

Universidade do Minho

MESTRADO EM ENGENHARIA INFORMÁTICA

Engenharia de segurança

Ficha de exercício 6

Grupo 1

RUI CARLOS AZEVEDO CARVALHO - PG47633

DANIEL BARBOSA MIRANDA - PG47123

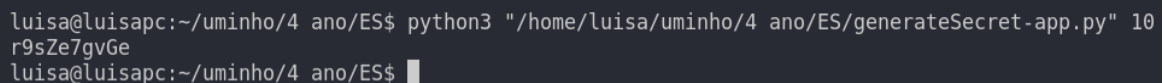
ANA LUÍSA LIRA TOMÉ CARNEIRO - PG46983

Parte IX: Criptografia aplicada

Pergunta P1.1

Exercício 1

O programa *generateSecret-app* tem como objetivo gerar uma string de valores aleatórios dando como *input* o tamanho dessa string. Tal como podemos ver pela imagem, a string gerada tem 10 caracteres de comprimento, sendo esses caracteres dígitos ou letras.



```
luisa@luisapc:~/uminho/4 ano/ES$ python3 "/home/luisa/uminho/4 ano/ES/generateSecret-app.py" 10
r9sZe7gvGe
luisa@luisapc:~/uminho/4 ano/ES$
```

Figure 1: String de valores aleatórios limitado a dígitos e a letras

De forma a gerar a string utiliza-se a função *generateSecret* do ficheiro *shamirsecret.py* que através da função *urandom* vai gerar os valores aleatório. Contudo, a função *generateSecret* só gera strings de dígitos e letras. Para isso a função implementa uma condição que verifica se o caractere correspondente ao valor gerado é um dígito ou letra. Caso o caractere não for um dígito ou uma letra então a função volta a gerar caracteres. Abaixo podemos ver a condição que verifica se o caractere gerado é um dígito (*string.digits*) ou uma letra (*string.ascii_letters*).

```
if (chr(c) in (string.ascii_letters + string.digits) and l < secretLength):
    l += 1
    secret += chr(c)
```

Exercício 2

Para este exercício é necessário que a string gerada de caracteres aleatórios não esteja limitada a dígitos e a letras. Para isso alterou-se o código da função *generateSecret* do ficheiro *shamirsecret.py* de forma a utilizar na condição mencionada no exercício 1 o método *printable* da biblioteca String em vez de utilizar os métodos *ascii_letters* e *digits*. Este método *printable* permite verificar se os caracteres são do tipo *printable*, isto é, se o caractere é um dígito, uma letra ou um sinal de pontuação, sendo este último conjunto composto pelos caracteres:

```
!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~
```

Assim conseguimos gerar string de caracteres aleatórios que não esteja limitada a caracteres do tipo dígito ou letra. Em abaixo encontra-se a implementação na íntegra da função *generateSecret* alterada de forma a responder ao problema.

```
def generateSecret(secretLength):
    l = 0
    secret = ""
    while (l < secretLength):
```

```

s = utils.generateRandomData(secretLength - 1)
for c in s:
    if (chr(c) in (string.printable) and l < secretLength):
        l += 1
        secret += chr(c)
return secret

```

Tal como podemos ver pela imagem seguinte, a string gerada tem 10 caracteres de comprimento, sendo esses caracteres dígitos, letras, sinais de pontuação entre outros.

```

luisa@luisapc:~/uminho/4 ano/ES$ python3 "/home/luisa/uminho/4 ano/ES/generateSecret-app.py" 10
Q*LY8>dBIr
luisa@luisapc:~/uminho/4 ano/ES$ █

```

Figure 2: String de valores aleatórios não limitado a dígitos e a letras

Pergunta P2.1

A primeira parte deste exercício consiste em gerar um par de chaves, onde a chave privada vai ser utilizada para assinar um objeto JWT, utilizando para tal o comando `openssl genrsa -aes128 -out mykey.pem 1024`, que irá criar um ficheiro PEM.

