## Pergunta 1.1

/dev/random e /dev/urandom são dois arquivos especiais, que servem de geradores de números aleatórios/pseudoaleatórios.

O /dev/random está continuamente a apanhar ruído eletrônico do dispositivo, ou seja, caso se queira criar um número relativamente alto de bits, dependendo da atividade do computador, pode demorar a criar esses mesmos bits (porque depende da quantidade de ruído produzido pelo dispositivo, e o /dev/random utiliza apenas bits criados a partir do momento em que é invocado). tende a ser “mais” aleatório que o /dev/urandom.

No entanto, como o /dev/random mantêm uma parte dos bits que vão sendo gerados numa *pool* interna, o /dev/urandom utiliza esses bits para gerar novos bits aleatórios, ou seja, ainda que não tenha a aleatoriedade do /dev/random, o /dev/urandom cria, com uma qualidade bastante alta, números aleatórios, sendo este o utilizado para criar *seeds* para a maioria do *software* em criptografia.

## Pergunta 1.2

Como o ruído electrónico produzido por um dispositivo é relativamente pouco, o arquivo /dev/random demora um tempo muito significativo para apresentar o tamanho de bits desejado. Ao instalar um daemon de entropia adaptado do algoritmo HAVEGE que produz ruído electrónico, o tempo que o arquivo /dev/random demora a produzir os bits pseudoaleatórios (embora que, agora os bits têm menos aleatoriedade, porque derivam do daemon haveged e não de lixo aleatório do dispositivo) é quase inexistente.

O arquivo /dev/urandom continua a derivar os seus bits pseudoaleatórios da pool produzida pelo arquivo /dev/random, e como o tempo que este demorava antes do daemon já era quase inexistente, apenas irá, por um lado, aumentar a aleatoriedade, porque a pool de bits do arquivo /dev/random é renovada mais rápidamente, por outro lado, diminui a aleatoriedade por derivar de bits produzidos pelo daemon.

Como conclusão, a segurança dos bits aleatórios produzidos por ambos os arquivos, irão depender (em parte, porque ambos os arquivos utilizam esses bits como semente, gerando novos bits a partir desses) dos bits gerados pelo daemon.

## Pergunta 1.3

O motivo do método shamirsecret do pacote eVotUM.Cripto apenas apresentar letras e dígitos (não contendo por exemplo caracteres de pontuação ou outros), devido à forma como o método está implementado na API do pacote.

```
        #Cripto-4.4.0
        def generateSecret(secretLength):
            """
                This function generates a random string with secretLength characters (ascii_letters and digits).
            Args:
                secretLength (int): number of characters of the string
            Returns:
                Random string with secretLength characters (ascii_letters and digits)
            """
            l = 0
            secret = ""
            while (l < secretLength):
                s = utils.generateRandomData(secretLength - l)
                for c in s:
                    if (c in (string.ascii_letters + string.digits) and l < secretLength): # printable character
                        l += 1
                        secret += c
            return secret
```

Devido ao pedaço de código:

```
        for c in s:
                    if (c in (string.ascii_letters + string.digits) and l < secretLength): # printable character
                        l += 1
                        secret += c
```

apenas letras e dígitos passam na condição, sendo estes os únicos imprimidos. Qualquer outro caracter é descartado.

# 2. Partilha/Divisão de segredo (Secret Sharing / Splitting)

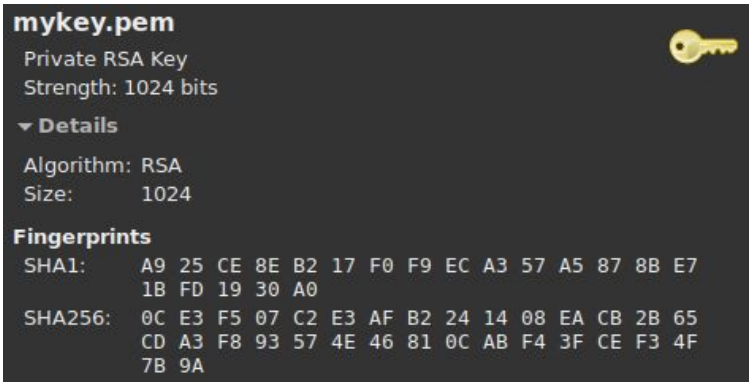
## Pergunta P2.1

**A)**

Para que se consiga dividir o segredo “Agora temos um segredo muito confidencial” em 7 partes distintas e com um quórum de 3, será necessário numa fase inicial, gerar uma chave privada. Para tal utilizou-se a ferramenta openssl. Executou-se o seguinte comando:

