Lorena Mamede Botelho
TAG: Criptografia

Relatório Referente aos Desafios 1, 2 e 3 do Set 1 do Cryptopals

Desafio 1:
Consistia em fazer a conversão de uma string em Hexa para uma em Base64. Para isso, usei a linguagem Python e o módulo Codecs (nesse e nos demais desafios). O módulo já oferece uma série de métodos prontos para conversão.

A primeira etapa para a resolução desse desafio era decodificar a String fornecida de Hexadecimal para dado binário.

```
decode_hex_string =
codecs.decode("49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f6973
6f6e6f7573206d757368726f6f6d", 'hex')
```

Se printarmos a mensagem decodificada, teremos a seguinte String:
>>> b"I'm killing your brain like a poisonous mushroom"

Indicando que estamos no caminho certo. Depois, nossa tarefa é codificar essa String em base 64.

```
encoded_b64_string = codecs.encode(decode_hex_string, 'base64')
```

Ao printarmos:
>>> b'SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t\n'

Que é a saída requisitada pelo desafio.

Desafio 2:
O Segundo desafio pede que a partir de duas strings em Hexadecimal, nós efetuemos a operação XOR, e o resultado obtido, devemos codificar novamente para Hexa.

Primeiro, decodificamos as duas Strings, como no Desafio 1.

```
hex_string1 = "1c0111001f010100061a024b53535009181c"
hex_string2 = "686974207468652062756c6c277320657965"
decode_hex_string1 = codecs.decode(hex_string1, 'hex')
decode_hex_string2 = codecs.decode(hex_string2, 'hex')
```

Depois, precisamo efetuar a operação XOR, que segue a seguinte lógica:

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

O Python já oferece um operador para isso que é sinalizado por ^. Basta fornecermos duas entrada inteiras A e B e efetuarmos A ^ B.

Como faremos isso em cada um dos Bytes das nossas duas Strings iniciais, optei por usar um *for* para isso.

```
for byte1, byte2 in zip(decode_hex_string1, decode_hex_string2):

      xord_string += bytes([byte1 ^ byte2])
```

Podemos observar que usei a função *zip(),* ela serve para, em cada *loop*, trabalharmos com *bytes* de posição correspondente em cada string.

Dentro do *for,* efetuo o XOR como explicado anteriormente e guardo o resultado em uma string previamente inicializada.

Se printarmos o resultado deste XOR, temos a seguinte mensagem:
>>> b"the kid don't play"

E por fim, codificamos novamente em Hexa.

Desafio 3:
O último desafio requer que decifremos uma string em Hexa que sofreu XOR com um caracter não informado. Devemos descobrir esse caracter e com uso dele decifrar a mensagem.

Primeira etapa, como sempre, é decodificar a string dada em Hexa, da mesma forma que fizemos nossos passos anteriores (usando *codec.decode()*).

Primeira observação feita é que ao aplicarmos novamente o XOR entre a string fornecida e o caracter usado, teremos a string original (que contém a mensagem).

Logo, por força bruta podemos efetuar XOR com todos os caracteres possíveis e analisar os resultados encontrados para saber se chegamos na mensagem desejada.

Esse caracter necessariamente vai estar compreendido entre 0 e 255, já que corresponde a 1 *byte*.

Assim, apostei em um *for* que percorre este intervalo.

```
for possivel_char in range(256):
```

Dentro dele, faremos o XOR com cada um dos bytes da string dada e o nosso possível caracter salvando o resultado em uma nova string.

```
for byte in decode_hex_string:

        xord_string += bytes([byte ^ possivel_char])
```

A cada *loop* do primeiro *for*, teremos um resultado fornecido pelo XOR do segundo, esse resultado pode ser ou não nossa mensagem original. Para descobrir, precisaremos analisá-lo. Seguindo a proposta do Desafio, realizaremos uma análise de ocorrência de letras do Inglês.

No dicionário abaixo, guardei todas as porcentagens de ocorrências de cada letra do alfabeto inglês. Esses dados podem ser encontrados facilmente na internet, a única alteração (tendenciosa) que fiz foi incluir o caracter especial * e lhe atribuir um valor negativo de ocorrência, para sinalizar a baixa prioridade de *strings* que o contivessem.

```
frequencia_letras_ingles = {

    'a' : 8.167,
    'b' : 1.492,
    'c' : 2.202,
    'd' : 4.253,
    'e' : 12.702,
    'f' : 2.228,
    'g' : 2.015,
    'h' : 6.094,
    'i' : 6.966,
    'j' : 0.153,
    'k' : 1.292,
    'l' : 4.025,
    'm' : 2.406,
```

```
    'n' : 6.749,
    'o' : 7.507,
    'p' : 1.929,
    'q' : 0.095,
    'r' : 5.987,
    's' : 6.327,
    't' : 9.356,
    'u' : 2.758,
    'v' : 0.978,
    'w' : 2.560,
    'x' : 0.150,
    'y' : 1.994,
    'z' : 0.077,
    '*' : -1000,
}
```

Ainda dentro do *for* inicial que escolhe o caracter possível, faremos um novo *for*, este responsável pela análise da porcentagem. Nele, buscaremos no dicionário *byte* a *byte* a porcentagem dos caracteres contidos na *string* que geramos anteriormente com o XOR. Adicionaremos essas porcentagens a um *array* **ocorrencias.**

```
for byte in xord_string:

        porcentagem = frequencia_letras_ingles.get(chr(byte), 0)
        ocorrencias.append(porcentagem)
```

Depois de coletarmos todas as porcentagens (atribuindo 0 aos caracteres que não estão no dicionário), as somaremos. Essa soma representa a probabilidade de uma determinada *string* ser a certa.

Portanto, fora desse último *for* realizamos a tal soma e, por meio da biblioteca Heapq, usaremos uma heap para guardar a soma obtida de cada string. A heap foi escolhida por ser uma estrutura de dados que trabalha com valores ordenados. Mais especificamente estamos interessados em usar a Max Heap que ordena em forma decrescente, o que facilita a busca pelo maior valor.

Como essa biblioteca apenas implementa a Min Heap (ordem crescente), usamos a multiplicação do valor por -1 para que se “torne” uma Max Heap.

```
heappush(heap, sum(ocorrencias) * -1)

possiveis_mensagens[sum(ocorrencias)] = xord_string
```

Adicionalmente, criamos um dicionário **possiveis_mensagens**, que guarda a *string* analisada associando como chave a soma das ocorrências.

Para obter a string certa basta pegar a que obteve maior probabilidade, ou seja, a de maior soma cujo valor vai estar no topo da nossa heap. De posse desse valor, a resgatamos do nosso dicionários de possíveis mensagens.

```
mensagem_original = possiveis_mensagens.get(-1 * heap[0])
```

O print retorna:
>>> b"Cooking MC's like a pound of bacon"