A *pass phrase* utilizada no exemplo foi **password**.

```

rhezzus@RhEzZuS:~/Desktop/ShamirSharing$ openssl genrsa -aes128 -out mykey.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
Enter pass phrase for mykey.pem:
Verifying - Enter pass phrase for mykey.pem:
rhezzus@RhEzZuS:~/Desktop/ShamirSharing$ cat mykey.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,AF6E18276A87DDB497F8F0294332816F

UX/PpW/Qwu9B01VR07+itP90+rLwfgXinffwEv3nPVI4n8AviHtvBBd05AkEjj1m
d3Yfx0hRb7oWp3yQzYber7LaYmTuetdB3k0frCGJPsrSQcjDopGHVEfK00QWYTqc
6yRPChbGW10/shH5dR2tq6MFF5h8FcR6PHX/D2RPTgXrLtX03bza94r7F1K88g0A
mTRA1yEhZkXy0+UjkQj9F0Sh8+IXWrgtFjwMEYpHzdRjXzjomL0o9T883pBrM72P
co3yAJH0fZtLzhWvGm1MX1nPh0VFmbPfvuX8Ep5jsLaCk1SN0sRJ0rq8MdKp/kaX
wpvekGCOpIuSt6WJFmz3FPtU7bQizufv9GUjATgcoM3FXCpafyrerdK7BRWvzPaL
nLDXlCe5je/xjGnVb9VbCH3JJTFGGnLXC0jroyeFWcBmFe7bVuGkkVrgjYA9ig2M
9K3I66HSVTYqEyuIVWnIx2sreaUmKGke2uLQ4+Wz3ncVdwb18mxsnNM4sxuB6N2Y
T65GydpdzBTvvH0QYx0TuvqKLhmCI/MGg/ya//PkPjUH7kE3VaSj60AKpWi8GQ1Y
1thFVA6oLE8SpAN/n3+Re2Ds9A+LxsBZ6T0PySwG2+16Y1SjI3N3a9/m86InuKAU
CxzWeBUfMxbjYcnCwAovRSHULocW5bGXtTbjEQeuaIU2vWMEUF/+sk7aESG3dWl
f90QwGgA/3/K0ZUC5CjRdFAU06n08gQZrwScqo9LdbAgXea9M/pNLRRZW9LaqNU
Zk5LrasFe6rvpM0hEaen7sjpxOJ1f7Iwt6LZ2j+owGNg/kwwUA8fSxbjC8BRVYkm
-----END RSA PRIVATE KEY-----
rhezzus@RhEzZuS:~/Desktop/ShamirSharing$ █

```

Figure 3: Geração do par de chaves

De seguida, como se observa na figura abaixo, gera-se um certificado para o ficheiro PEM, que mais tarde irá servir para reconstruir o segredo. Esse certificado é gerado com recurso ao comando `openssl req -key mykey.pem -new -x509 -days 365 -out mykey.crt`, onde é pedida a *pass phrase*, que é **password**, e de seguida, outros dados cujos valores foram deixados em *default*.

Figure 4: Geração do certificado do par de chaves

[illegible]

Figure 5: Divisão do segredo

```
python3 recoverSecretFromAllComponents-app.py 3 uid mykey.crt
python3 recoverSecretFromComponents-app.py 3 uid mykey.crt
```

Nestes 2 casos está-se a reconstruir o segredo com o valor do quorum e a diferença pode ser facilmente observada. No caso do primeiro programa são necessárias todas as partes em que o segredo foi dividido para o recuperar, sendo por isso que a reconstrução

falhou, enquanto o segundo basta o valor respetivo do quorum, em partes, tendo sido um sucesso.

[illegible]

Figure 6: Reconstrução do segredo com quorum

Estas 2 últimas figuras representam a reconstrução do segredo utilizando para tal o total de partes em que este havia sido dividido. Os comandos utilizados foram `python3 recoverSecretFromAllComponents-app.py 7 uid mykey.crt`, no caso do primeiro programa e `python3 recoverSecretFromComponents-app.py 7 uid mykey.crt`, no caso do segundo.

Como seria de se esperar o segredo foi reconstruído com sucesso.

[illegible]

Figure 7: Reconstrução do segredo com número de partes


```

6 //Criação do objeto correspondente a cifra AES no modo GCM
7 Cipher AES_128_Cipher = Cipher.getInstance("AES/CTR/NoPadding");
8 AES_128_Cipher.init(Cipher.ENCRYPT_MODE, sk, params);

```

De seguida, o conteúdo do ficheiro é lido e o resultado de aplicar a cifra mencionada ao seu conteúdo é guardado no ficheiro de output:

```

1 CipherInputStream in = new CipherInputStream(new
  ↳ FileInputStream(input), AES_128_Cipher);
2 int readBytes;
3
4 //Leitura para o buffer e escrita do texto cifrado para o ficheiro de
  ↳ output
5 while((readBytes = in.read(buffer)) != -1){
6     out.write(buffer, 0, readBytes);
7     out.flush();
8 }

```

Por fim, cria-se um objeto que permite realizar o HMAC já que se pretende implementar um *encrypt-then-MAC*. A função de hash utilizada é o SHA256 sendo aplicado ao conteúdo do ficheiro de output (ficheiro com o ciphertext) sendo o hash resultante guardado no fim do ficheiro (últimos 32 bytes).

```

1 Mac hmac = Mac.getInstance("HmacSHA256");
2 hmac.init(sk);
3
4 //Leitura para o buffer e escrita do texto cifrado para o ficheiro de
  ↳ output
5 while((readBytes = hmIN.read(buffer)) != -1){
6     hmac.update(buffer, 0, readBytes);
7 }
8
9 out.write(hmac.doFinal());
10 out.flush();

```

Por fim o IV é escrito num ficheiro à parte (indicado pelo utilizador) para que possa ser utilizado no processo em que se decifra a mensagem.

