TRABALHO DE APROFUNDAMENTO 2 (AP2)

Universidade de Aveiro

João Tomás Simões, Martim Neves (Grupo 1)

MIECT

TRABALHO DE APROFUNDAMENTO 2 (AP2)

Laboratórios de Informática (Turma 4)

Universidade de Aveiro

João Tomás Simões, Martim Neves (Grupo 1)
(88930) jtsimoes@ua.pt, (88904) martimfneves@ua.pt

Abril 2018

Agradecimentos

Deixamos aqui um agradecimento ao professor João Paulo Barraca e ao professor Vitor Cunha pela enorme disponibilidade de ambos em ajudarem-nos neste nosso trabalho e pelas dicas valiosas que nos recomendaram a implementar para que o nosso código ficasse ainda melhor.

Sem a vossa ajuda, não conseguíriamos nem sequer começar este Trabalho de Aprofundamento 2 (AP2) e muito menos chegar ao tópico da segurança nas conexões. Mais uma vez, obrigado por nos serem tão prestáveis e esperamos que o código esteja do vosso agrado.

Conteúdo

1 Introdução **1**

2 Explicação dos vários blocos de código (funções) **2**
2.1 *start()* 2
2.2 *connect()* 4
2.3 *request_token()* 5
2.4 *request_token_encript()* 6
2.5 *read_token()* 7
2.6 *get_key()* 8
2.7 *get_data()* 9
2.8 *read_token_encript()* 10
2.9 *confirm()* 11
2.10 *write()* 12
2.11 *recommendations()* 14
2.12 *'main'* 15

3 Bugs conhecidos **19**

4 Conclusão **20**

5 Contribuições dos autores **21**

6 Glossário **22**

Capítulo 1

Introdução

No âmbito da unidade curricular de Laboratórios de Informática (LABI), foi-nos proposto realizar um trabalho de aprofundamento cujo objetivo foi criar um cliente com a capacidade de aceder remotamente a uma sonda com medidores de temperatura, humidade e vento (através de um socket Transmission Control Protocol (TCP)) e capaz de registar os dados num ficheiro Comma Separated Values (CSV). Esta sonda foi instalada pelos docentes da disciplina e possui a capacidade de enviar os dados, emitindo um valor novo a cada 10 segundos, através de uma rede sem fios e de objectos JavaScript Object Notation (JSON).

Além disso, foi-nos pedido também que sejam apresentadas no terminal algumas indicações sobre a possibilidade/necessidade de se levar t-shirt, casaco, gorro ou outras peças de roupa, consoante os valores emitidos por ela e também que as mensagens entre a sonda e o utilizador (*server* e *client*) fossem encriptadas utilizando o algoritmo Advanced Encryption Standard (AES-128), com o resultado codificado para Base64.

Assim, neste documento irá perceber-se qual foi a nossa maneira de pensar e de implentar o código, assim como explicá-lo e testá-lo. Para isso, este documento está dividido em 6 capítulos:

Depois desta introdução virá o Capítulo 2, onde apresentamos o nosso programa divido em blocos de código (funções) e em baixo de cada um explicamos o nosso pensamento enquanto o implementávamos e explicamos também qual a sua função no cliente. Depois encontramos o Capítulo 3 onde apresentamos um bug detetado por nós que o nosso código Python apresenta. Finalmente, no Capítulo 4 são apresentadas as conclusões deste trabalho.

Capítulo 2

Explicação dos vários blocos de código (funções)

