Laboratórios de Informática 2019–2020
DETI - Universidade de Aveiro

Trabalho de aprofundamento 2

Objetivos:

- Planeamento e preparação do trabalho de aprofundamento 2.

Este guião apresenta as regras o segundo trabalho de aprofundamento. Os exercícios sugeridos dão os primeiros passos para a concretização do trabalho recorrendo às ferramentas estudadas.

1.1 Regras

O trabalho deve ser realizado por um grupo de 2 alunos e entregue, via a plataforma `https://elearning.ua.pt`, dentro do prazo lá indicado. A entrega deverá ser feita por **apenas um dos membros** do grupo e deve consistir de **um único arquivo** `.zip`, `.tgz` (TAR comprimido por `gzip`) ou `.tbz` (TAR comprimido por `bzip2`).

O arquivo deve conter o código desenvolvido assim como o ficheiro PDF final do relatório, todos ficheiros com código fonte (`.tex`, `.bib`, etc.) e todas as imagens ou outros recursos necessários à compilação do código ou documento.

Na elaboração do relatório recomenda-se a adopção do estilo e estrutura de relatório descrito nas aulas teórico-práticas e a utilização de recursos de escrita como: referências a fontes externas, referências a figuras e tabelas, tabela de conteúdo, resumo, conclusões, etc. O objectivo do relatório é descrever a motivação, a implementação (não é só o código, mas também o algoritmo), apresentar testes que comprovem o seu funcionamento correto e analisar os resultados obtidos.

É obrigatório incluir uma secção "Contribuições dos autores" onde se descrevem resumidamente as contribuições de cada elemento do grupo e se avalia a percentagem de trabalho de cada um. Esta auto-avaliação poderá afetar a ponderação da nota a atribuir a cada elemento.

1.2 Avaliação

A avaliação irá incidir sobre:

1. **cumprimento dos requisitos apresentados**,
2. **qualidade do código produzido e comentários**,
3. **testes unitários e funcionais realizados**,
4. **o suporte de segurança adicionado**,
5. **estrutura e conteúdo do relatório**,
6. **utilização das funcionalidades de tarefas do Code@ua e git**.

Relatórios meramente descritivos sem qualquer descrição da aplicação, apresentação dos resultados obtidos, testes efetuados, ou discussão serão fracamente avaliados.

Só serão avaliados trabalhos enviados via a plataforma `https://elearning.ua.pt`. Ficheiros corrompidos ou inválidos não serão avaliados à posteriori e não será permitido o reenvio.

Deve ser utilizado um projeto na plataforma Code.UA, com um identificador segundo o formato `labi2020-ap2-gX`. **Substitua o carácter X pelo número 1. Se não for possível criar este projeto, incremente o número até que ele seja aceite. Não use valores aleatórios. Não se esqueça de incluir todos os professores da UC como membros do projeto (não necessitam de ser administradores)**

1.3 Tema Proposto

Imagine que tem um conjunto de pessoas que pretende seriar aleatoriamente para uma determinado fim. Por exemplo, para uma etapa de ciclismo em contra-relógio, ou para uma etapa de rali. E nenhuma das pessoas a serem seriadas confia em terceiros para fazerem a seriação de forma isenta. O que fazer, neste caso, para cada saber o seu número de ordem, diferente dos demais, e garantir que o mesmo foi gerado de forma absolutamente aleatória e isenta?

O objetivo deste trabalho é criar um serviço que suporte a criação de ordenações de pessoas e de aplicações cliente que apoiem as pessoas. As aplicações cliente usam interfaces abertas, i.e., disponíveis publicamente, pelo que cada pessoa é livre de fazer a sua aplicação cliente, caso não confie noutras.

O processo de criação possui 3 etapas:

1. As pessoas juntam-se a um processo de ordenação fornecendo um elemento de identificação conhecido pelos demais (nome, alcunha, número único num determinado contexto, etc.).

2. As pessoas participam num processo colaborativo de ordenação mútua, onde cada um obtém aleatoriamente o seu número de ordem.

3. As pessoas revelam o seu número de ordem e todos podem comprovar que não há números coincidentes.

A ordem final deverá constar de um relatório em formato CSV (`report.csv`), criado pelos clientes. Em caso de erro na produção do relatório, os clientes podem protestar e mostrar uma evidência que suporte a sua insatisfação.

O processo de distribuição aleatória dos números de ordem por um conjunto de N participantes num processo de ordenação deverá ser o seguinte:

1. O servidor gera uma sequência de números de ordem (lista de valores entre 1 e N) e passa-a a um dos participantes.

2. Este cifra-a com uma chave simétrica gerada na hora, aleatoriamente, que guarda para mais tarde reutilizar (K). A sequência cifrada é então baralhada pelo participante, e o resultado é enviado ao servidor. Nota: para permitir a baralhação dos valores cifrados, estes têm de ser cifrados independentemente uns dos outros!

3. O servidor repete o processo anterior com os demais participantes. No final, o servidor possui uma sequência de números de ordem cifrados N vezes, com N chaves individuais, mas ninguém sabe que número de ordem representa cada valor cifrado.

4. O servidor circula os números de ordem cifrados pelos participantes. A ordem destes não é relevante. Cada um retirará um elemento cifrado (C).

5. O servidor pede a cada participante que se comprometa com o número de ordem (ainda desconhecido) que selecionou aleatoriamente. Para isso, cada participante i deverá produzir e devolver ao servidor o valor C_i que retirou anteriormente e uma síntese de C_i e da chave K_i que usou para produzir C_i (e não só). Chamaremos B_i a este valor resultante (de *bit commitment*).

$$B_i = \text{digest}\,(C_i, K_i)$$

6. O servidor envia todos os pares $\langle C_i, B_i \rangle$ recebidos para todos os participantes, devidamente associados ao seu nome. Cada participante pode verificar se o valor de C corresponde ao que escolheu e B ao que gerou. Em caso de inconformidade, deverá retirar-se do processo de ordenação indicando a razão para a autoexclusão.

Note-se que se algum participante se excluir no final deste passo, os demais são inúteis porque o protocolo não conseguirá evoluir.

7. O servidor pede a todos os participantes que lhe enviem as suas chaves K.

8. O servidor envia a todos os participantes todos os trios $\langle C_i, B_i, K_i \rangle$. Cada participante pode verificar se a sua chave K está certa, bem como todos os outros valores de C e B (seu e dos demais). Em caso de erro poderá ser elaborado um protesto. Caso contrário, cada participante poderá verificar os valores B, poderá decifrar os valores C e poderá obter a ordem de cada participante. O número de ordem de cada um deverá ser um valor entre 1 e N, sem repetições. Tudo isto deverá ser verificado.

9. A ordem final é enviada pelo servidor e verificada por todos os participantes. Em caso de inconformidade com a ordem obtida por um participante, este pode indicá-lo à pessoa que representa.

A comunicação entre clientes e servidores deverá ser suportada por *sockets* UDP ou TCP. O uso de TCP tem a vantagem de permitir detetar a falha de um interlocutor aquando do uso do *socket* com a ligação para com o mesmo.

O cliente possui os seguintes requisitos funcionais:

1. Se for iniciado com um número de argumentos inferior a 3, este deverá imprimir a ajuda no seguinte formato:
   `client.py id_processo_ordenação id_pessoal porto [máquina]`;

2. Os dois primeiros argumentos são sequências arbitrárias de caracteres;

3. O segundo argumento (identificador pessoal) pode ter um valor especial, **a definir pelos alunos**, para referir um participante especial relativo a um processo de ordenação que apenas gere o mesmo, não sendo seriado. Cabe a este participante a criação de um processo de ordenação, a obtenção da lista de participantes já inscritos e dar a ordem para se iniciar o processo de ordenação.

4. O terceiro argumento deverá ser um valor inteiro positivo, especificando o porto (UDP ou TCP) do servidor.

5. O quarto argumento deverá ser um nome DNS ou endereço IPv4 no formato X.X.X.X, onde X representa um valor inteiro decimal entre 0 e 255. Se este argumento não for indicado, o cliente deverá usar um servidor na mesma máquina onde está (`localhost`);

6. Se for iniciado com um qualquer argumento inválido (tipo errado, valor incorreto ou não encontrado), deverá ser apresentada uma mensagem de erro respetiva ao argumento que gera o erro;

7. Caso não seja possível contactar o servidor indicado, o programa deve apresentar uma mensagem a explicar o problema e terminar;

8. O ficheiro `report.csv` deverá possuir a seguinte estrutura: número de ordem, identificador pessoal, número de ordem cifrado (C), chave (K), *bit commitment* (B). De forma a permitir a sua validação por terceiros, a ordem das linhas deverá refletir a ordem por que foram obtidos, por cifra, os valores C, e não a ordem crescente ou decrescente dos números de ordem obtidos.

1.4 Protocolo utilizado

O protocolo utilizado pelos servidores e clientes é da exclusiva responsabilidade dos alunos. Porém, recomenda-se o uso de TCP para detetar a falha dos interlocutores.

Aconselha-se também que as mensagens sejam estruturadas em JSON. As mensagens JSON podem ser facilmente convertidas de e para dicionários Python.

Atenção, porém, ao facto de os dicionários suportarem quaisquer valores (associados a chaves textuais), nomeadamente vetores de octetos (*bytes*), o que não é suportado pelo JSON. Este problema coloca-se quando é preciso lidar com criptogramas ou sínteses. Para o resolver, estes valores podem ser guardados nos dicionários usados para estruturar mensagens como valores textuais, usando, por exemplo, o formato Base64 (ver Figura 1.1).

Na Secção 1.6 são fornecidos exemplos de funções que enviam e recebem dicionários Python como objetos JSON através de *sockets* TCP. Estes exemplos podem ser usados tal qual ou modificados.

Na Secção 1.7 é fornecido um esqueleto da parte do servidor que lida com a receção de pedidos de ligações de clientes e com a receção de mensagens dos clientes.

1.5 Notas importantes

A cifra dos números de ordem deverá ser feita com cifras simétricas por blocos (o uso de cifras contínuas ou de fluxo é desaconselhado porque liberta informação). Nessas cifras não deverá ser usado alinhamento (*padding*), porque o mesmo é inútil. Porém, cada número de ordem deverá ser guardado num bloco completo a ser processado pela cifra por blocos (que deverá operar em modo ECB). Na Figura 1.2 é mostrado como se pode cifrar e decifrar 1000 vezes um número respeitando o alinhamento imposto pelas funções de cifra.

Embora a diversidade seja possível, o servidor e todos os clientes deverão usar as mesmas funções de cifra e síntese. Recomenda-se o uso de AES-128 e SHA-256, respetivamente.

Deve ser utilizado o módulo `csv` para processar valores tabulados em CSV.

```
import base64
from Crypto.Hash import SHA256

msg = 'This is a message that is going to be hashed with SHS-256'

hash_f = SHA256.new()
hash_f.update( bytes(msg, 'utf8') )
digest = hash_f.digest()

b64_digest = base64.b64encode( digest )
recovered_digest = base64.b64decode( b64_digest )

if digest == recovered_digest:
    print( 'Success!' )
else:
    print( 'Failure, %s is different from %s' % (digest, recovered_digest) )
```

Figura 1.1: Codificação e descodificação de um valor binário (neste caso, do resultado de uma função de síntese) com Base64

```
import os
from Crypto.Cipher import AES

iterations = 1000
engines = []

# Use index number 12, create a 128-bit array with it

index = 12
_128_bit_padded_index = bytes("%16d" % (index), 'utf8')

# Use the 128-bit array as the input to the multiple ciphering

data = _128_bit_padded_index
engines = []

for i in range(iterations):
    key = os.urandom(16)
    engines.append( AES.new( key, AES.MODE_ECB ) )
    data = engines[i].encrypt( data )

# Decipher in the oposite order

for i in range(iterations - 1, -1, -1):
    data = engines[i].decrypt( data )

recovered_index = int(str(data, 'utf8'))

if recovered_index == index:
    print( 'Success!' )
else:
    print( 'Failure, %d is different from %d' % (index, recovered_index) )
```

Figura 1.2: Cifra 1000 vezes de um número de ordem (`index`) com AES-128, subsequente decifra 1000 vezes, pela ordem inversa, e recuperação do valor inicial (12, no exemplo).

