EXERCICIS OPENSSL:

1.-a) Codifica una imatge PNG qualsevol que tinguis disponible al sistema en Base64. Això ho pots fer executant la comanda *openssl enc -e -a -in foto.png -out foto.png.b64* (en aquest exemple la imatge s'anomena "foto.png"). ¿Quin és el contingut de l'arxiu "foto.png.b64" (obre'l amb un editor de text)?

> NOTA: Fixeu-vos que a la comanda emprada no s'ha indicat cap algoritme d'encriptació: només s'ha fet una transformació de format (és a dir, una "codificació"). El mateix es podria haver aconseguit executant *base64 foto.png > foto.png.b64*

b) Elimina uns quants caràcters qualssevol ubicats al mig del contingut de "foto.png.b64" (ha de ser al mig per tal de no modificar el "magic number" del fitxer) i genera la imatge corresponent a partir d'aquest fitxer modificat executant la comanda *openssl enc -d -a -in foto.png.b64 -out novafoto.png* Intenta obrir amb un visor de fotos (per exemple, "eog") el fitxer "novafoto.png". ¿Què veus? ¿Què et diu la comanda *file novafoto.png*?

> NOTA: Una comanda alternativa per "decodificar" de Base64 a binari seria *base64 -d foto.png.b64 > foto.png*

c) Torna a repetir l'apartat *a)* i tot seguit torna a executar, sense que hagis modificat res de "foto.png.b64", la mateixa comanda *openssl* indicada a l'apartat *b).* ¿Què veus ara si obres "nova.foto.png" amb el visor de fotos?

d) Executa la comanda *echo "VmlzY2EgbGEgbXVudGFueWE=" | openssl enc -d -a* ¿Què veus a pantalla i per què? ¿I si ara executes la comanda *echo "Vm1selkyRWdiR0VnYlhWdWRHRnVlV0U9Cg==" | openssl enc -d -a | openssl enc -d -a ?*

dII) Edita un arxiu anomenat "breaker.py" amb el següent contingut i dóna-li permisos d'execució. D'altra banda, crea un altre arxiu anomenat "missatgecodificat.txt" contenint la mateixa cadena emprada a l'apartat anterior (és a dir, sense cometes: "Vm1selkyRWdiR0VnYlhWdWRHRnVlV0U9Cg==") ¿Per a què serveix aquest codi? ¿Quina funció té el bucle allà indicat? ¿Quantes vegades l'hauràs de repetir fins trobar el missatge en clar? ¿Quin és?

```
#!/usr/bin/python3
import base64
fitxer = open("missatgecodificat.txt", "r")
contingut= fitxer.read()
num = int(input("Indica el nombre de cops a fer:"))
for i in range(1,num+1):
    contingut=base64.b64decode(contingut)
    print(contingut)
```

2.-a) Crea un arxiu de tipus "tar.gz" que inclogui la carpeta "~/Imatges" (amb tot el seu contingut); suposarem que representa una còpia de seguretat de les teves fotos que vols emmagatzemar de forma segura. Per crear aquest arxiu "tar.gz" ho pots fer mitjançant la comanda *tar -czf Fotos.tar.gz ~/Imatges* (en aquest exemple l'arxiu "tar.gz" s'anomenarà "Fotos.tar.gz"). Comprova tot seguit que, efectivament, "Fotos.tar.gz" conté el que hauria executant la comanda *tar -tzf Fotos.tar.gz*

b) Xifra l'arxiu "Fotos.tar.gz" amb l'algoritme simètric aes-256-cbc. Això ho pots fer mitjançant la comanda *openssl enc -e -aes-256-cbc -pbkdf2 -in Fotos.tar.gz -out Fotos.tar.gz.enc* (en aquest exemple l'arxiu xifrat s'anomenarà "Fotos.tar.gz.enc"). Indica, com a valor de la "passphrase" que se't demanarà interactivament, la cadena "1234".

bII) Executa ara la comanda *tar -tzf Fotos.tar.gz.enc* ¿Què veus? ¿Què et diu la comanda *file Fotos.tar.gz.enc*?

