Tutorial #4

João Freitas up202100373
Rui Gonçalves up202103077

O presente relatório tem como objetivo descrever o processo seguido para a resolução da ficha `Tutorial #4`, disponibilizada no âmbito da disciplina de Criptografia Aplicada. As seções numeradas em baixo representam cada um dos exercícios resolvidos.

1) Stream Cipher RC4

A Alice e Bob concordaram em comunicar através de um canal de comunição cifrado com um esquema baseado no stream cipher RC4. Contudo, estes não querem produzir uma chave secreta a cada transmissão, reutilizando a mesma ao longo das várias sessões de comunicação. Para tal, estes concordaram em usar uma chave *k*, de 128 bits, e para cifrar uma mensagem *m*, o seguinte procedimento:

1. Escolher um valor aleatório *v*, de 80-bit;
2. Produzir a cifra da mensagem `c = RC4(v.k) XOR m`, sendo a concatenação de duas strings;
3. Enviar a bit string `(v.c)` sobre o canal de comunicação.

a) Supondo que a Alice envia a mensagem *m* cifrada ao Bob, seguinde o procedimento descrito, como pode este recuperar a mensagem original através de k e (v.c)?

O inverso de uma operação XOR é dado com o XOR do resultado, ou seja:

```
a XOR b = c
c XOR b = a
a XOR c = b
```

Neste modo, como `c = RC4(v.k) XOR m`, então podemos concluir que `m = RC4(v.k) XOR c`. Tendo (c.v) e sabendo que *v* tem o perído de 80 bits, então para obter *c* é necessário eliminar os últimos 80 bits presentes em (c.v).

Obtendo *c*, aplica-se este sobre a operação XOR a RC4(v.k) e obtem-se *m*.

b) Se um atacante observar vários valores da bit string produzida (v1.c1, v2.c2, ...), como pode este determinar que a mesma key-stream (output de RC4), foi usada para cifrar duas mensagem diferentes?

Como é usado sempre a mesma chave como input de RC4, então isso significa que para além de *v* e *m*, não existe nenhum atributo que diferencia as bit-strings produzidas. O output de RC4(v.k) é igual quando usado o mesmo valor aleatório *v*, pois a chave é igual para todas as sessões de comunicação. Com isto e seguindo o mesmo processo descrito anteriormente, é possível obter o valor de (v1, v2, ...) ao eliminar os primeiro 80 bits em (v.c), para se obter *c*. Uma vez sabendo *c*, determina-se o seu período, de forma a que seja possível eliminar este de (c.v), resultando no valor de *v*.

Sabendo este valor, o atacante apenas tem que observar uma repetição do mesmo de forma a comprovar que a mesma keystream foi usada para cifrar duas mensagems diferentes.

c) Apróximadamente, quantas mensagems terão que ser transmitidas, até que a mesma keystream seja usada duas vezes?

Uma vez que a keystream depende apenas do valor de *v*, que tem o período de 80-bit, então isso significa que, apróximadamente após `2^80/2 = 2^40` mensagens, irá ocorrer uma colisão.

d) Escreva um programa Python, que dado um valor *n*, calcula o valor menor presente num conjunto de valores produzidos aleatóriamente, de forma uniforme, tanto que hajam mais números aleatórios repetidos, do que não repetidos

Usando a biblioteca random e a estrutura *dict* do Python, é facil criar este programa, como demonstrado no seguinte code snippet:

```python
import random

def explore_until_more_repetitions_than_unique(n):
    generated_count_dict = dict()
    repeated = 0
```

```
    non_repeated = 0
    dont_stop = True

    while dont_stop:
        gen = random.randint(0, n)
        if gen in generated_count_dict:
            repeated += 1
        else:
            non_repeated += 1
            generated_count_dict[gen] = -1
        generated_count_dict[gen] = generated_count_dict[gen] + 1
        dont_stop = non_repeated > repeated
    return non_repeated + repeated

n = 100
smallest_number_until_repetition = explore_until_more_repetitions_than_unique(n)

print(smallest_number_until_repetition)
```

e) Quantas mensagems deve a Alice usar a mesma chave *k*, até produzir uma nova?

Uma vez reutilizando a mesma chave, torna-se muito mais facil obter o valor produzido por RC4(k.v), estando as colisões neste dependentes de *v*. A melhor solução seria a Alice produzir uma nova chave *k* a cada mensagem enviada. Contudo, como tanto a Alice e Bob querem evitar o envio repetido de novas chaves, podem ser enviadas tantas mensagens até que haja uma possível colisão (2^40).

2) Num dado ficheiro, existe uma lista de mensagens codificadas com os caratéres ASCII da língua portuguesa, por um stream cipher. Sabe-se que algumas destas mensagens foram cifradas com a mesma keystream. Como podemos identificar estas mensagens?

Sabendo que as mensagens estão codificas sobre o ASCII Português, então sabemos que o valor dos seus caratéres não pode ser superior ao valor 128. É tambem sabido que ao aplicarmos a operação XOR sobre um valor préviamente XORed, obtemos o valor iniciar. Então, isso significa que para uma stream cipher `C = P XOR KS` que utilize a mesma keystream:

```
C1 = P1 XOR KS1
C2 = P2 XOR KS1
C1 XOR C2 = P1 XOR P2
```

Sendo *P1i* e *P2i* carateres das mensagens P1 e P2 numa dada posição da mensagem *i*, então `P1i XOR P2i` nunca poderá ser maior que 128, pois os valores de P1i e P2i tem o valor igual ou menor que 128. Então, para comprovar que duas cifras usaram a mesma keystream, apenas é necessário percorrer todos os carateres desta e encontrar o primeiro cujo a operação XOR seja maior que 128. Uma vez encontrado, descartamos a possibilidade de ter sido utilizado a mesma keystream.

```
from data import *

def used_same_key_stream(l1, l2):
    zip_list = zip(l1, l2)
    return next((x for x in zip_list if int(x[0] ^ x[1]) >= 128), None) == None

packets_using_same_key_stream = set()

for i in range(0, len(packets)):
    for j in range(i+1, len(packets)):
        if(used_same_key_stream(packets[i], packets[j])):
            packets_using_same_key_stream.add(i)
            packets_using_same_key_stream.add(j)

print('Found the following packets using the same keystream: (# -> index)')
for x in packets_using_same_key_stream:
    print('Packet: #{}'.format(x))
```