Já a funcionalidade de decifrar é implementada através do método "decipherHandler" que utiliza os mesmos objetos para a chave privada e cifra, mas inicializa este último no modo de decifragem. Para além disto, necessita de garantir que decifra todos os bytes do ficheiro que contém o cipher text exceto os últimos 32 que correspondem ao hash:

```

1 while((readBytes = in.read(buffer)) != -1){
2     if ((totalBytes + readBytes > fi.length() - 32) && (readBytes - 32
  ↳ > 0))
3         out.write(buffer, 0, readBytes - 32);
4     else
5         out.write(buffer, 0, readBytes);

```

```

6
7     out.flush();
8
9     totalBytes += readBytes;
10 }

```

Por fim a funcionalidade de verificar a autenticidade do ficheiro é implementada pelo método "verifyHandler" que realiza o processo de hash, tal como foi realizado no processo de cifra, sobre os bytes presentes no ficheiro que contém o ciphertext, ignorando os últimos 32 bytes, pois estes correspondem ao hash criado na funcionalidade de cifra:

```

1  hmac.init(sk);
2
3  BufferedInputStream hmIN = new BufferedInputStream(new
4      ↪ FileInputStream(input));
5  int totalBytesRead = 0;
6  while((readBytes = hmIN.read(buffer)) != -1){
7
8      if ((totalBytesRead + readBytes) > (file.length() - 32) &&
9          ↪ readBytes - 32 > 0){
10         hmac.update(buffer, 0, readBytes - 32);
11     }else{
12         hmac.update(buffer, 0, readBytes);
13     }
14     totalBytesRead += readBytes;
15 }
16 byte[] result = hmac.doFinal();

```

Antes deste processo é retirado o hash que está presente nos últimos 32 bytes do ficheiro:

```

1     fi.seek((file.length() - 32));
2     fi.readFully(fileHmac);

```

Por fim é feita a comparação entre o hash que foi criado neste método e aquele presente no ficheiro:

```

1     return Arrays.equals(result, fileHmac);

```

O código correspondente a esta questão pode ser acedido através deste [link](#).

Para executar a funcionalidade de gerar as chaves basta correr o seguinte comando:

```
java AE cifra [INPUT_FILE] [OUTPUT_FILE] [IV_OUTPUT_FILE] [KEY (16 bytes)]
```

- INPUT FILE: ficheiro que contém a mensagem que se pretende cifrar.
- OUTPUT FILE: ficheiro que irá guardar o output da operação de cifrar.
- IV INPUT FILE: ficheiro que irá guardar o IV utilizado para cifrar a mensagem.

- KEY: chave de 16 bytes que foi utilizada para cifrar o ficheiro.

Para executar a funcionalidade de assinar o ficheiro utiliza-se o seguinte comando:

```
java AE decifra [INPUT_FILE] [OUTPUT_FILE] [IV_INPUT_FILE] [KEY (16 bytes)]
```

- INPUT FILE: ficheiro que contém a mensagem que se pretende decifrar.
- OUTPUT FILE: ficheiro que irá guardar o output da operação de decifrar.
- IV INPUT FILE: ficheiro que contém o IV utilizado para cifrar a mensagem.
- KEY: chave de 16 bytes que foi utilizada para cifrar o ficheiro.

Por fim, para verificar uma assinatura basta utilizar o seguinte comando:

```
java AE verifica [INPUT_FILE] [KEY (16 bytes)]
```

- INPUT FILE: ficheiro que contém o resultado da operação de cifrar um ficheiro.
- KEY: chave de 16 bytes que foi utilizada para cifrar o ficheiro.

Pergunta P4.1

O resultado obtido da aplicação do comando "openssl x509 -in cert.crt -text -noout" pode ser verificado nas duas figuras que se seguem:

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
    a1:47:25:33:ba:f7:d0:0f:66:74:d3:e4:7e:f7:77:2e
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: C = ch, O = Swisscom, OU = Digital Certificate Services, CN = Swisscom Root CA 2
  Validity
    Not Before: May 18 08:44:46 2017 GMT
    Not After : May 18 08:44:46 2027 GMT
  Subject: organizationIdentifier = VATAT-U64741248, C = AT, O = Swisscom IT Services Finance S.E., OU = Digital Certificate Services, CN = Swisscom Diamant EU CA 4
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public-Key: (2048 bit)
    Modulus:
      00:ba:6c:e3:5c:fb:ce:72:9f:cf:b6:18:2c:e2:a1:
      cd:fa:d4:54:dd:a6:96:8b:e5:5d:71:1b:04:21:e7:
      14:fc:d0:92:af:23:93:8b:d3:0e:8a:c6:04:af:ae:
      59:4d:52:a2:3a:92:06:20:e8:8f:c0:fe:2f:60:f7:
      8f:04:9b:5d:8f:35:e5:37:68:ea:37:21:de:db:59:
      32:2d:2d:45:47:93:bd:61:ae:26:4c:76:85:29:a2:
      ee:bf:df:2b:21:16:bc:9f:6d:fd:70:87:e1:27:3a:
      b6:b0:da:90:a1:97:ff:ad:69:3e:97:7b:b6:a4:af:
      91:27:2a:26:1f:78:30:04:c6:85:e6:c0:16:b6:8e:
      a3:38:1f:e8:c8:74:17:e7:85:a8:b0:05:c3:76:4f:
      a3:fa:bb:ef:9e:dd:94:46:b8:9c:ca:19:2b:ad:82:
      7c:75:cc:db:df:c5:b7:0c:93:f7:15:9c:d0:0b:14:
      6c:e4:11:d4:1a:ea:40:e3:5f:81:da:b1:c3:88:89:
      69:ae:fd:27:c4:d6:54:c8:5e:f8:1f:31:46:54:de:
      77:65:0d:cf:5a:37:82:03:e0:9f:10:fb:7e:e8:0e:
      a9:c5:2e:72:6f:f2:6f:00:fa:8a:fc:78:65:4b:72:
      57:9d:46:ac:39:7f:21:86:88:7f:e4:f0:82:46:5c:
      9e:3b
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Authority Key Identifier:
      keyid:4D:26:20:22:89:4B:D3:D5:A4:0A:A1:6F:DE:E2:12:81:C5:F1:3C:2E

    Authority Information Access:
      CA Issuers - URI:http://aia.swissdigidcert.ch/sdcs-root2.crt

    X509v3 Basic Constraints: critical
      CA:TRUE, pathlen:0
    X509v3 Certificate Policies:
      Policy: 2.16.756.1.83.100.4.1
      CPS: http://www.swissdigidcert.ch/cps/
```

```

Authority Information Access:
  CA Issuers - URI:http://aia.swissdigicert.ch/sdcs-root2.crt

X509v3 Basic Constraints: critical
  CA:TRUE, pathlen:0
X509v3 Certificate Policies:
  Policy: 2.16.756.1.83.100.4.1
  CPS: http://www.swissdigicert.ch/cps/

X509v3 CRL Distribution Points:

  Full Name:
    URI:http://crl.swissdigicert.ch/sdcs-root2.crl

X509v3 Key Usage: critical
  Certificate Sign, CRL Sign
qcStatements:
  0.0.....F..0.....F..
X509v3 Subject Key Identifier:
  8B:01:D7:DE:C7:92:B2:E4:5B:24:9E:B6:8F:49:3A:AF:A9:C6:72:DD
Signature Algorithm: sha256WithRSAEncryption
  7d:7d:2f:7a:4c:13:88:f9:c5:82:bb:0a:38:8c:a1:7c:6d:e4:
  03:76:af:53:b8:21:87:34:5d:bb:6d:06:d4:17:99:b7:be:da:
  3c:62:e1:03:e3:8f:f0:a2:a2:c5:df:44:d2:a1:48:f9:00:75:
  3a:08:4b:52:71:17:62:5f:85:99:f7:ec:53:a7:cb:38:d8:66:
  13:44:a1:cc:a1:04:c3:4d:ba:f5:3e:ec:62:a2:dd:fa:e5:0b:
  28:bc:d4:2f:46:46:35:0d:d7:95:45:5c:75:eb:ae:9e:21:e9:
  ca:35:ac:08:d7:78:b7:89:ac:6c:3d:d9:e2:57:d5:af:9f:bc:
  0a:10:4f:8e:ef:b2:f5:f0:63:ab:e0:09:c8:17:e6:60:a3:5e:
  e3:dc:81:77:c5:db:bf:7f:a1:2e:23:87:6d:b5:aa:d2:99:2d:
  03:b6:21:15:97:3c:e0:8a:2a:73:58:9a:66:69:74:67:d9:79:
  c4:1f:52:6c:bc:3e:57:73:cc:6d:8d:15:8b:84:43:b5:f8:1f:
  e1:05:4c:1e:b9:2d:ea:b5:b5:74:9d:4a:4e:11:69:16:7f:c9:
  39:83:00:24:8e:b1:39:68:2e:39:d1:10:b4:6a:fb:d1:61:a5:
  ee:58:2e:a8:fd:74:a0:6a:88:8e:57:09:af:ec:2a:c0:73:75:
  ab:af:23:6b:a6:f6:74:5c:b8:54:e8:f6:7b:69:87:4b:1d:04:
  4a:e1:1d:7f:41:23:61:ae:f7:d8:23:91:d4:54:96:83:1c:8f:
  4e:d3:70:19:3f:66:b7:03:ca:59:f0:c2:41:6f:50:83:f3:07:
  81:e9:03:d8:ad:67:81:e8:65:48:92:cc:ac:16:d6:98:09:1f:
  bc:b7:c2:46:0c:16:b7:57:44:65:76:81:98:9f:72:62:08:1b:
  bf:3c:a1:1e:29:22:61:ad:bf:5f:24:9b:c7:da:5f:2c:c8:7a:
  46:23:60:c7:a9:65:73:bd:6b:0f:90:4d:3d:50:b3:53:23:9f:
  b2:68:df:13:7d:8d:aa:2e:e3:0f:46:f9:c4:c0:c8:c8:00:d6:
  30:c7:d5:04:c0:7c:bb:9c:a7:50:a6:07:57:4b:ef:f1:02:db:
  5c:51:8e:05:59:6b:f0:69:b8:c2:60:96:43:e3:6b:6d:09:09:
  42:ea:15:97:76:c3:2f:49:46:99:c2:5f:c4:16:21:99:75:9b:
  0d:41:18:f9:30:2b:e4:59:37:c3:ca:6d:13:2c:9e:d8:0c:5b:
  78:d4:0c:d7:0f:29:67:9d:63:56:09:91:3e:b6:77:54:86:29:
  60:cb:14:bc:b2:f4:9b:b0:cc:ab:fe:8d:d5:f4:6e:93:59:fd:
  26:40:6f:97:81:80:78:0b

```

Através das figuras acima é possível verificar que o algoritmo de chave pública utilizado é o RSA cujo tamanho de chave é de 2048 bits e o expoente igual a 65537. Ora, apesar do valor do expoente ser adequando $\log_2 65537 > 16$ o valor do tamanho da chave (tamanho do modulus RSA) deveria ser maior ou igual a 3000, sendo inclusive o valor 2048 considerado *legacy* cujo prazo é até 2025.[1]

Pergunta P5.1

Este problema consistia na alteração dos ficheiros presentes na pasta [aula7/BlindSignature](#) de forma a simplificar o input e output das funções que implementam o método de assinatura cega (*Blind signature*) baseada no *Elliptic Curve Discrete Logarithm Problem* (ECDLP).

A assinatura cega possibilita que uma entidade peça a uma terceira entidade para assinar digitalmente uma mensagem, sem lhe revelar o conteúdo. Para isso, é necessário 3 participantes: um requerente, isto é, a entidade que vai enviar a mensagem para ser assinada; um assinante que vai assinar a mensagem sem saber o seu conteúdo e por fim um verificador que vai validar a autenticidade da assinatura. Cada um destes participantes é responsável por etapas desta processo, sendo o assinante encarregue da fase de inicialização e assinatura, o requerente responsável pelas fases de ofuscação e desofuscação e o verificador está encarregue da fase de verificação.

Como forma de implementar este método começamos com a inicialização de todos os componentes através da fase de inicialização. De seguida, o requerente vai ocultar a mensagem a ser enviada utilizando os componentes gerados na inicialização. Esta mensagem ofuscada vai ser enviada ao assinante para que este assine utilizando a sua chave privada e devolvendo desta forma a assinatura ofuscada do próprio. É possível, na fase de desofuscação, o requerente obter a assinatura original do assinante utilizando componentes da fase inicial e a assinatura ofuscada. Finalmente, utilizando a chave pública, o verificador consegue verificar se a assinatura obtida na fase de desofuscação é válida e por isso autêntica.

- **Inicialização:** Fase implementada no ficheiro **init-app.py** onde se vai gerar todas as componentes a serem utilizadas para o processo de assinatura. É no ficheiro `init-app.py` que se implementou a função *generateKeys* onde se gera a chave privada e pública utilizando curvas elípticas. Estas chaves serão armazenadas em ficheiros de formato pem para depois serem utilizadas na fase de assinatura e verificação. É de notar que no caso do armazenamento da chave privada será necessário ao assinante introduzir uma *password* para que esta chave esteja protegida de entidades externas. Nesta fase são ainda geradas componentes, *InitComponent* e *pRDashComponents* que serão utilizadas na fase de ofuscação e desofuscação. Estas componentes vão ser armazenadas num ficheiro de nome *AssinateFile.txt*. No caso da abordagem `init-app.py` só a componente *InitComponent* é que será a armazenada neste ficheiro, já na abordagem `init-app.py -init` ambas as componentes serão armazenadas no ficheiro.
- **Ofuscação:** Etapa implementada no ficheiro **ofusca-app.py** onde se vai ocultar a mensagem original a ser enviada para o assinante, devolvendo desta forma a mensagem ofuscadas (*blind message*). É na função *showResults* deste ficheiro que vamos gerar as componentes *BlindComponents* e *pRComponents* que serão utilizadas na fase de desofuscação e na fase de verificação. Estas componentes vão ficar armazenadas no ficheiro *RequerenteFile.txt*.
- **Assinatura:** Esta etapa foi implementada no ficheiro **blindSignature-app.py** e corresponde à fase onde o assinante irá receber a mensagem ofuscada e o ficheiro com a sua chave privada para assinar a mensagem devolvendo, no final, a sua assinatura ofuscada (*blind signature*). Para além destes parâmetros será

necessário a componente *InitComponents* que será obtida a partir do ficheiro *AssinateFile.txt*.

- **Desofuscação:** Fase implementada no ficheiro **desofusca-app.py** onde a partir da assinatura ofuscada e da componente *pRDashComponents*, conseguimos obter a assinatura original do assinante (*signature*). Para além destes parâmetros será necessário a componente *BlindComponent* que será obtida a partir do ficheiro *RequerenteFile.txt*.
- **Verificação:** Etapa final implementada no ficheiro **verify-app.py** onde através da chave pública do assinante (certificado da assinatura), da mensagem e assinatura original e do ficheiro *RequerenteFile.txt* conseguimos verificar se a assinatura original sobre a mensagem recebida como parâmetros é ou não válida.

Finalmente, a assinatura cega desenvolvida em Python encontra-se implementada na íntegra no seguinte [link](#) do repositório do Github. Nesta pasta encontra-se o código alterado segundo as regras apresentadas no enunciado do problema e dividido segundo os vários participantes deste método.

Pergunta P6.1

1

O modo mais indicado para guardar o tipo de ficheiro em questão, num ambiente *cloud* seria utilizando criptografia homomórfica, dado que serão guardados dados privadas e não se deseja a exposição dos mesmos. Neste tipo de criptografia são enviados dados cifrados para a *cloud*, sendo que na mesma podem ser realizadas operações sobre os dados, sem a necessidade de decifrar os mesmos.

2

Tal como havia sido mencionado na questão anterior, não há necessidade de decifrar os dados, portanto pode-se efetuar diretamente e de forma normal, dentro da *cloud*, as operações em questão sobre os mesmos, sendo que o seu resultado continuarão a ser dados cifrados, e portanto não compromete os mesmos.

3

Guardar ficheiros O nome do programa criado para guardar dado localmente é `store.py`.

```
import json
from phe import paillier
import re
```

A biblioteca `json` servirá para a serialização dos valores de cada tuplo, a biblioteca `paillier`, biblioteca `phe` em para utilizar a criptografia homomórfica de Paillier (Partially homomorphic encryption) e a biblioteca `re` para efetuar a interpretação inicial do ficheiro com os dados que serão testados.

```
def store(pub_key, filename, saveFile):

    # abertura do ficheiro que contem os dados
    testfile = open(filename, "r")

    # dicionario onde serão armazenados os dados
    dictionary = {}
```

Assim sendo o programa `store` irá necessitar da chave pública, para poder cifrar os valores dos tuplos, o nome do ficheiro com os dados iniciais que se pretende armazenar e por fim o nome do ficheiro onde os dados serão armazenados.

Assim começa-se por abrir o ficheiro com os dados iniciais e criar um dicionário onde serão armazenados esses dados.

```
for linha in testfile:

    # obter o NIC de cada cliente
    nic = re.search(r'([A-Z0-9]+)', linha)
    nic = nic.group(0)

    # obter os tuplos de cada cliente
    valores = re.findall(r'([A-Z0-9]+\)\s([0-9\,]+)', linha)

    # adição do cliente ao dicionário caso ainda não esteja
    if nic not in dictionary.keys():
        dictionary[nic] = {}
```

Nesta etapa, a partir de cada linha do ficheiro inicial, adquirimos o NIC de cada cliente assim como os seus tuplos e caso o seu NIC não esteja presente no dicionário adiciona-se uma entrada do mesmo.

```
# percorre cada tuplo
for pair in valores:

    # tipo de analise
    tipo = pair[0]

    # valor e sua cifra
    valor = float(pair[1].replace(",","."))
    valor = pub_key.encrypt(valor)

    # prepara o valor para poder ser guardado no ficheiro
    valor_para_serializar = (str(valor.ciphertext()),
        ↪ valor.exponent)

    # se o tipo de analise ainda nao estiver no dicionario
    # adiciona-se o mesmo e o valor em questão a lista
    if tipo not in dictionary[nic].keys():
```

```

        dictionary[nic][tipo] = [valor_para_serializar]

        # se estiver apenas adiciona-se o valor ao fim da lista
        else: dictionary[nic][tipo].append(valor_para_serializar)

testfile.close()

```

Cada tuplo é percorrido e cifra-se a parte correspondente ao valor com a criptografia de Pallier e, de seguida, esse valor é preparado para mais tarde poder ser serializado.

Por fim resta adicionar o mesmo ao dicionário sendo que para tal verifica-se se, para o cliente em questão, o tipo da análise está no seu dicionário e, caso não esteja, cria-se uma entrada para o mesmo, com uma lista que conterá o valor em questão. Se o tipo já está no dicionário significa que já havia valores na sua lista antes e portanto acrescenta-se o valor ao fim da lista.

```

# usado para deserialization dos valores
dictionary['public_key'] = { 'g':pub_key.g, 'n':pub_key.n}

# guarda-se o dicionário num ficheiro
savefile = open(saveFile, "w")
savefile.write(json.dumps(dictionary))
savefile.close()

```

Nesta fase é necessário criar uma entrada, com elementos da chave pública que servirão para mais tarde reconstruir os valores serializados.

Agora, com o dicionário com todos os dados necessários e os valores preparados para serialização, cria-se um ficheiro, ou altera-se um já existente, onde os dados serão serializados e escritos no mesmo, terminando assim o processo de armazenamento de dados.

Calcular Média O nome do programa criado para calcular a média dos dados armazenados localmente, sem os decifrar é `average.py`.

```

from phe import paillier
import ast

```

Na implementação deste programa é mais uma vez utilizada a biblioteca `phe`, com a criptografia de Pallier e a biblioteca `ast` para interpretar os dados contidos no ficheiro como um dicionário, para se poder trabalhar sobre o mesmo.

```

def average(tipo, filename):

    # abertura do ficheiro com os dados
    savefile = open(filename, "r")
    dictionary = savefile.read()

    # recuperação do dicionário
    dictionary = ast.literal_eval(dictionary)

```



```

# será utilizada na reconstrução do valor
public_key_rec =
    ↪ paillier.PaillierPublicKey(n=int(dictionary['public_key']['n']))

soma = 0.0
average = 0.0
count = 0

```

Assim começamos por abrir o ficheiro com os dados armazenados e recuperar o dicionário para se poder realizar operações sobre os seus valores. De seguida, através do dicionário preparamos, com os parâmetros da chave pública nele contidos, uma variável que irá permitir a reconstrução dos valores serializados.

Também são inicializados os acumuladores e contadores que serão utilizados para determinar o resultado pretendido, a média.

```

# percorre cada cliente e cada tipo e para cada elemento contido
↪ nessa
# combinação adiciona ao acumulador soma e incrementa o contador
↪ count
for nic in dictionary.keys():
    if tipo in dictionary[nic].keys():
        for valor in dictionary[nic][tipo]:

            # reconstrução do valor
            valor = paillier.EncryptedNumber(public_key_rec,
            ↪ int(valor[0]), int(valor[1]))
            soma = soma + valor
            count = count + 1

average = soma / count

return average

```

Com o dicionário reconstruído e com a possibilidade de recuperar valores, resta percorrer o dicionário, por cada NIC de cada cliente e verificar se o mesmo possui valores do tipo pretendido. Se possuir então esse valor será recuperado e adicionado ao acumulador `soma`, além de ocorrer uma incrementação no contador de elementos, `count`.

Para terminar, após finalizar o processo mencionado simplesmente divide-se o acumulador `soma` pelo contador `count` e está o valor da média adquirido, ainda no estado cifrado, dado que em nenhum momento se decifrou algum valor.

Testes De modo a facilitar a fase de testes do problema foi criado um programa, `TestagemEx6.py` onde simplesmente são geradas as chaves pública e privada de Paillier e, de seguida são pedidos os argumentos para os quais se deseja testar os programas, sendo que os mesmos são chamados.

```

Pratica 1 > TP6 > Problema P6.1 > ≡ ficheiroTeste.txt
1 123456789, (A23, 12,2), (B4, 32,1), (A2, 102), (CAA2, 34,5)
2 012345678, (B1, 2,2), (B3, 321,0), (C2, 10,2), (CAA2, 21,0)
3 234567890, (B3, 32,5), (A82, 3,21), (A2, 102,0), (CAA2, 12,2)
4 102030405, (C2, 92,9), (B4, 27,0), (CAA2, 2,10), (A23, 85,5)
5 070809010, (A23, 93,0), (B4, 20,0), (A2, 12,1), (CAA2, 11,7)

```

Figure 9: Ficheiro com os dados iniciais

A imagem acima corresponde a um ficheiro acima possui um pequeno conjunto de exemplos com a formatação do enunciado, que serão utilizados nestes testes.