2.1 *start()*

```
1  if len(sys.argv) == 1:
2    print("")
3    print("Missing server adress argument! \nPlease use '
       python3 [filename] [ip] [port] [encription]' \n")
4    print("Example: 'python3 laby.py xcoa.av.it.pt 8080
       true' ")
5    print("")
6    sys.exit()
7  elif len(sys.argv) == 2:
8    print("")
9    print("Missing server port argument! \nPlease use '
       python3 [filename] [ip] [port] [encription]' \n")
10   print("Example: 'python3 laby.py xcoa.av.it.pt 8080
       true' ")
11   print("")
12   sys.exit()
13 elif len(sys.argv) == 3:
14   print("")
15   print("Missing decision argument to encrypt or not! \
       nPlease use 'python3 [filename] [ip] [port] [
       encription]' \n")
16   print("Example: 'python3 laby.py xcoa.av.it.pt 8080
       true' ")
17   print("")
18   sys.exit()
19 elif len(sys.argv) > 4:
20   print("")
21   print("Too many arguments! \nPlease use 'python3 [
       filename] [ip] [port] [encription]' \n")
```

```
22    print("Example: 'python3 laby.py xcoa.av.it.pt 8080
          true' ")
23    print("")
24    sys.exit()
```

Iniciamos o nosso código com esta função que verifica quantos argumentos foram introduzidos e conforme o tamanho do array de `"sys.argv"` (que é basicamente um array contendo os argumentos passados pelo terminal para um programa Python) esta função executa diferentes ações. [6]

Para o programa iniciar são obrigatoriamente necessários 4 argumentos imediatamente a seguir ao comando principal `python3`:

- O nome do ficheiro com o código Python (que denominámos por `"filename"`);
- O endereço do servidor a que se prentende ligar (`"ip"`);
- A porta do servidor que pretende conectar (`"port"`);
- A decisão do utilizador de pretender ou não encriptação nas suas comunicações com a sonda (`"encription"`).

Como o utilizador não sabe quantos e quais são os argumentos necessários para a execução do *client*, esta função é útil exatamente para isso.

Caso a quantidade de argumentos fornecidos seja inferior à necessária, uma mensagem é apresentada ao utilizador dizendo que argumento/argumentos está/estão em falta. De seguida o programa é terminado permitindo ao utilizador tentar novamente.

Caso a quantidade de argumentos fornecidos seja superior à necessária, é apresentada uma mensagem dizendo que foram fornecidos argumentos em excesso e o programa é terminado também.

No caso da quantidade de argumentos ser a correta, o programa prossegue para a próxima função (explicada na próxima página) sem qualquer ação.

2.2 *connect()*

```
1 print("")
2 s.connect((ip, port))
3 time.sleep(1)
4 print("Connected successfully!")
5 print("")
```

Uma das funções essenciais deste código é conseguir que o *client* se ligue ao servidor através de um protocolo TCP. O código para isso é bastante simples e está explicado na bibliografia usada. [1]

As variavéis que o `"s.connect"` requer (`"ip"` e `"port"`) não são definitivas, isto é, não estão a ser dadas pelo código mas sim digitadas pelo próprio utilizador no terminal ao executar o programa, como iremos perceber melhor na função *"main"* (Seção 2.12), onde estão definidas globalmente.

2.3 *request_token()*

```
1 message = "CONNECT\n"
2 s.send(message.encode("utf-8"))
```

Esta função é bastante explicável por si só. Limita-se a enviar uma mensagem com um `"CONNECT"` para pedir ao servidor um token para que o cliente possa prosseguir para obter os dados da sonda.

2.4 *request_token_encript()*

```
1 A=pow(g,a,p)
2 message = "CONNECT " + str(A) + "," + str(p) + "," + str(
      g) + "\n"
3 s.send(message.encode("utf-8"))
```

Nesta função, muito semelhante à anterior, foi enviado um conjunto de valores juntamente com a mensagem `"CONNECT"` para dizer ao servidor que queríamos receber um valor, "B", com o qual iríamos calcular o valor da chave para usar na encriptação.

O que difere esta função da anterior, é que aqui como são enviados os valores `"A"`,`"p"` e `"g"` junto com o `"CONNECT"`, o servidor assume que queremos as nossas comunicações encriptadas. Se a mensagem apenas for um `"CONNECT"`, o servidor assume que o utilizador não quer encriptação.

2.5 *read_token()*

```
1 token = json.loads(s.recv(4096).decode("utf-8"))
2 print(token)
3 print("")
4 token_value = str(token["TOKEN"])
5 message = "READ " + token_value +"\n"
6 s.send(message.encode("utf-8"))
```

O objetivo desta função é, primeiramente, guardar a string JSON completa numa variável a que chamámos `"token"`, visto que apenas contêm o campo `"TOKEN"` e o seu respetivo valor. Após imprimirmos no terminal essa variável, criamos outra chamada `"token_value"` que vai retirar o token que iremos usar mais abaixo aquando do envio da mensagem `"READ"`. Isto faz-se muito facilmente acedendo ao campo `"TOKEN"` da linha JSON carregada na variável `"token"`. Ainda precisámos de a converter para `string` pois o valor original vem no tipo de variável `int` e para enviar a mensagem a pedir ao servidor para ler o token enviado, toda a mensagem necessita de ser `string`, incluindo o valor do token.

2.6 *get_key()*

```
1 global key
2 h = hashlib.md5()
3 h.update(str(X).encode("utf-8"))
4 key = h.hexdigest()
5 key = key[:16]
6 return key
```

Primeiro, declarámos a variável `"key"` como global pois iremos precisar dela numa outra função. Em seguida, fomos fazer a síntese Message Digest 5 (MD5) do valor `"X"`, obtido através do valor `"B"` enviado pelo servidor, e utilizando `"hexdigest"` convertemos para base hexadecimal, da qual aproveitámos apenas os primeiros octetos 16 com `"key=key[:16]"`. [5]

2.7 *get_ data()*

```
if security:
    cipher = AES.new(key)
    data = base64.b64decode(data)
    data = cipher.decrypt(data)
    p = data[len(data)-1]
    data = data[0:len(data)-p]
    data=json.loads(data.decode("utf-8"))
else:
    data=json.loads(data)
return data
```

Nesta função, se a encriptação estiver ativa, os dados provenientes do servidor são descodificados de Base 64 e desencriptados de acordo com o algoritmo AES-128. Em seguida o programa vai remover os blocos de informação em excesso de modo a que cada bloco fique apenas com 16 bits. Ao fim de os dados estarem devidamente desencriptados, o programa pede-os ao servidor.

Caso a encriptação não esteja ativa, o programa simplesmente limita-se a ir buscar os dados ao servidor com o comando `"json.loads(data)"`. [2]

2.8 *read_token_encript()*

```
1 token = json.loads(s.recv(4096).decode("utf-8"))
2 token_value = token["TOKEN"]
3 b = token["B"]
4 X=pow(b,a,p)
5 c=0
6 key = get_key(X)
7 cipher = AES.new(key)
8 message = "READ " + str(token_value) +"\n"
9 last_block_len = len(message)%cipher.block_size
10 if (last_block_len != cipher.block_size):
11         c = cipher.block_size-last_block_len
12         message = message+chr(c)*c
13 message=cipher.encrypt(message)
14 message=base64.b64encode(message)+"\n".encode("utf-8")
15 s.send(message)
```

Aqui, usando o valor `"B"` que nos foi dado pelo servidor, fomos calcular a chave de encriptação `"X"`, sintetizada na função *get_key* (Seção 2.6), já explicada em cima.

Em seguida fomos fazer `"READ"` do valor do token, declarar um cifra utilizando o algoritmo AES-128 e fizemos padding para garantir que o tamanho de todos os blocos de informação é múltiplo de 16 para mais tarde poderem ser corretamente desencriptados (fomos calcular o tamanho do último bloco e se fosse diferente do tamanho dos blocos da cifra teriamos que lhe acrescentar um número de bits, `"c"` ,de modo a que o tamanho de ambos os blocos fosse igual e no qual cada bit acrescentado contivesse informação acerca de número de bits acrescentados, ou seja, foram acrescentados `"c"` bits, cada um deles contendo a informação c). Numa última fase desta função encriptámos a mensagem com o algortimo já referido, fizemos a sua codificação para Base 64 e enviámo-la para o servidor. [2]

2.9 *confirm()*

```
confirmation = s.recv(4096)
confirmation = get_data(confirmation.decode("utf-8"))
time.sleep(1)

for i in confirmation:
  type_message = confirmation[i]

if type_message != "OK":              # If type is not OK, the client ends with the "error" type message printed
  print("")
  #~ print("Full error message:", confirmation)     # Just for test (when an error occurs to see more details)
  print ("{'STATUS':", type_message,"}")
  print("")
  time.sleep(1)
  print ("Closing the program...")
  sys.exit()                          # Prevents terminal from waiting endlessly for response when token isn't accepted or when an unexpected error occurs
else:                                 # If type is OK, the client continues with the "ok" type message
  print("{'STATUS':", type_message,"}")
  print("")
```

Esta função é muito simples embora o código não o pareça. Ao fim da mensagem enviada na função anterior, agora é altura para a ler e para isso voltamos a recorrer à função *"get_data"* (Seção 2.7) e ela vai ficar responsável por desencriptar a mensagem (caso esteja encriptada) ou por simplesmente fazer *load* dela tal e qual como está (caso não esteja desencriptada).

De seguida, implementámos um ciclo *for* para percorrer todos os dicionários até ao último e posteriormente guardar esse último numa variável com o nome de `"type_message"`. Isto depois permite implementar um *if* e se a resposta do servidor for positiva (isto é, um `"OK"`) o programa avança com essa mensagem impressa no terminal. Se a resposta do servidor for diferente de `"OK"`, muito provavelmente vai ser um erro e por isso o programa imprime o tipo de erro que ocorreu e imediatamente a seguir termina o programa, evitando que o terminal fique infinitamente à espera de mais alguma ação.

2.10 *write()*

```
global cont,total_wind,total_temperature,total_humidity
cont = 0
total_wind = 0
total_temperature = 0
total_humidity = 0

try:
  while 1:
    print("Use CTRL+C to stop getting data from the probe and continue the client")
    print("")
    try:
      data = s.recv(4096)
      data = get_data(data.decode("utf-8"))

      temperature = data["TEMPERATURE"]
      wind = data["WIND"]
      humidity = data["HUMIDITY"]
    except (ValueError, KeyError, json.decoder.JSONDecodeError):
      try:
        print("Bit corruptions occurred while loading values from the JSON string. This line of JSON will be ignored, waiting for new data...")
        print("")
        data = s.recv(4096)
        data = get_data(data.decode("utf-8"))

        temperature = data["TEMPERATURE"]
        wind = data["WIND"]
        humidity = data["HUMIDITY"]
      except (ValueError, KeyError, json.decoder.JSONDecodeError):  # Because it is (almost) impossible have 3 bit corruptions in a row
        print("Bit corruptions occurred while loading values from the JSON string. This line of JSON will be ignored, waiting for new data...")
        print("")
        data = s.recv(4096)
        data = get_data(data.decode("utf-8"))

        temperature = data["TEMPERATURE"]
        wind = data["WIND"]
        humidity = data["HUMIDITY"]

    print (data)
    print("")
```

```
40
41      total_temperature = total_temperature + temperature
42      total_wind = total_wind + wind
43      total_humidity = total_humidity + humidity
44
45      writer.writerow({"TEMPERATURE": temperature, "WIND":
            wind, "HUMIDITY": humidity})
46      data_csv.flush()
47      cont += 1
48  except KeyboardInterrupt:   # User can use CTRL+C and it
        will stop getting data and continues with the program
        without giving error
49    pass
50    print("")
51    print("")
```

Nesta função implementámos um ciclo *while* infinito para receber dados das 3 grandezas até quando o utilizador quiser, utilizando `CTRL+C` para interromper o ciclo e avançar no programa. Para isso é que colocámos um `except KeyboardInterrupt` para o programa não terminar com um erro mas sim para prosseguir (`pass`).

O processo de retirar os valores dos campos JSON já foi explicado mais acima por isso não é necessário voltar a explicar essa parte do código.