**openssl genrsa -aes128 -out mykey.pem 1024**

Este comando necessita de uma palavra-chave que será utilizada para cifrar os dados, posteriormente gera um ficheiro *myKey.pem* contendo a chave privada gerada para o utilizador, neste caso obteve-se o seguinte output:

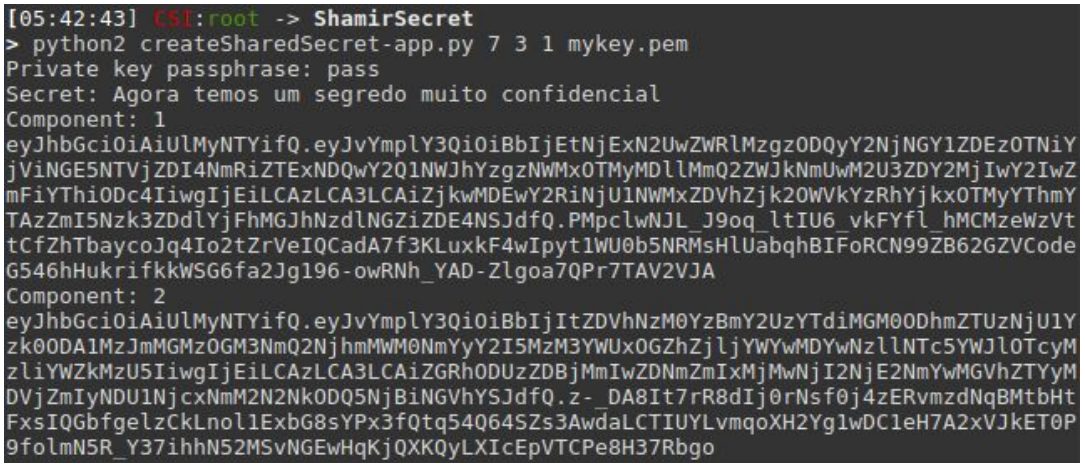



Posterior a criar o certificado, pode-se agora avançar para a divisão do segredo em 7 partes distintas, garantindo ainda que serão necessários apenas 3 elementos para que se consiga reconstruir este mesmo. Para realizar esta divisão, deverá-se utilizar o comando seguinte:

**python2 createShareSecret-app.py 7 3 1 mykey.pem**

Caso não sejam fornecidos todos os parâmetros necessários para a execução desta aplicação, é fornecido uma informação adicional acerca de todos os parâmetros necessários para entrada. Após execução do comando, é pedida a introdução da palavra-chave associada, juntamente com o segredo que se pretende partilhar. Após processamento dos dados de *input*, geram-se 7 componentes que

deverão ser divididas por elementos diferentes. Este *output* pode ser parcialmente observado na figura seguinte.

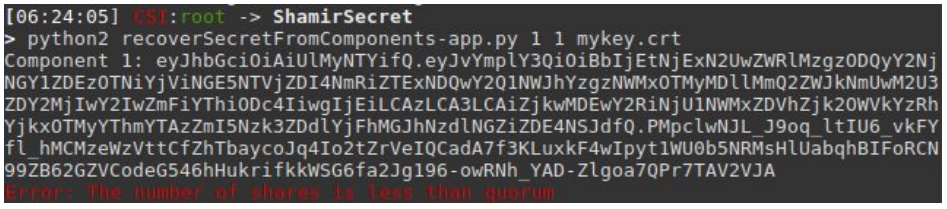


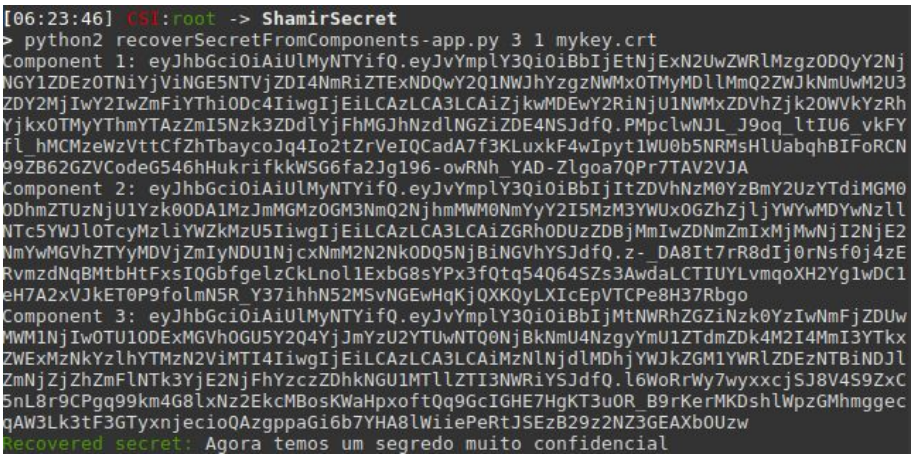

**B)**

Quando se pretende recuperar um segredo partilhado poder-se-á recorrer às partes criadas anteriormente. Para isto, recorreu-se às 2 aplicações distintas, *recoverSecretFromComponents-app* e *recoverSecretFromAllComponents-app*, visto ambas possibilitarem a decifragem do segredo.

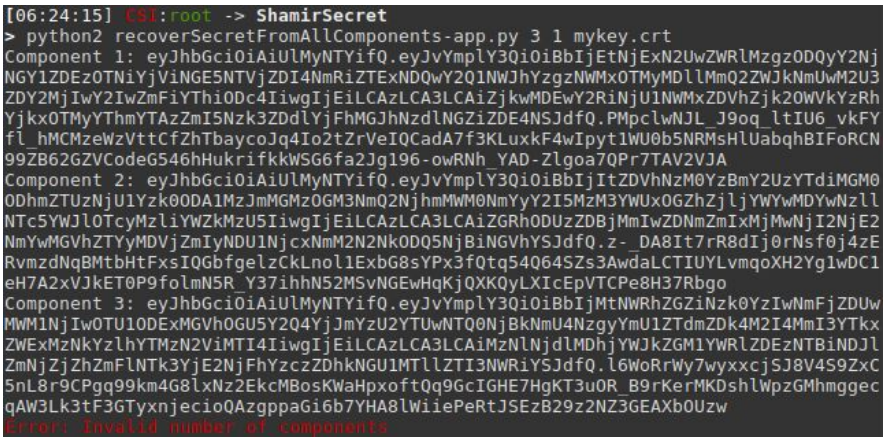
As diferenças existentes entre estas têm a ver com o facto de a primeira fornecer a possibilidade de decifragem através de 3 (no caso em que o quorum é igual a 3) ou mais elementos distintos, enquanto a segunda apenas consegue recuperar o segredo quando são inseridas todas as componentes deste.

Nas figuras seguintes demonstram-se os exemplo da primeira aplicação, onde foi testada a decifragem do segredo com 1 e com 3 elementos, respetivamente.






Demonstra-se agora o exemplo de utilização da segunda aplicação, onde apenas é possível decifrar o segredo quando todos os componentes estão presentes. As figuras apresentam os testes realizados para um *input* de 3 shares (testado na *app* anterior, e sendo o valor de *quorum*) e com a totalidade dos elementos (7 neste caso).





Poderá provar-se importante, a utilização da segunda aplicação (recuperar através de todas as fontes) quando se tratam de segredos importantes, onde parte dos intervenientes não deverão ter o poder de tomar decisões relativamente ao conteúdo destas.

## Pergunta 3.1:

Para utilizar os serviços providenciados pela empresa, um utilizador necessita de pagar a anuidade do serviço, garantindo que todos os seus certificados válidos, tendo estes uma data de validade (de um ano), serão guardados na base de dados da empresa.

O cliente envia ao serviço oferecido pela empresa, uma mensagem, a qual pretende obter cifrada, esta será cifrada utilizando a chave do utilizador criada para aquele dia. Envia também uma etiqueta que será associada à cifra, sendo que esta não se encontra cifrada.

Após a cifragem por parte do serviço, o cliente receberá a mensagem cifrada, juntamente com a indicação do dia em que esta foi cifrada.

A etiqueta deverá ter 32 bytes.

```
cifra(segredo_plaintext):
    key = getAno()+'.'+getMes()+'.'+getDia()
    msgCifrada = encrypt(key, segredo_plaintext)
    mac = hmac(key, msgCifrada)
    etiqueta = getEtiqueta()
    se(length(etiqueta)!=256 bits)
        print("Etiqueta deverá ter 32 bytes de tamanho (32 caracteres).")
        return -1
    return msgCifrada+mac+etiqueta
```

A cifra utilizada deverá ser AES-256, visto que tem boa compatibilidade com o HMAC-SHA256 (segundo as recomendações da NIST de 2016).

É utilizado a "Authenticated Encryption" "Encrypt-then-MAC", de modo a providenciar integridade e autenticidade à confidencialidade providenciada pela cifra.

O cliente, devido à etiqueta da cifra, poderá perceber o conteúdo da mensagem cifrada, sem necessitar que a decifre. Note-se que esta etiqueta poderá ser vista por qualquer utilizador.

Quando o cliente pretender decifrar uma mensagem previamente cifrada pelo serviço, terá de enviar a mensagem cifrada, juntamente com a data em que foi cifrada (ano, mês e dia).

```
decifra (segredo_cyphertext, chave_cifra):
    tamanho = length(segredo_cyphertext)
    cypher = segredo_cyphertext[0:(tamanho-512)]                    #cyphertext
    mac = segredo_cyphertext[(tamanho-512):(tamanho-256)]           #HMAC
    se(certificado fora da validade):
        print("Renove a anuidade do serviço)
        return -1;
    se(hmac(chave_cifra, cypher)!=mac)
        print("Chave errada, ou mensagem cifrada inválida");
        return -1
    msg = decrypt(chave_cifra, cypher)
    return msg
```

# Pergunta P4.1

- Itália, para as ECs "Banca d'Italia", "Ministero della Difesa", "Intesi Group S.p.A.";

## Banca d'Italia:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1203955915 (0x47c2e8cb)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C = IT, O = Banca d'Italia/00950501007, OU = Servizi di certificazione, CN = Banca d'Italia
        Validity
            Not Before: Feb 25 15:42:03 2008 GMT
            Not After : Feb 25 16:12:03 2018 GMT
        Subject: C = IT, O = Banca d'Italia/00950501007, OU = Servizi di certificazione, CN = Banca d'Italia
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:d4:2c:25:be:1c:8f:43:12:51:14:12:c7:d2:80:
                    1b:43:9d:57:68:6c:51:52:11:f0:0c:48:05:7d:46:
                    e7:72:97:d4:51:ca:31:6f:dc:34:03:e8:6e:75:39:
                    62:65:0a:44:c3:89:b8:6a:0e:db:1b:12:49:54:37:
                    a1:38:8e:3a:b3:38:14:3c:48:d6:5b:a0:b9:e8:48:
                    a2:da:b8:3c:34:ae:c7:48:47:9c:51:c9:d7:0e:55:
                    40:10:6d:41:8e:a0:d9:b5:42:5f:22:80:a4:97:28:
                    62:24:ca:65:75:3c:12:09:c6:fe:b1:89:91:3d:9c:
                    cf:ac:ee:21:c5:4c:bc:29:df:5d:cc:dc:23:b7:b9:
                    a7:d7:94:85:e3:2e:cd:34:af:bb:e2:c3:64:c9:2d:
                    89:8c:d8:f7:4d:90:bc:b6:e8:1b:0a:3a:79:34:3b:
                    a4:9b:d7:2c:f0:32:d4:9c:54:39:00:ed:21:e9:d0:
                    99:65:01:7f:16:4a:a4:0d:5c:c5:74:1f:c9:a9:f3:
                    ca:c8:46:eb:e3:7a:c5:9a:a3:77:2a:bb:7a:00:64:
                    6a:02:04:95:9c:a1:09:7d:8d:66:4d:3e:ee:59:56:
                    39:de:7d:3d:a6:2f:e1:1f:e4:0c:06:7c:2d:65:68:
                    f7:3a:2c:e5:60:9c:45:5d:7b:8a:e5:00:43:70:92:
                    f4:63
                Exponent: 65537 (0x10001)
```

Algoritmo de assinatura: sha1WithRSAEncryption
Algoritmo de chave pública: rsaEncryption
Tamanho da chave pública: 2048bit

Este certificado está dentro da validade e apresenta tamanho da chave e algoritmo de acordo com as recomendações do NIST de 2016 pelo que é adequado.

## Ministerio della Difesa:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 6936921824674324395 (0x6044e5a56a5e1fab)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IT, O = Ministero della Difesa, OU = S.M.D. - C.do C4 Difesa, serialNumber = 97355240587, CN = Ministero della Difesa - CA di Firma Digitale
        Validity
            Not Before: Jul 15 08:11:41 2014 GMT
            Not After : Jul 15 08:11:41 2044 GMT
        Subject: C = IT, O = Ministero della Difesa, OU = S.M.D. - C.do C4 Difesa, serialNumber = 97355240587, CN = Ministero della Difesa - CA di Firma Digitale
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
```

