Universidade do Minho
Escola de Engenharia

One-Time Pad

Tecnologia Criptográfica
Trabalho Prático II

Trabalho realizado por:
Filipe Freitas (PG42828)
Maria Barbosa (PG42844)

Dezembro de 2020

Índice

Listagens

Capítulo 1

Introdução

1.1 Contextualização

Na sequência da UC de Tecnologia Criptográfica foi proposto o presente trabalho prático cujo o objetivo é analisar 20 criptogramas disponibilizados pelo docente. Cada um deles, foi gerado utilizando a cifra One Time Pad de forma incorrecta, nomeadamente porque se utilizou uma sequência pseudo-aleatória, gerada por um gerador aleatório fraco (o do python). Sabe-se que cada critprograma, pode corresponder a uma de duas mensagens de texto limpo, cifradas com chaves diferentes. A excepção de dois criptogramas que foram cifrados usando a mesma chave. Pretende-se com a análise, descobrir quais são esses dois criptogramas.

1.2 Estrutura do relatório

Este relatório divide-se em duas partes principais.

A primeira, correspondente a esta introdução e pretende contextualizar o trabalho e explicar a sua estrutura.

Na segunda parte, apresentamos uma breve explicação do trabalho realizado e dos métodos usados para descobrir os dois criptogramas cifrados com a mesma chave.

Capítulo 2

Trabalho desenvolvido

Antes de começar a procurar pelos criptogramas corretos, quisemos tentar perceber qual seria o padrão que iria aparecer nos criptogramas corretos. Assim, descobrimos que, se uma *pad* for repetida em dois criptogramas, o resultado da operação de decifragem do par correto de criptogramas é igual ao resultado da operação de decifragem já aplicada aos textos limpos correspondentes (Katz e Lindell 2014, 34). Assim, e juntando isso com o facto de que o *xor* de ambos os criptogramas nos diz exatamente onde é que esses criptogramas diferem (Katz e Lindell 2014, 34; davidlowryduda 2013), percebemos que deveríamos começar, então, por obter os *XORs* de todos os pares possíveis de criptogramas.

Foi, portanto, isso que fizemos:

Ficheiro *TP2.py*

```
49  xors = {}
50
51  # Para cada par de criptogramas, vamos descobrir qual é o seu xor e guardar no dicionário
    ↪ acima
52  for i in range(len(lines)):
53      for j in range(i + 1, len(lines)):
54          if i == j:
55              continue
56
57          x = lines[i]
58          y = lines[j]
59
60          xors[(x, y)] = (sxor(x, y))
```

Agora, foi necessário perceber o que é que estaríamos à espera que acontecesse no *XOR* correto. Apercebemo-nos que a probabilidade, em dois pedaços de texto aleatórios, de dois caracteres coincidirem é de $\frac{1}{26}$ (ou seja, para a mesma posição i, dadas duas strings s^1 e s^2, $P(s_i^1 = s_i^2) = \frac{1}{26}$). No entanto, a língua inglesa (língua fonte dos textos limpos) não é completamente aleatória: existem caracteres mais frequentes do que outros (a letra E, por exemplo, é a mais frequente de aparecer em qualquer texto

suficientemente grande).

Assim, e visto que os criptogramas originais têm um tamanho consideravelmente grande, e considerando que o *XOR* dos mesmos nos dá as diferenças entre os dois, esperamos ver, no *XOR* do criptograma correto, uma maior ocorrência de letras *A* do que nos restantes *XORs*. Assim sendo, fizemos uma análise de frequências em todos os *XORs*, e apresentamos os resultados com a seguinte ordenação:

- Para cada análise de frequências individual, os resultados desta são ordenados desde a letra mais frequente até à letra menos frequente;
- Para todos os *XORs*, a lista de *XORs* aparece ordenada pela frequência da sua letra mais frequente, em ordem decrescente também.

Assim, no topo da lista de frequências, aparece sempre o *XOR* cuja letra mais frequente dentro desse *XOR* foi também a letra mais frequente entre todos os *XORs*.

O código foi o seguinte:

Ficheiro *TP2.py*

```
63  # Para cada xor, vamos fazer uma análise de frequências nos seus caracteres
64  frequencies = OrderedDict(sorted({xor: list(frequency_analysis_blocks(xor, 1).items()) for
    ↪  xor in xors.values()}.items(), key = lambda x: x[1][0][1], reverse=True))
65  # Imprimir o top 5 dos caracteres mais frequentes em cada xor
66  pprint([x[:5] for x in frequencies.values()])
```

Este código apenas imprime o *top 5* das letras mais frequentes de cada *XOR*. No entanto, olhando para os resultados, é imediatamente aparente o padrão que estávamos à procura:

Resultado da análise de frequências

```
1  [[('A', 0.07), ('@', 0.04), ('E', 0.038), ('=', 0.036), ('B', 0.034)],
2   [('A', 0.047), ('@', 0.04), ('C', 0.037), ('>', 0.036), ('?', 0.036)],
3   [('@', 0.046), ('C', 0.041), ('A', 0.038), ('>', 0.037), ('?', 0.035)],
4   [('A', 0.046), ('@', 0.042), ('B', 0.04), ('?', 0.039), ('C', 0.035)],
5   [('A', 0.045), ('D', 0.036), ('C', 0.036), ('@', 0.034), ('=', 0.034)],
6   [('@', 0.044), ('A', 0.039), ('?', 0.036), ('C', 0.035), ('B', 0.034)],
7   [('B', 0.044), ('@', 0.038), ('C', 0.037), ('A', 0.035), ('E', 0.035)],
8   (...)
```

Assim sendo, só falta descobrir quais são os índices dos criptogramas que deram origem a este *XOR*.

Ficheiro *TP2.py*

```
68  # O xor correto, i.e., o xor correspondente aos dois criptogramas cifrados com a mesma chave,
    ↪  é o primeiro da lista
69  # de frequências calculada acima, visto que esta está ordenada por frequências descendentes
70  xor_correto = list(frequencies.items())[0][0]
71  print(xor_correto)
72
73  # Descobrir quais os índices dos criptogramas que deram origem ao xor correto
74  a, b = None, None
```

```
75  for (x, y), xor in xors.items():
76      if xor == xor_correto:
77          a, b = lines.index(x), lines.index(y)
78
79  # Imprimir os índices, seguido do texto criptogramas corretos
80  print(a, b, '\n\n')
81  print(lines[a], '\n\n', lines[b])
```

Temos, portanto, que os criptogramas corretos são o 5 e o 13, a contar a começar do 0. O texto desses criptogramas é o seguinte:

Início do criptograma 5

```
LXBVRMYMSDQVIHEQNCYLCOXRBZPELBIITFPILAXGJZZIQPLPZSSNIAOUXHNQPYJFJNVTPVLOGUURZUXJEYYVESBMAGZYPTPO
SWYIMLTYMYXSOWNHWAILNIVJVIFEALEMOHOUISEPGUBDPGYISMUVQAGSUOBUDVHOBZGLPRXPUQYNVQUGWIYODRPSPULDXHIB
IDBPTAICULWCCHWRJIUZAOODNGUYISOUEEIMIKUJRJXXCASTEOVICYSFLIRDNXFDAJWQKBOQXOLPYDJNWUSTGFVZPMZCDFRY
XIUYCXFNRZUTBWYYUIBRVTIEBYNQBWGAVADWJPWJLWRPTCFKIKAEHHDNVJTGJNZNSOSOFIEPOKPPZDCKCXJSYLRRLSXZPVUR
PWNRJDABCYWXUELQRCBHQEIABHYLTCIICPTFYIJINRJDOLLXMWWUICJAIEQOEDZVUFIWOYEDGGTOMDXHYYNWMHMXALXKOEEQ
(...)
```

Início do criptograma 13

```
HCOGIIGQWDPLUNEAGITPFSPXAFPEWEDRFJZSYWTRJPBXWUXWJNQTLVMUFEHFWHJFKCQDVFDKONOTXWKPECQVGQCAEWNNTUBW
NNWNEWOFHUVOEHJWTANZAYIWMJRHSGQWJOIIBFMPKULTKCSTSDOBTQSNSMTAPLAOGIEXTTXUDMCCHTUSEFIATDGBIKVQCHSX
KGLPSNLVJQHUNWKSLYNTAOUINOBPMDGCZMIUFOEZWFUXGBGXJEZEFTHNVZZUXDZMAQZLVLPTGOUXPXIKOVLECASQTFZYIWUH
JTBUGMTLHLMFASSIGXXCHKBVWMXJAKCPKWFNTHTREJBNCNFYNAFKNGXYFXPXKXECHSRWDUURWXHLLHLMAXCAFWZBRGZEDQKJ
CSTEROYMSFNBVKPDNWNZLMAKDFUSNLGYNLPTPAFMMRYPOBCJMULAMQOMRMDYEXPWQJRVNFQRBATOMDZVZUJMMHMXAIROPMRG
RNJSGJLFGYGUQEDRGLBHBYPMVLPRRSHDNRLRDEPKQTMYSIVHIUMATKSGCFDLQGFLITOIKRUAVLNGQFHFYQTQAJQJZMPBXJSW
(...)
```

Bibliografia

davidlowryduda. 2013. “Taking advantage of one-time pad key reuse?” Acedido a 6 de dezembro de 2020. https://crypto.stackexchange.com/a/108.

Katz, J., e Y. Lindell. 2014. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC.