1-Números aleatórios/pseudoaleatórios

Pergunta 1.1

Ao testarmos o comando **head -c 1024 /dev/random — openssl enc -base64** verificámos que o computador fica muito tempo à espera antes que aconteça alguma coisa, isto deve-se ao facto de o comando **/dev/random** necessitar de entropia para poder correr.

Por outro lado, com o comando **head -c 1024 /dev/urandom — openssl enc -base64** vemos que existe uma resposta imediata, isto pelo facto de o comando **/dev/urandom** não necessitar de entropia o que o leva a ser bastante mais rápido.

Pergunta 1.2

Depois de instalado o **haveged** verificámos que ambos os comandos respondem, praticamente, em simultâneo. Isto leva-nos a concluir que este novo pacote cria entropia sem ser necessário estar o próprio utilizador a criá-la, isto é, sem o utilizador estar sempre a utilizar o rato ou a fazer *downloads* etc.

Pergunta 1.3

Ao contrário do programa RandomBytes.java o programa generateSecret-app.py cria apenas caracteres imprimíveis (ASCII), isto, porque a função generateSecret, da API, verifica se cada caractér é imprimível antes de o escolher, ou seja, ela testa se é letra ou dígito e só se for é que são escolhidos.

2-Partilha/Divisão de segredo (Secret Sharing/Splitting)

Pergunta 2.1

A

Primeiro foi necessário gerar um ficheiro do tipo **.pem** (chave privada) utilizando o comando **openssl genrsa -aes128 -out pKey.pem 1024**. Depois utilizou-se o comando **python createSharedSecret-app.py 7 3 1 pKey.pem** (7 partes e quorum de 3). Para recuperar o segredo deve criar-se um certificado utilizando o comando **openssl req -key pKey.pem -new -x509 -days 365 -out pKey.crt** seguido de **python recoverSecretFromComponents-app.py 3 1 pKey.crt**, dando também os componentes pedidos.

B

A diferença é que no **recoverSecretFromComponets** apenas precisamos de dar o número de componentes especificados no quorum (neste caso 3), enquanto que no **recoverSecretFromAllComponents** precisamos de fornecer todos os componentes.

3- Authenticated Encryption

Pergunta 3.1

```
Cifrar( plainText)

secret = cifra( plainText)
mac = hmac( key, secret)

Decifrar(cypherText, receivedMac):

mac = hmac( key, cypherText)
if(mac == receivedMac):
decifra(cypherText, key)
else:
print("not authentic\n")
return -1
```

4-Algoritmos e tamanhos de chaves

Pergunta 4.1

Para a EC *Ministère de la Justice* o certificado *QCert for ESig* mais recente é o *MJL Signature RGS****. Este certificado usa o algoritmo de cifração do RSA com uma chave de 2048 bits. Este tipo de chave prevê-se que só seja segura durante um período curto de tempo e aconselham a que a chave de RSA usada seja do tamanho de 15360 bits.

Para a EC *Imprimerie Nationale* o certificado *QCert for ESig* mais recente é o *Pass'IN - Signature RGS****. Este certificado usa o algoritmo de cifração do RSA com uma chave de 2048 bits. Este tipo de chave prevê-se que só seja segura durante um período curto de tempo e aconselham a que a chave de RSA usada seja do tamanho de 15360 bits.

Para a EC *Certinomis* o certificado *QCert for ESig* mais recente é o *Certinomis - Prime CA*. Este certificado usa o algoritmo de cifração do RSA com uma chave de 4096 bits. Este tipo de chave prevê-se que só seja segura durante um período curto de tempo e aconselham a que a chave de RSA usada seja do tamanho de 15360 bits.