1.6 Troca de dicionários Python via *sockets* TCP

```python
import socket
import json
import base64

#
# Universal function to send a given amount of data to a TCP socket.
# It returns True or False, depending of the success on sending all
# the data to the socket.
#
def exact_send( dst, data ):
    try:
        while len(data) != 0:
            bytes_sent = dst.send( data )
            data = data[bytes_sent : ]
            return True
    except OSError:
        return False

#
# Universal function to receive a given amount of data from a TCP socket.
# It returns None or data, depending of the success on receiving all
# the required data from the socket.
#
def exact_recv( src, count ):
    data = bytearray( 0 )
    while count != 0:
        new_data = src.recv( count )

        if len(new_data) == 0:
            return None

        data += new_data
        count -= len(new_data)

    return data

#
# Universal function to send a dictionary message to a TCP socket.
# It actually transmits a JSON object, prefixed by its length (in
# network byte order).
# The JSON object is created from the dictionary.
# It returns True or False, depending of the success on sending the
# message to the socket.
#
def send_dict( dst, msg ):
    # DEBUG print( 'Send: %s' % (msg) )
    data = bytes(json.dumps( msg ), 'utf8')
    prefixed_data = len(data).to_bytes( 4, 'big' ) + data
```

```
    return exact_send( dst, prefixed_data )

#
# Universal function to receive a dictionary message from a TCP socket.
# It actually receives a JSON object, prefixed by its length (in network byte order).
# The dictionary is created from that JSON object.
#
def recv_dict( src ):
    prefix = exact_recv( src, 4 )

    if prefix == None:
        return None

    length = int.from_bytes( prefix, 'big' )
    data = exact_recv( src, length )

    if data == None:
        return None

    msg = json.loads( str(data, 'utf8') )
    # DEBUG print( 'Recv: %s' % (msg) )
    return msg

#
# Universal function to send and receive a dictionary to/from a TCP socket peer.
# It returns None upon an error.
#
def sendrecv_dict( peer, msg ):
    if send_dict( peer, msg ) == True:
        return recv_dict( peer )
    else:
        return None
```

1.7 Esqueleto da interação do servidor com clientes

```
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
s.bind( ... )
s.listen()

clients = []

while True:
    try:
        available = select.select( [ s ] + clients, [], [] )[0]
    except ValueError:
        # Sockets may have been closed, check for that
        for c in clients:
            if c.fileno() == -1: # closed
                clients.remove( c )
        continue # Reiterate select

    for c in available:
         # New client?
         if c is s:
            new, addr = s.accept()
            clients.append( new )

         # Or a client message/disconnect?
         else:
            # See if client sent a message
            if len(c.recv( 1, socket.MSG_PEEK )) != 0:
                # Handle the new message received in socket c

            # or just disconnected
            else:
                # You may need to perform some internal cleanup of your data structures here
                clients.remove( c )
                c.close()
                break # Reiterate select
```

Glossário

AES	Advanced Encryption Standard (cifra simétrica por blocos)
Base64	Base64 (forma de codificar octetos arbitrários como caracteres)
CSV	Comma Separated Values
DNS	Domain Name System
ECB	Electronic Code Book (modo de cifra por blocos elementar, sem realimentação)
IPv4	Internet Protocol v4
JSON	JavaScript Object Notation
SHA-256	Secure Hashing Algorithm (versão 2 com resultado de 256 bits)
TCP	Transmission Control Protocol (protocolo de transporte da Internet orientado à ligação)
UDP	User Datagram Protocol (protocolo de transporte da Internet sem ligação, orientado ao datagrama)