Algoritmo de assinatura: sha256WithRSAEncryption
Algoritmo de chave pública: rsaEncryption
Tamanho da chave 4096bit

Este certificado está dentro da validade e apresenta tamanho da chave e algoritmo de acordo com as recomendações do NIST de 2016 pelo que é adequado.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            e8:2e:3d:a6:c2:2b:f1:3c
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C = IT, O = Ministero della Difesa, OU = S.M.D. - C.do C4 Difesa, serialNumber = 97355240587, CN = Ministero della Difesa - PKI di Firma Qualificata, em
ailAddress = info_pkiff@smd.difesa.it
        Validity
            Not Before: Nov  9 10:48:40 2006 GMT
            Not After : Nov  8 10:48:40 2021 GMT
        Subject: C = IT, O = Ministero della Difesa, OU = S.M.D. - C.do C4 Difesa, serialNumber = 97355240587, CN = Ministero della Difesa - PKI di Firma Qualificata, e
mailAddress = info_pkiff@smd.difesa.it
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
```

Algoritmo de assinatura: sha1WithRSAEncryption
Algoritmo de chave pública: rsaEncryption
Tamanho da chave 2048bit

Este certificado está dentro da validade e apresenta tamanho da chave e algoritmo de acordo com as recomendações do NIST de 2016 pelo que é adequado.

# Intesi Group S.p.A.:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1115558334025201052 (0xf7b4264f1cc759c)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = IT, 2.5.4.97 = VATIT-02780480964, O = Intesi Group S.p.A., OU = Qualified Trust Service Provider, CN = Intesi Group EU Qualified Electronic Signatur
e CA G2
        Validity
            Not Before: Nov 23 10:10:33 2017 GMT
            Not After : Nov 18 10:10:33 2037 GMT
        Subject: C = IT, 2.5.4.97 = VATIT-02780480964, O = Intesi Group S.p.A., OU = Qualified Trust Service Provider, CN = Intesi Group EU Qualified Electronic Signatu
re CA G2
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
```

Algoritmo de assinatura: sha256WithRSAEncryption
Algoritmo de chave pública: rsaEncryption
Tamanho da chave 4096bit

Este certificado está dentro da validade e apresenta tamanho da chave e algoritmo de acordo com as recomendações do NIST de 2016 pelo que é adequado.

# Anexo .:

- NIST Recommendation, 2016

TDEA (Triple Data Encryption Algorithm) and AES are specified in [10].
Hash (A): Digital signatures and hash-only applications.
Hash (B): HMAC, Key Derivation Functions and Random Number Generation.
The security strength for key derivation assumes that the shared secret contains sufficient entropy to support the desired security strength. Same remark applies to the security strength for random number generation.

| Date | Minimum of Strength | Symmetric Algorithms | Factoring Modulus | Discrete Logarithm | | Elliptic Curve | Hash (A) | Hash (B) |
|---|---|---|---|---|---|---|---|---|
| | | | | Key | Group | | | |
| (Legacy) | 80 | 2TDEA* | 1024 | 160 | 1024 | 160 | SHA-1** | |
| 2016 - 2030 | 112 | 3TDEA | 2048 | 224 | 2048 | 224 | SHA-224<br>SHA-512/224<br>SHA3-224 | |
| 2016 - 2030 & beyond | 128 | AES-128 | 3072 | 256 | 3072 | 256 | SHA-256<br>SHA-512/256<br>SHA3-256 | SHA-1 |
| 2016 - 2030 & beyond | 192 | AES-192 | 7680 | 384 | 7680 | 384 | SHA-384<br>SHA3-384 | SHA-224<br>SHA-512/224 |
| 2016 - 2030 & beyond | 256 | AES-256 | 15360 | 512 | 15360 | 512 | SHA-512<br>SHA3-512 | SHA-256<br>SHA-512/256<br>SHA-384<br>SHA-512<br>SHA3-512 |