Lab 9

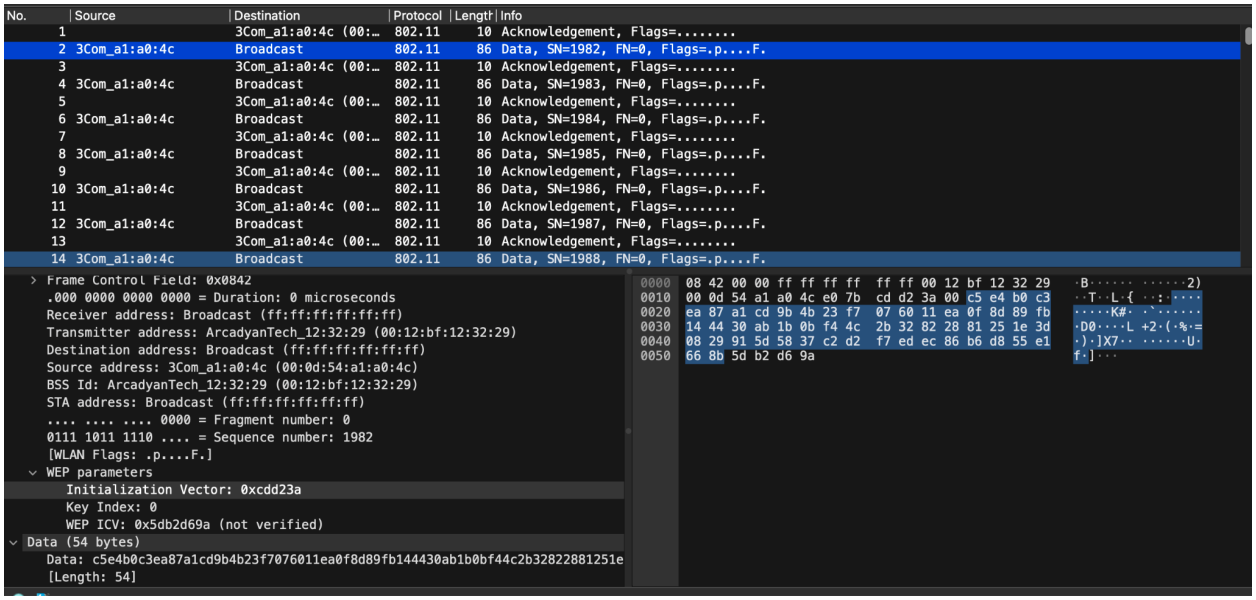
Student ID: 1005288

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', cipertext = '00112233' -> plaintext = '0F6D13BC'
#     2. key = '000000', cipertext = '00112233' -> plaintext = 'DE09AB72'
#     3. key = '012345', cipertext = '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.



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: AAAA0300000008060001080006040001000EA66BFB69AC10000100000000000AC1000F000000000000000000000000000000006B8FE49D
crc from decrypted payload: 6B8FE49D
data without crc: AAAA0300000008060001080006040001000EA66BFB69AC10000100000000000AC1000F00000000000000000000000000000000
```

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.