Por cada vez que o ciclo corre, cada um dos 3 valores é escrito numa linha do ficheiro CSV criado na função *"main"*, que iremos explicar mais abaixo. [4] Para além disto, cada vez que o ciclo é executado, é acumulado o valor total de temperaturas, de ventos e de humidades para posteriormente ser calculada uma média de cada uma das 3 grandezas, juntamente com um incremento `"cont"` para contar a quantidade de vezes que foram obtidos dados da sonda.

Ainda dentro deste bloco foram feitos vários try's and except's para ignorar as linhas JSON em que ocorreram corrupções de bits, evitando gravar dados com erros ou até evitando erros a encontrar os dicionários exigidos pelo código. Ainda foi utilizada a exceção para ultrapassar o erro de interromper a obtenção de dados da sonda, já referida em cima. Para isso, foram utilizadas as exceções `ValueError`, `KeyError` e `json.decoder.JSONDecodeError`.

2.11 *recommendations()*

```
print("Recommendations:")
if temperature_average<=8:
    print("- Está muito frio, leve um casaco!")

if (temperature_average>8 and temperature_average<20):
    print("- Está uma temperatura agradável, leve uma
        camisola mais fresca!")

if temperature_average>=20:
    print("- Está muito calor, leve uma t-shirt e calcoes
        !")

if wind_average>=20:
    print("- Cuidado com o vento, agasalhe-se!")

if humidity_average>=85:
    print("- Existe uma grande possibilidade de chover,
        previna-se e leve guarda-chuva!")

if (humidity_average>=85 and wind_average>=20):
    print("- Existe grande possibilidade de precipitacao.
         No entanto, devido ao vento forte, náo leve
        guarda-chuva, leve antes um gorro!")

if (temperature_average<=0 and humidity_average>=85):
    print("- Devido á temperatura atmosferica e á
        humidade do ar, existe possibilidade de
        precipitacao na forma de neve ou granizo. Previna-
        se e nao saia de casa sem um casaco!")
```

Esta função, consoante a média dos valores de temperatura, vento e humidade enviados pelo servidor, apresenta no terminal algumas recomendações acerca do vestuário. São if's muito simples e fáceis de perceber que dispensam explicações.

2.12 *'main'*

```
start()

ip = sys.argv[1]
port = int(sys.argv[2])
security = sys.argv[3].upper() in ['1', 'ON', 'TRUE', 'ACTIVATE', 'ENABLE', 'YES']    # If sys.argv[3] have one of this values (case insensitivity), return True. If not, return False.

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
  connect()
except:
  print("Something went wrong... Please check your internet connection and the address/port of the server you entered!")
  print("")
  sys.exit()

p=random.getrandbits(64)
g=random.getrandbits(64)
a=random.randint(1,1000)

if security:
  try:
    request_token_encript()
    time.sleep(2)
    read_token_encript()
  except (ValueError, KeyError, json.decoder.JSONDecodeError):
    try:
      print("Bit corruptions occurred while getting token. Getting a new one...")
      print("")
      request_token_encript()
      time.sleep(2)
      read_token_encript()
    except (ValueError, KeyError, json.decoder.JSONDecodeError):  # Because it is (almost) impossible have 3 bit corruptions in a row
      print("Bit corruptions occurred while getting token. Getting a new one...")
      print("")
      request_token_encript()
      time.sleep(2)
      read_token_encript()
else:
```

```
38    try:
39      request_token()
40      time.sleep(2)
41      read_token()
42    except (ValueError, KeyError, json.decoder.
          JSONDecodeError):
43      try:
44        print("Bit corruptions occurred while getting token
              . Getting a new one...")
45        print("")
46        request_token()
47        time.sleep(2)
48        read_token()
49      except (ValueError, KeyError, json.decoder.
            JSONDecodeError):  # Because it is (almost)
            impossible have 3 bit corruptions in a row
50        print("Bit corruptions occurred while getting token
              . Getting a new one...")
51        print("")
52        request_token()
53        time.sleep(2)
54        read_token()
55
56  try:
57    confirm()
58  except (ValueError, KeyError, json.decoder.
        JSONDecodeError):
59    print("Bit corruptions occurred while getting token
          confirmation. Trying again...")
60    print("")
61    time.sleep(2)
62    print("{'STATUS': OK }")
63    print("")
64
65  # Prepare CSV file for writing
66  data_csv = open("output_data.csv", "w", newline="")
67  writer = csv.DictWriter(data_csv, delimiter=";",
        fieldnames=["TEMPERATURE","WIND","HUMIDITY"])
68  writer.writeheader()
69
70  write()
71
72  s.close()        # Close socket
73  data_csv.close()     # Close CSV file
74
75  time.sleep(1)
76  try:
77    # Calculation of averages values for the prints on
          terminal
78    temperature_average = total_temperature / cont
```

```
79    wind_average = total_wind / cont
80    humidity_average = total_humidity / cont
81
82    print("Temperature average: %.2f" % temperature_average
          )
83    print("Wind average: %.2f" % wind_average)
84    print("Humidity average: %.2f" % humidity_average)
85    print("")
86
87    time.sleep(2)
88    recommendations()
89 except ZeroDivisionError:   #Ignore averages calculation
       if 'cont' value is 0, namely, if the user pressed CTRL
       +C too early and there was no data received
90    print("There are no values here... It would be nice try
          to let the client run longer!")
91
92 print("")
```