```

rhezzus@RheZZuS:~/Desktop/Grupo1/Pratica 1/TP6/Problema P6.1$ python3 TestagemEx6.py
Nome do ficheiro com os dados a serem guardados: ficheiroTeste.txt
Nome do ficheiro onde os dados serão guardados: SavedData.json
A guardar dados...
Dados guardados com sucesso!

Nome do ficheiro de análise: SavedData.json
Tipo da análise cuja média deseja calcular: A23
Média: 63.56666666666666
rhezzus@RheZZuS:~/Desktop/Grupo1/Pratica 1/TP6/Problema P6.1$

```

Figure 10: Teste

Na figura acima podemos observar um exemplo em que chamamos o programa de testes, com o comando `python3 TestagemEx6.py`. É pedido o programa que contém os dados iniciais que se pretende guardar, seguido pelo nome do ficheiro onde se pretende guardar os dados. Após estes serem guardados é necessário fornecer o seu ficheiro ao programa que calcula a média, bem como o tipo de análise cuja média se deseja calcular sendo, por fim, imprimido o resultado correspondente a esse valor.

```

{"123456789": {"A23": [
["1484208222126735758778849148860958532387427519843879202137734527416424996409157400155867160258755692
467832895180333689653104229030985991664312267694201154455214926244994863253267773679023701408061141459
916405965458131265875172792505572650324663840727176830648066125855267508527987003743636261156832796554
345030270123696253348957559229818885533521767693328124562868188873232746273080585334074635044564868369
997003227594846037570665709150185893258218824447676070143865077972824455680026098127448567311063759111
979007785401359723149788403666713370633794421169812114444291120609071797313513802853257991286116481917
279720710423347190799202004162184996035285375222427536950220672401686481774991041930321746700265439572
58017186132143626110461343077004633326213195138608025308940999085962575255546924254876307365700383243
989926502566729065409774207957135310248642157631849123070693536337169874880614546636389412939270221583
613639948321824177256251561998451545902971601810881858089890224708123830719102525150263792649662165447
393868262122430035526310133635712442612602914845321575136545246652764855908017317436293121406199740420
123770795340042131431500372516039095084494506354057804160938365199880854062985248967826443378696416139
225914865978450443300929931550636642535462561628939371607651953342652984208736231373394883300752621309
706914795339642165989267701565590824402008376956761169186738180264151927489859813046662117207650339244
625914636429676047886921755075754627335538462959337819031333650706107280027770651890573565838183449754
910446278477573839364386393581650146460782880076334887337626730918612602155159426504312719305413723882
140376157796861879501394138736070497015186390144595603636931703071162123969498778663411510038386941242
102491674977569432029452115958282358928036997896337051944550015722846870405454056485205090750581993872
7661147428944165", -13]], "B4": [
["1690555219974050662585806889062615512124186922130713963283280226164194810611370770208298036895177807
166118656544863101782749887129232661587003865501151395068993686137141393392371101233532300468838998485
295628925568336498285287434228078389710418316814663683375816179467407852350472846749598949704577380065
363770957443771938707831878954108319779444300323785310304572506243329169690247995451230071834509532421
611161284746623001382200596597819766102628697288163200242482890135061368430141544203518348502203714029
517615967874399266082628019686991811401907617624711878459730162086791108602893592500896048920243079046

```

Figure 11: Ficheiro de dados guardados

Nesta figura está demonstrado uma pequena parte do ficheiro gerado na fase de armazenamento, em que os valores cifrados estão serializados.

```

1  123456789, (A23, 12,2), (B4, 32,1), (A2, 102), (CAA2, 34,5)
2  012345678, (B1, 2,2), (B3, 321,0), (C2, 10,2), (CAA2, 21,0)
3  234567890, (B3, 32,5), (A82, 3,21), (A2, 102,0), (CAA2, 12,2)
4  102030405, (C2, 92,9), (B4, 27,0), (CAA2, 2,10), (A23, 85,5)
5  070809010, (A23, 93,0), (B4, 20,0), (A2, 12,1), (CAA2, 11,7)

```

Figure 12: Análise do tipo A23

No exemplo, o tipo de dado escolhido para calcular a média foi A23, e podemos observar que os 3 valores para este tipo, contidos no ficheiro inicial, foram 12.2, 85.5 e 93.0.

```

🔄 (12.2 + 85.5 + 93.0) / 3 =
63.5666666667

```

Figure 13: Média

Assim sendo, podemos fazer o cálculo da média destes valores em qualquer calculadora e veremos que o resultado é igual ao exibido na primeira figura deste exemplo, o que demonstra que, de facto, os programas estão a funcionar corretamente.

Bibliography

- [1] "SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms", SOG-IS Crypto Working Group [Janeiro 2020]. Pode ser acedido em: https://www.sogis.eu/uk/supporting_doc_en.html