DPROT LAB WORK 1: Practical Block Ciphers

Ismael Douha Prieto

MCYBERS - Q1 2020/21

1 Introduction

In this report we will see how to use stream ciphers in a practical way, more exactly the stream cipher mode RC4. For do it, I used the library OpenSSL, which implements RC4. In addition, I performed a kind of attack to RC4, using OpenSSL for the most technical part and Python for the rest.

2 A (somewhat) practical attack against RC4

The attack that I performed in order to break RC4 is based on statistics, more precisely in the cipher value generated in function of the IV value. Although with high probability, the key guessed is the correct one, in some cases this does not happens, making it a non full reliable method.

For perform it, we need to assume several things:

- The key is a 16-byte string, which the first 3-byte are the IV and the last 13-byte are the long-term key.
- IV is incremented by 1 in every encryption.
- The message should have al least one byte.
- The attacker can access to many cipher messages, being able to wait until he detects some specific IV values.

The first part of this attack consists in guess the first byte of the plain text. Guessing it is possible because, with a high probability, when IV is equal to *01 FF X*, *C[0] XOR M[0] == X + 2*. Knowing it, now it is possible to compute all the possible values for X and the one that accomplish the statement in more cases, it will be the correct one.

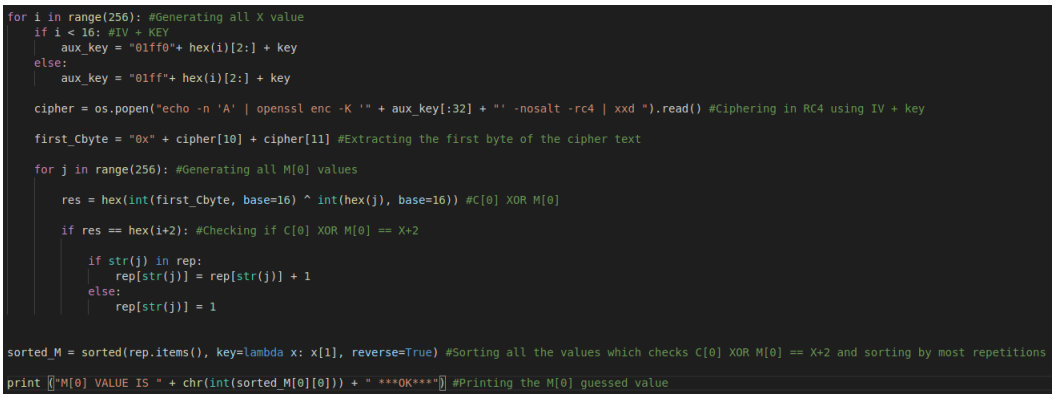

Figure 1: Code use to guess the *M[0]* byte. *Source: Myself.*

In Figure 1, It can be appreciate how, for every possible X value in IV, the cipher text generated "xored" with the guessed *M[0]*, is checked if it is equal to $X + 2$. For the ones that follows this rule, they are stored in a dictionary with the number of times that appears. Finally, they are sorted by repetitions and the first value is M[0] with a high probability.

Once *M[0]* is guessed, is it possible to guess the key used to cipher the plain text. For the first byte of the key (*K[0]*), when IV is *03 FF X*, the first byte of the result of "xor" *C[0]* and *M[0]* is, with high probability, $X + 6 + K[0]$. Computing it with all possible X values and extracting the most repeated value, with a bit of luck, you guessed correctly *K[0]*. For the rest of the long-term key, you can guessing it using the rule used in the previous byte, adding the first byte of the IV value used this time and the value of the byte of the key that are being guessed.

For example, to guess *K[1]*, you can use the rule of *K[0]* ($X + 6 + K[0]$) adding the first byte of IV (for each byte of the long-term key, you must add by 1 the first byte of IV) and the value of the byte of the key that are being guessed ($X + 10 + K[0] + K[1]$). Following this scheme, is it possible to guess the whole key.

```python
for k in range(3,16): #Guessing all the key
    num = num + k
    rep = {}

    for i in range(256): #Generating all X values to guess k[k-3]

        if i < 16: #IV + KEY
            aux_key = "0" + hex(k)[2:] +"ff0"+ hex(i)[2:] + key
        else:
            aux_key = "0" + hex(k)[2:] + "ff"+ hex(i)[2:] + key

        cipher = os.popen("echo -n 'A' | openssl enc -K '" + aux_key[:32] + "' -nosalt -rc4 | xxd ").read()#Ciphering in RC4 using IV + key

        first_Cbyte = cipher[10] + cipher[11]#Extracting the first byte of the cipher text

        res = hex(int(first_Cbyte, base=16) ^ 65) # C[0] XOR M[0]
        res = int(res, base=16) - i - num #Calculating the value that has the breaking point

        for val in deduced_key:#Calculating the value that has the breaking point
            res = res - int(val, base=16)

        while res < 0: #Making positive the value
            res = res + 256

        if len(str(hex(res))) < 4: #Adding the value to the list
            res = "0x0" + str(hex(res))[2]
            if res in rep:
                rep[res] = rep[res] + 1
            else:
                rep[res] = 1

        else:

            if str(hex(res)) in rep:
                rep[str(hex(res))] = rep[str(hex(res))] + 1
            else:
                rep[str(hex(res))] = 1

    sorted_K = sorted(rep.items(), key=lambda y: y[1], reverse=True)#Sorting all the values as guessed keys by most repetitions

    if len(str(sorted_K[0][0])[2:]) < 2: #Adapting the guessed value
        sorted_K[0][0] = "0x0" + str(sorted_K[0][0])[2:]

    if str(sorted_K[0][0])[2:] == key[checker:checker + 2]: #Printing if the guessed value is correct or not

        print ("K[" + str(k-3) + "] VALUE IS " + str(sorted_K[0][0]) + " ***OK***")
        deduced_key.append(sorted_K[0][0])
    else:
        print ("K[" + str(k-3) + "] VALUE IS " + str(sorted_K[0][0]) + " ***WRONG***")

    checker = checker + 2
```

Figure 2: Code use to guess the *K*. *Source: Myself.*

Figure 2 shows how the rule to guess the whole long-term key is generalized for every byte and how it is checked to ensure that worked properly.

```
ismael@MSI:~/Documentos/MCYBERS/DPROT/LAB1$ python3 breakRC4.py
The key is 5056c1e049111cd553181a0e28

The message ciphered is "A"

M[0] VALUE IS A ***OK***
K[0] VALUE IS 0x50 ***OK***
K[1] VALUE IS 0x56 ***OK***
K[2] VALUE IS 0xc1 ***OK***
K[3] VALUE IS 0xe0 ***OK***
K[4] VALUE IS 0x49 ***OK***
K[5] VALUE IS 0x11 ***OK***
K[6] VALUE IS 0x1c ***OK***
K[7] VALUE IS 0xd5 ***OK***
K[8] VALUE IS 0x53 ***OK***
K[9] VALUE IS 0x18 ***OK***
K[10] VALUE IS 0x1a ***OK***
K[11] VALUE IS 0x0e ***OK***
K[12] VALUE IS 0x28 ***OK***
ismael@MSI:~/Documentos/MCYBERS/DPROT/LAB1$
```

Figure 3: Results of the script execution. *Source: Myself.*

Finally, with a bit of luck, the script guess the whole long-term key correctly, as can be appreciate in Figure 3.

3 Bibliography

- https://web.mat.upc.edu/jorge.villar/doc/notes/DataProt/pract1.html