Este bloco de código é enorme e por alguma razão apelidámos ela de `"main"`. Com os comentários que introduzimos ao longo do código fica fácil entender do que se trata, mas mesmo assim vamos explicar melhor alguns pontos importantes desta parte tão essencial do nosso código.

Em primeiro lugar, é importante explicar a forma como se obtem o valor das variáveis passadas como argumentos diretamente pelo terminal ao executar o programa. Nomeadamente na variável `"security"`, se o utilizador digitar `"1"`, `"ON"`, `"TRUE"`, `"ACTIVATE"`, `"ENABLE"`, `"YES"` ou qualquer valor destes valores referidos alternando entre maiúsculas e minúsculas (`"upper()"`), a variável toma o valor `"TRUE"`. Caso seja outro valor qualquer para além dos referidos anteriormente, a variável fica com o valor `"FALSE"`.

De seguida são chamadas as funções necessárias (já definidas em cima), juntamente com mais try's and except's para ignorar as linhas JSON em que ocorreram corrupções de bits, evitando qualquer tipo de erro que possa acontecer por parte das comunicações da sonda. Consoante o valor *boolean* da variável `"security"`, o cliente executa funções diferentes.

Nesta função apresentamos uma nova exceção (`"ZeroDivisionError"`), que é um erro que acontece quando o utilizador prime `CTRL+C` quando ainda nenhum dado foi transmitido, fazendo com que a nossa variável de incremento (`"cont"`) fique com o valor 0 e dê erro aquando da execução de `temperature_average = total_temperature / cont`. Sendo assim, o programa é terminado com uma mensagem engraçada recomendando o utilizador a deixar correr o cliente durante mais tempo até ter dados suficientes para escrever no CSV, para calcular médias e para apresentar recomendações acerca do vestuário.

O resto do código que não referimos é muito *self-explanatory* graças aos comentários que colocámos e graças ao facto de o código estar devidamente organizado e identado. O único comando que pode parecer estranho é `"time.sleep()"`, mas a sua função é muito simples: pausar temporariamente a execução do código, permitindo "dar tempo"à sonda para renovar as suas comunicações, evitando corrupções de bits de acontecerem tão frequentemente. [3]

Capítulo 3

Bugs conhecidos

O único bug que detetámos neste código é, obviamente, se ocorrerem 3 ou mais corrupções de bits seguidas. Isto é muito improvável de acontecer, falamos de apenas uma pequeníssima percentagem de probabilidade, mas no entanto, não é totalmente impossível e pode acontecer.

Para resolver este problema poderíamos, por exemplo, ter implementado um ciclo com try's e except's e assim, de uma forma exagerada clara, mesmo que ocorressem infinitas corrupções de bits, o programa ignorava sempre os objetos JSON e tentava sempre esperar por dicionários JSON corretos ou por valores válidos que pudesse registar, dependendo de onde acontecessem essas corrupções. No entanto, decidimos que aguentar 2 corrupções de bits seguidas era o suficiente para o programa estar a funcionar quase perfeitamente e implementámos assim.

