

## Lab 9

**Student ID: 1005288**

With the given skeleton code and the RC4 implementation, we can design the decryption algorithm. As shown, all test cases passed where the plaintext after decryption is the same as in the test cases, which implies that RC4 implementation is correct

```
####
RC4 TESTING
####
## Cracking the ciphertext
#   Several test cases: (to test RC4 implementation only)
#   1. key = '1A2B3C', ciphertext = '00112233' -> plaintext = '0F6D13BC'
#   2. key = '000000', ciphertext = '00112233' -> plaintext = 'DE09AB72'
#   3. key = '012345', ciphertext = '00112233' -> plaintext = '6F914F8F'
print("=====\nRC4 TEST CASES\n=====")
test_ciphertext = '00112233'
print(f"TEST CASE 1: {'0F6D13BC' == decrypt(key = '1A2B3C', ciphertext=test_ciphertext)}")
print(f"TEST CASE 2: {'DE09AB72' == decrypt(key = '000000', ciphertext=test_ciphertext)}")
print(f"TEST CASE 3: {'6F914F8F' == decrypt(key = '012345', ciphertext=test_ciphertext)}")
print("=====")
```

```
=====
RC4 TEST CASES
=====
```

```
TEST CASE 1: True
TEST CASE 2: True
TEST CASE 3: True
=====
```

After which, we start analysing the WEP packet capture using wireshark. I used SN=1982, IV is 0xcdd23a and encrypted ICV is 0x5db2d69a. These values will be used in our decryption and encryption algorithm. Alongside the information given from the aircrack output – WEP key is 1F1F1F1F1F.

No.	Source	Destination	Protocol	Length	Info
1		3Com_a1:a0:4c (00:...	802.11	10	Acknowledgement, Flags=.....
2	3Com_a1:a0:4c	Broadcast	802.11	86	Data, SN=1982, FN=0, Flags=.p....F.
3		3Com_a1:a0:4c (00:...	802.11	10	Acknowledgement, Flags=.....
4	3Com_a1:a0:4c	Broadcast	802.11	86	Data, SN=1983, FN=0, Flags=.p....F.
5		3Com_a1:a0:4c (00:...	802.11	10	Acknowledgement, Flags=.....
6	3Com_a1:a0:4c	Broadcast	802.11	86	Data, SN=1984, FN=0, Flags=.p....F.
7		3Com_a1:a0:4c (00:...	802.11	10	Acknowledgement, Flags=.....
8	3Com_a1:a0:4c	Broadcast	802.11	86	Data, SN=1985, FN=0, Flags=.p....F.
9		3Com_a1:a0:4c (00:...	802.11	10	Acknowledgement, Flags=.....
10	3Com_a1:a0:4c	Broadcast	802.11	86	Data, SN=1986, FN=0, Flags=.p....F.
11		3Com_a1:a0:4c (00:...	802.11	10	Acknowledgement, Flags=.....
12	3Com_a1:a0:4c	Broadcast	802.11	86	Data, SN=1987, FN=0, Flags=.p....F.
13		3Com_a1:a0:4c (00:...	802.11	10	Acknowledgement, Flags=.....
14	3Com_a1:a0:4c	Broadcast	802.11	86	Data, SN=1988, FN=0, Flags=.p....F.

  

> Frame Control Field: 0x0842  
 .000 0000 0000 = Duration: 0 microseconds  
 Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)  
 Transmitter address: ArcadyanTech\_12:32:29 (00:12:bf:12:32:29)  
 Destination address: Broadcast (ff:ff:ff:ff:ff:ff)  
 Source address: 3Com\_a1:a0:4c (00:0d:54:a1:a0:4c)  
 BSS Id: ArcadyanTech\_12:32:29 (00:12:bf:12:32:29)  
 STA address: Broadcast (ff:ff:ff:ff:ff:ff)  
 .... 0000 = Fragment number: 0  
 0111 1011 1110 .... = Sequence number: 1982  
 [WLAN Flags: .p....F.]

> WEP parameters  
 Initialization Vector: 0xcdd23a  
 Key Index: 0  
 WEP ICV: 0x5db2d69a (not verified)

> Data (54 bytes)  
 Data: c5e4b0c3ea87a1cd9b4b23f7076011ea0f8d89fb144430ab1b0bf44c2b32822881251e  
 [Length: 54]

```

0000 08 42 00 00 ff ff ff ff ff ff 00 12 bf 12 32 29  .B.....(2)
0010 00 0d 54 a1 a0 4c e0 7b cd d2 3a 00 c5 e4 b0 c3  ..T..L.:.:...
0020 ea 87 a1 cd 9b 4b 23 f7 07 60 11 ea 0f 8d 89 fb  ....K#.....
0030 14 44 30 ab 1b 0b f4 4c 2b 32 82 28 81 25 1e 3d  .D0...L +2.(.%=-
0040 08 29 91 5d 58 37 c2 d2 f7 ed ec 86 b6 d8 55 e1  .).]X7...U-
0050 66 8b 5d b2 d6 9a  f.].

```

The keystream is derived from the RC4(PRNG), which is built on `IV || KEY`, which will be used in our decryption algorithm. We know that the key stream will do bitwise XOR with the cipher text (both message and CRC/ICV) to recover the original text messages, this implies that we need to concatenate our encrypted ICV to the encrypted payload before decrypting.

The encryption was done with ICV and message, so to reverse this, we need to decrypt the data with ICV.

After decryption, we will get the ICV and message, and we know that the ICV is appended at the end of the decrypted payload.

Thus, we can splice the decrypted payload to right before the start of the ICV to find the message.

```

decrypted payload with crc: AAAA0300000008060001080006040001000EA66BF69AC100001000000000000AC1000F0000000000000
0000000000000000000000000000000006B8FE49D
crc from decrypted payload: 6B8FE49D
data without crc: AAAA0300000008060001080006040001000EA66BF69AC100001000000000000AC1000F0000000000000000000
0000000000000000

```

Next, to check that our decryption was done correctly, we use `binascii.crc32(message)` to generate a CRC to check that it is the same as the one above.

```
calculated crc using binascii.crc32: 6B8FE49D
```

We can see that they are indeed the same. Afterwards, we go through the encryption process again to check if our ICV is the same as the encrypted ICV.

```
newly encrypted payload: C5E4B0C3EA87A1CD9B4B23F7076011EA0F8D89FB144430AB1B0BF44C2B32822881251E3D0829915D5837C2  
D2F7EDEC86B6D855E1668B5DB2D69A  
newly encrypted payload ICV == 0x5db2d69a: True
```

As we can see, the ICV from the encrypted payload is the same as the one captured from wireshark.