c) Desxifra l'arxiu "Fotos.tar.gz.enc" mitjançant la comanda *openssl enc -d -aes-256-cbc -pbkdf2 -pass pass:1234 -in Fotos.tar.gz.enc -out FotosFinal.tar.gz* i comprova que, efectivament, "FotosFinal.tar.gz" és equivalent a "Fotos.tar.gz" executant la comanda *tar -tzf FotosFinal.tar.gz*

> NOTA: Els apartats a) i b) es podrien haver fet de cop executant *tar -czf - ~/Imatges | openssl enc -e -aes-256-cbc -pbkdf2 -out Fotos.tar.gz.enc* (noteu el guió per indicar que l'hipotètic fitxer "tar.gz" que empaquetaríem es correspon en realitat a la sortida estàndar). Igualment, es pot dexifrar un arxiu "tar.gz" i desempaquetar-lo a la vegada executant *openssl enc -d -aes-256-cbc -pbkdf2 -pass pass:1234 -in Fotos.tar.gz.enc | tar -xzf -* (noteu el guió per indicar que l'hipotètic fitxer "tar.gz" que desempaquetaríem es correspon en realitat a l'entrada estàndar)

2BIS.-a) Crea un fitxer de text amb un contingut qualsevol (que anomenarem "a.txt"). Ara hauràs de xifrar aquest arxiu de nou simètricament però indicant directament la clau d'encriptació en lloc d'una "passphrase". Per fer això, primer cal que creïs la clau pròpiament dita (és a dir, un valor hexadecimal aleatori) amb la mida adient; com que farem servir l'algoritme AES-256-CBC, la mida haurà de ser de 256 bits (32 bytes). Per tant, executa *openssl rand -hex -out clau.txt 32*

b) D'altra banda, OpenSSL necessita que li aportem explícitament el valor del vector d'inicialització (IV) de l'algoritme emprat (si aquest el necessita, com és el cas) quan indiquem la clau directament (com és el cas) ja que només genera aquest valor automàticament quan fem servir l'opció d'introduïr passphrase (és a dir, quan fem servir una KDF). La mida d'aquest valor en AES sempre és de 128 bits (16 bytes), ja que és la mida del bloc que aquest algoritme fa servir i la mida del IV ha de ser la mateixa. Per tant, executa *openssl rand -hex -out iv.txt 16*

c) Ara sí, xifra "a.txt" així: *openssl enc -e -aes-256-cbc -K $(cat clau.txt) -iv $(cat iv.txt) -a -in a.txt -out b.txt*

d) Esborra l'arxiu "a.txt" i regenera el seu contingut desxifrant l'arxiu "b.txt" obtingut a l'apartat anterior, així: *openssl enc -d -aes-256-cbc -K $(cat clau.txt) -iv $(cat iv.txt) -a -in b.txt*

3.-a) Després de consultar la documentació de https://github.com/glv2/bruteforce-salted-openssl , digues per a què serviria aquesta comanda (amb els paràmetres indicats): *bruteforce-salted-openssl -c aes-256-cbc -l 5 -m 5 -s "0123456789abcdef" -b "00" encrypted.file* ¿I aquesta altra: *bruteforce-salted-openssl -c aes-256-cbc -f dictionary.txt -v 30 -w state.txt encrypted.file* ? Pots provar-ne alguna amb algun fitxer xifrat que tinguis, per jugar

b) És matemàticament impossible saber amb quin algoritme s'ha xifrat un determinat fitxer xifrat. Per tant, si en tinguéssim un i volguéssim desxifrar-lo coneixent el passphrase però sense saber-ne l'algoritme, no tindrem més remei que anar provant un a un els diferents algoritmes que suporta OpenSSL a veure si "sóna la flauta" (i també les possibles funcions de derivació de clau, ja que també caldrà esbrinar quina és l'escollida). En aquest sentit, llegeix el següent article i resumeix breument (en un paràgraf) el procediment que s'hi descriu https://myexperiments.io/finding-cipher-algorithm-encrypted-file.html

4.-a) Crea una clau privada RSA de 2048 bits xifrada (amb el "passphrase" "1234") i anomenada "private.key". Això ho pots aconseguir executant la comanda *openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -aes-256-cbc -out private.key*

aII) Desxifra la clau privada anterior, per tenir-la "a pèl". Això ho pots aconseguir executant la comanda *openssl pkey -in private.key -passin pass:1234 -out newPrivate.key*

aII) Observa executant la comanda *cat private.key* (o també *openssl pkey -in private.key*) la diferència existent en l'encapçalament (i peu) dels dos fitxers, "private.key" i "newPrivate.key". En tot cas, quin format tenen ambdues, PEM o DER?

> NOTA: Si es vol saber no és el contingut sinó la mida d'una clau, es pot executar llavors *openssl pkey -in private.key* ***-text***

b) Genera, a partir de la clau privada que ja tens (tant se val la xifrada -que et preguntará sempre la "passphrase" o la sense xifrar), la corresponent clau pública. Això ho pots aconseguir executant la comanda *openssl pkey -in private.key -pubout -out public.key* (aquí anomenarem "public.key" a aquesta clau pública)

bII) Observa executant la comanda *cat public.key* (o també *openssl pkey -pubin -in public.key*) si el format de la clau generada és PEM o DER.

> NOTA: El paràmetre *-pubin* serveix per indicar que la clau especificada és pública (perquè per defecte o*penssl* sempre entén que la clau especificada és privada)

c) Demana a un company (o fes-ho tu en una altra màquina) que xifri un arxiu que tingui ell (per exemple, "Fotos.tar.gz") amb la nostra clau pública "public.key" (la qual li haurem passat d'alguna manera, no importa com). Això ho podrà fer ell amb la comanda *openssl pkeyutl -encrypt -in Fotos.tar.gz -out Fotos.tar.gz.enc -pubin -inkey public.key*. Un cop fet això (Compte! Si en voler fer l'encriptació apareix l'error "Data too large for key size", passa al següent apartat!), ell t'haurà d'enviar el fitxer xifrat (que suposarem que es diu "Fotos.tar.gz.enc") d'alguna manera (mail, ncat, scp, ...és igual).

A causa de la naturalesa de l'algorisme RSA, aquest només pot xifrar les dades d'entrada que siguin més petites que la mida de la clau RSA. En altres paraules, la mida (nombre de bytes) de les dades d'entrada hauria de ser més petita que la mida (nombre de bytes) de la clau RSA. Si no és així, no podrem realitzar l'apartat anterior i haurem de fer l'apartat següent

cALTERNATIU) Per solventar el problema descrit al paràgraf blau anterior, cal seguir els passos següents i després passar a l'apartat dALTERNATIU):

> *La criptografia simètrica és la única manera de xifrar fitxers grans si no es volen fer servir algoritmes asimètrics de fluxe, més complicats. Així doncs, primer crearem un fitxer que contingui la "passphrase", el qual compartirem més endavant amb l'altre extrem . Una manera ràpida de crear-lo és amb aquesta comanda: *openssl rand -hex -out pass.bin 10* (anomenarem "pass.bin" a aquest fitxer-"passphrase")
>
> *Xifrar el fitxer gran amb aquest fitxer-"passphrase", igual que vam fer a l'exercici 2 , així: *openssl enc -e -aes-256-cbc -pbkdf2 -in Fotos.tar.gz -out Fotos.tar.gz.enc -pass file:./pass.bin*
>
> *Xifrar "pass.bin" amb la clau pública "public.key" del destinatari (per tal de protegir-la, ja que l'haurem de compartir amb el destinatari per un mitjà insegur), així: *openssl pkeyutl -encrypt -in pass.bin -out pass.bin.enc -pubin -inkey public.key* (a més, es recomana destruir completament l'arxiu "pass.bin" per a què ningú el pugui fer servir, així per exemple *shred -u pass.bin*)
>
> *Enviar al destinatari tant el fitxer "Fotos.tar.gz.enc" com "pass.bin.enc" al destinatari.

d) Desxifra l'arxiu "Fotos.tar.gz.enc" que t'han enviat amb la teva clau privada, generada a l'apartat a). Això ho pots fer mitjançant la comanda *openssl pkeyutl -decrypt -in Fotos.tar.gz.enc -out Fotos.tar.gz -inkey private.key* ¿Què obtens?