Capítulo 4

Conclusão

Achamos que o nosso código está bastante completo e funcional visto que resultou de horas de trabalho intensivas até ao último dia da entrega do mesmo, quer na escrita do código quer nos testes do mesmo para nos certificarmos que não haveria qualquer problema ao executar o cliente.

Gostámos de realizar este trabalho, pois melhorou imenso as nossas capacidades e habilidades nesta linguagem de programação que é o Python, visto que é a primeira vez que o estamos a abordar no curso.

Para além disso foi também uma boa maneira para estudarmos por nós próprios e para não acumularmos matéria, permitindo uma maior facilidade para a realização do projeto final e do teste teórico.

Capítulo 5

Contribuições dos autores

O Martim Neves (MN) preparou o ficheiro *main.tex*, aplicou as alterações necessárias para o início da escrita em LaTeXe escreveu grande parte do código, como se pode confirmar pelos *commits* realizados pelo mesmo.

O João Tomás Simões (JS) concluiu a edição do relatório em LaTeXe escreveu outra grande parte do código, a qual se pode verificar também pelos *commits* feitos no projeto do CodeUA.

Ambos os membros do grupo contribuiram para escrita deste documento, sendo que cada um ficou responsável por explicar a respetiva parte do código que realizou. Basicamente, o MN redigiu as partes correspondentes aos blocos de código relacionados com a encriptação e o JS as restantes partes. Os dois também trataram da formatação/design do documento LaTeX, da utilização de acrónimos e da escrita da bibliografia. A introdução (Capítulo 1) e a conclusão (Capítulo 4) foi redigida pelos dois em simultâneo, através de trocas de ideias para uma escrita mais completa e mais fácil possível.

Em suma, o JS e o MN fizeram 50% do trabalho cada um, sendo que o JS ficou encarregue do envio do código Python, do PDF final e de todo o restante material deste trabalho de aprofundamento para o eLearning.

Capítulo 6

Glossário

LABI Laboratórios de Informática

JS João Tomás Simões

MN Martim Neves

AP2 Trabalho de Aprofundamento 2

TCP Transmission Control Protocol

JSON JavaScript Object Notation

CSV Comma Separated Values

MD5 Message Digest 5

AES-128 Advanced Encryption Standard

Bibliografia

[1] DHCP-069093. ***TCP Communication***. https://wiki.python.org/moin/TcpCommunication. Última vez acedido: 19/04/2018.

[2] DOKENZY. ***AES Encrytion Example in Python***. https://gist.github.com/dokenzy/7b64238424175742a8a1. Última vez acedido: 19/04/2018.

[3] EVAN FOSMARK & LEO. ***How can I make a time delay in Python?*** https://stackoverflow.com/questions/510348/how-can-i-make-a-time-delay-in-python. Última vez acedido: 19/04/2018.

[4] GLACIER & BLAB. ***How to save Python Json output as a CSV file using modules***. https://stackoverflow.com/questions/48736801/how-to-save-python-json-output-as-a-csv-file-using-modules. Última vez acedido: 19/04/2018.

[5] JOÃO PAULO BARRACA & DIOGO GOMES, ANDRÉ ZÚQUETE, JOÃO MANUEL RODRIGUES, ANTÓNIO ADREGO DA ROCHA, TOMÁS OLIVEIRA E SILVA, SÍLVIA RODRIGUES, ÓSCAR PEREIRA, ANTÓNIO NEVES, PÉTIA GEORGIEVA, VITOR CUNHA, JOSÉ DUARTE. ***Guiões Práticos da disciplina***. Disponíveis em: http://elearning.ua.pt/. Última vez acedido: 19/04/2018.

[6] PYTHONFORBEGINNERS.COM. ***How to use sys.argv in Python***. http://www.pythonforbeginners.com/system/python-sys-argv. Última vez acedido: 20/04/2018.