21039c16b7b456840a418ce1d5ff72bc91fcf4c4bd8372bd5307e982a5e")
```

The following code to compute the encrypted ICV is written as such:

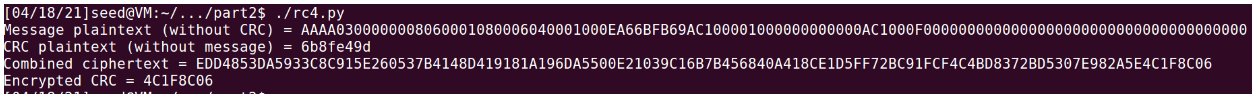


```
## Check ICV
crcle = binascii.crc32(bytes.fromhex(message_plaintext)) & 0xffffffff
crc = struct.pack('<L', crcle)
crc_plaintext = binascii.hexlify(crc).decode("utf-8")
print("CRC plaintext (without message) =", crc_plaintext)

combined_plaintext = message_plaintext + crc_plaintext
combined_plaintext = binascii.unhexlify(combined_plaintext)
keystream2 = RC4(iv_key)
combined_ciphertext = ""
for i in combined_plaintext:
    combined_ciphertext += ('{:02X}'.format(i ^ next(keystream2)))
print("Combined ciphertext =", combined_ciphertext)
crc_ciphertext = combined_ciphertext[len(message_plaintext):]
print("Encrypted CRC =", crc_ciphertext)
```

Name: Kwa Li Ying
Student ID: 1003833

The following output is shown:



```
[04/18/21]seed@VM:~/.../part2$ ./rc4.py
Message plaintext (without CRC) = AAAA0300000000806000108000604000 1000EA66BFB69AC100001000000000000AC1000F000000000000000000000000000000000000
CRC plaintext (without message) = 6b8fe49d
Combined ciphertext = EDD4853DA5933C8C915E260537B4148D419181A196DA5500E21039C16B7B456840A418CE1D5FF72BC91FCF4C4BD8372BD5307E982A5E4C1F8C06
Encrypted CRC = 4C1F8C06
```

To explain the output:

1. The message plaintext is fed into RC4 like before, and we get the following message_plaintext: **AAAA03000000008060001080006040001000EA66BFB69AC100001000000000000AC1000F000000000000000000000000000000000000**
2. The CRC plaintext is computed from the message plaintext, giving **6b8fe49d**
3. Concatenating the message plaintext and CRC plaintext, we get the combined plaintext. Feeding the combined plaintext into RC4, we get the combined ciphertext: **EDD4853DA5933C8C915E260537B4148D419181A196DA5500E21039C16B7B456840A418CE1D5FF72BC91FCF4C4BD8372BD5307E982A5E4C1F8C06**
4. Finally, we remove the message ciphertext from the front of the combined ciphertext to get the CRC ciphertext at the end: **4C1F8C06**

This final combined CRC ciphertext is compared with the encrypted ICV shown on Wireshark (**4c1f8c06**). Since they match, our RC4 algorithm is confirmed to be correct.

The corresponding cracked payload and ICV are as stated:

- Payload: **AAAA03000000008060001080006040001000EA66BFB69AC100001000000000000AC1000F000000000000000000000000000000000000**
- ICV: **6b8fe49d**