dALTERNATIU) Només si s'ha fet el cALTERNATIU) Desxifra l'arxiu "pass.bin.enc" que t'han enviat amb la teva clau privada, generada a l'apartat a). Això ho pots fer mitjançant la comanda *openssl pkeyutl -decrypt -in pass.bin.enc -out pass.bin -inkey private.key* . Tot seguit, un cop la clau ha sigut desxifrada amb tecnologia asimètrica, ja podrem desxifrar l'arxiu gros amb aquesta clau, que és simètrica, així: *openssl enc -d -aes-256-cbc -pbkdf2 -in Fotos.tar.gz.enc -out Fotos.tar.gz -pass file:./pass.bin* ¿Què obtens?

5.-a) Crea un fitxer de text amb un contingut qualsevol (que anomenarem "a.txt") i signa'l amb la teva clau privada, així: *openssl pkeyutl -sign -in a.txt -out aSignat.txt -inkey private.key*

b) Envia, de la manera que sigui (ssh, netcat, correu...) la teva clau pública i el fitxer "a-signat.txt" a un company de la classe, el qual haurà d'executar la següent comanda per verificar la signatura i extreure el contingut del fitxer en qüestió: *openssl pkeyutl -verifyrecover -in aSignat.txt -pubin -inkey public.key*

c) Demana al company que torni a intentar verificar la signatura però ara fent servir qualsevol altra clau pública que tingui a mà però que no es correspongui amb la teva. ¿Què passa?

6.-a) Executa la comanda *echo Hola > a.txt* Tot seguit, calcula el hash SHA256 del contingut d'aquest fitxer i guarda'l en un altre fitxer (anomenat "a.txt.dgst") executant la següent comanda *openssl dgst -sha256 -out a.txt.dgst a.txt* ¿Què obtens?

aII) Modifica de qualsevol manera el contingut de l'arxiu "a.txt" i torna a calcular el seu hash SHA256 (mostrant-lo aquest cop per pantalla). ¿Obtens el mateix valor que l'emmagatzemat al fitxer "a.txt.dgst" de l'apartat anterior? ¿Per què?

b) Torna a repetir l'apartat a) però ara signant a més el hash generat amb la clau privada que vas generar a l'apartat a) de l'exercici anterior (o una altra qualsevol, fins i tot nova, és igual). Això ho pots fer executant la comanda *openssl dgst -sha256 -sign private.key -out a.txt.dgst a.txt* ¿El contingut del fitxer "a.txt.dgst" és diferent de l'homònim creat a l'apartat anterior? ¿Amb quin objectiu voldries signar un hash?

> NOTA: Per obtenir una versió en mode text de "a.txt.dgst" (el seu contingut és binari) es pot codificar aquest a Base64.
> NOTA: Si "a.txt" fos un document qualsevol, signant-lo estaríem garantint la seva autenticitat però imagina que "a.txt" fos una clau pública: signant-la també estaríem garantint la seva autenticitat; això és justament el que fan, en essència, els certificats.
> NOTA: Recordeu que una altra manera de garantir l'autenticitat d'un hash (i per tant, la integritat del document associat) és mitjançant la criptografia simètrica i, concretament, xifrant-lo amb una clau compartida per tal de crear un HMAC; el qual no deixa de ser un hash calculat sobre les dades d'entrada combinades amb una clau (és a dir, és el hash d'aquesta combinació dada+clau). Un intermediari que no tingui, doncs, la clau, no podrà calcular aquest hash "protegit".

c) Verifica amb la clau pública "public.key" que vas generar a l'apartat b) de l'exercici anterior (o amb una altra qualsevol, fins i tot nova, és igual) que, efectivament, l'arxiu "a.txt.dgst" no ha sigut alterat. Això ho pots fer executant la comanda *openssl dgst -sha256 -verify public.key -signature a.txt.dgst a.txt* ¿Què veus a pantalla?

cII) Si ara canvies qualsevol caràcter del contingut de l'arxiu "a.txt.dgst" (això ho pots fer simplement modificant-lo amb un editor de text), el guardes amb aquests canvis i tornes a executar la comanda openssl de l'apartat anterior, ¿què veus ara?